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.

Example Language

Syntax of While-programs

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

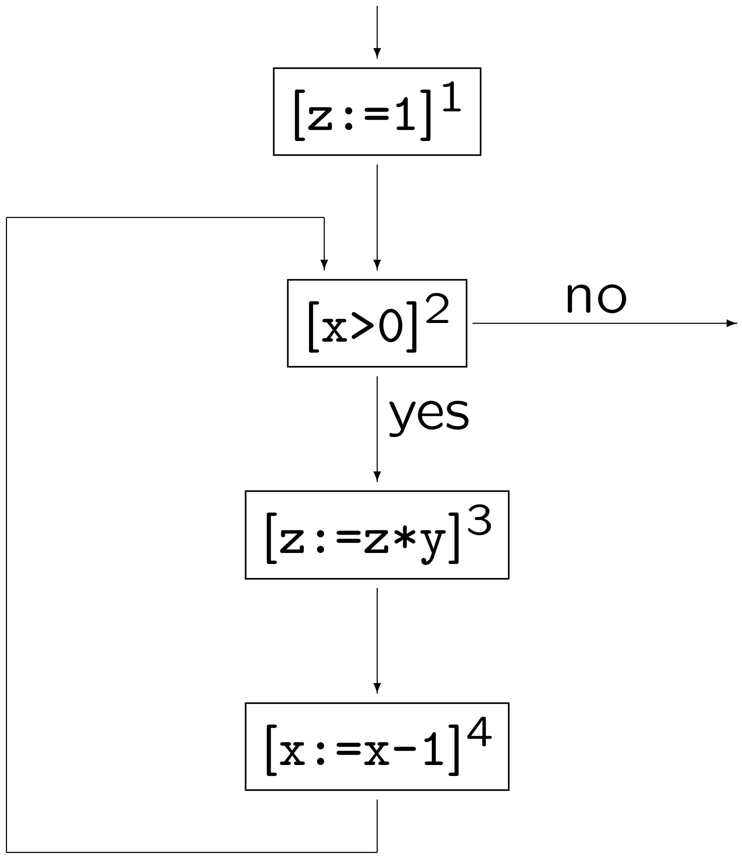
Example: $[\texttt{z:=1}]^1; \texttt{while}\ [\texttt{x>0}]^2\ \texttt{do}\ ([\texttt{z:=z*y}]^3; [\texttt{x:=x-1}]^4)$

Abstract syntax – parentheses are inserted to disambiguate the syntax

Building an "Abstract Flowchart"

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

$$
\begin{aligned}
\mathit{init}(\cdots) &= 1 \\
\mathit{final}(\cdots) &= \{2\} \\
\mathit{labels}(\cdots) &= \{1, 2, 3, 4\} \\
\mathit{flow}(\cdots) &= \{(1,2), (2,3), \\
&\quad\; (3,4), (4,2)\} \\
\mathit{flow}^R(\cdots) &= \{(2,1), (2,4), \\
&\quad\; (3,2), (4,3)\}
\end{aligned}
$$



Initial labels

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

$$\mathit{init} : \mathbf{Stmt} \to \mathbf{Lab}$$

$$\begin{array}{rcl}
\mathit{init}([x := a]^\ell) & = & \ell \\
\mathit{init}([\texttt{skip}]^\ell) & = & \ell \\
\mathit{init}(S_1; S_2) & = & \mathit{init}(S_1) \\
\mathit{init}(\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2) & = & \ell \\
\mathit{init}(\texttt{while } [b]^\ell \texttt{ do } S) & = & \ell
\end{array}$$

Example:

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

Final labels

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

$$\textit{final} : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab})$$

$$
\begin{array}{rcl}
\textit{final}([x := a]^\ell) & = & \{\ell\} \\
\textit{final}([\texttt{skip}]^\ell) & = & \{\ell\} \\
\textit{final}(S_1; S_2) & = & \textit{final}(S_2) \\
\textit{final}(\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2) & = & \textit{final}(S_1) \cup \textit{final}(S_2) \\
\textit{final}(\texttt{while } [b]^\ell \texttt{ do } S) & = & \{\ell\}
\end{array}
$$

Example:

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

Labels

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

$$\textit{labels} : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab})$$

$$\begin{array}{rcl}
\textit{labels}([x := a]^\ell) & = & \{\ell\} \\
\textit{labels}([\texttt{skip}]^\ell) & = & \{\ell\} \\
\textit{labels}(S_1; S_2) & = & \textit{labels}(S_1) \cup \textit{labels}(S_2) \\
\textit{labels}(\texttt{if } [b]^\ell \texttt{ then } S_1 \texttt{ else } S_2) & = & \{\ell\} \cup \textit{labels}(S_1) \cup \textit{labels}(S_2) \\
\textit{labels}(\texttt{while } [b]^\ell \texttt{ do } S) & = & \{\ell\} \cup \textit{labels}(S)
\end{array}$$

Example

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

Flows and reverse flows

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

$$\mathit{flow}, \mathit{flow}^R : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Lab} \times \mathbf{Lab})$$

$$\begin{array}{rcl}
\mathit{flow}([x := a]^\ell) & = & \emptyset \\
\mathit{flow}([\mathtt{skip}]^\ell) & = & \emptyset \\
\mathit{flow}(S_1; S_2) & = & \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \\
 & & \cup\ \{(\ell, \mathit{init}(S_2)) \mid \ell \in \mathit{final}(S_1)\} \\
\mathit{flow}(\mathtt{if}\ [b]^\ell\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2) & = & \mathit{flow}(S_1) \cup \mathit{flow}(S_2) \\
 & & \cup\ \{(\ell, \mathit{init}(S_1)), (\ell, \mathit{init}(S_2))\} \\
\mathit{flow}(\mathtt{while}\ [b]^\ell\ \mathtt{do}\ S) & = & \mathit{flow}(S) \cup \{(\ell, \mathit{init}(S))\} \\
 & & \cup\ \{(\ell', \ell) \mid \ell' \in \mathit{final}(S)\}
\end{array}$$

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

Elementary blocks

A statement consists of a set of *elementary blocks*

$$\textit{blocks} : \mathbf{Stmt} \to \mathcal{P}(\mathbf{Blocks})$$

$$\begin{aligned}
\textit{blocks}([\mathtt{x} := a]^\ell) &= \{[\mathtt{x} := a]^\ell\} \\
\textit{blocks}([\mathtt{skip}]^\ell) &= \{[\mathtt{skip}]^\ell\} \\
\textit{blocks}(S_1; S_2) &= \textit{blocks}(S_1) \cup \textit{blocks}(S_2) \\
\textit{blocks}(\mathtt{if}\ [b]^\ell\ \mathtt{then}\ S_1\ \mathtt{else}\ S_2) &= \{[b]^\ell\} \cup \textit{blocks}(S_1) \cup \textit{blocks}(S_2) \\
\textit{blocks}(\mathtt{while}\ [b]^\ell\ \mathtt{do}\ S) &= \{[b]^\ell\} \cup \textit{blocks}(S)
\end{aligned}$$

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

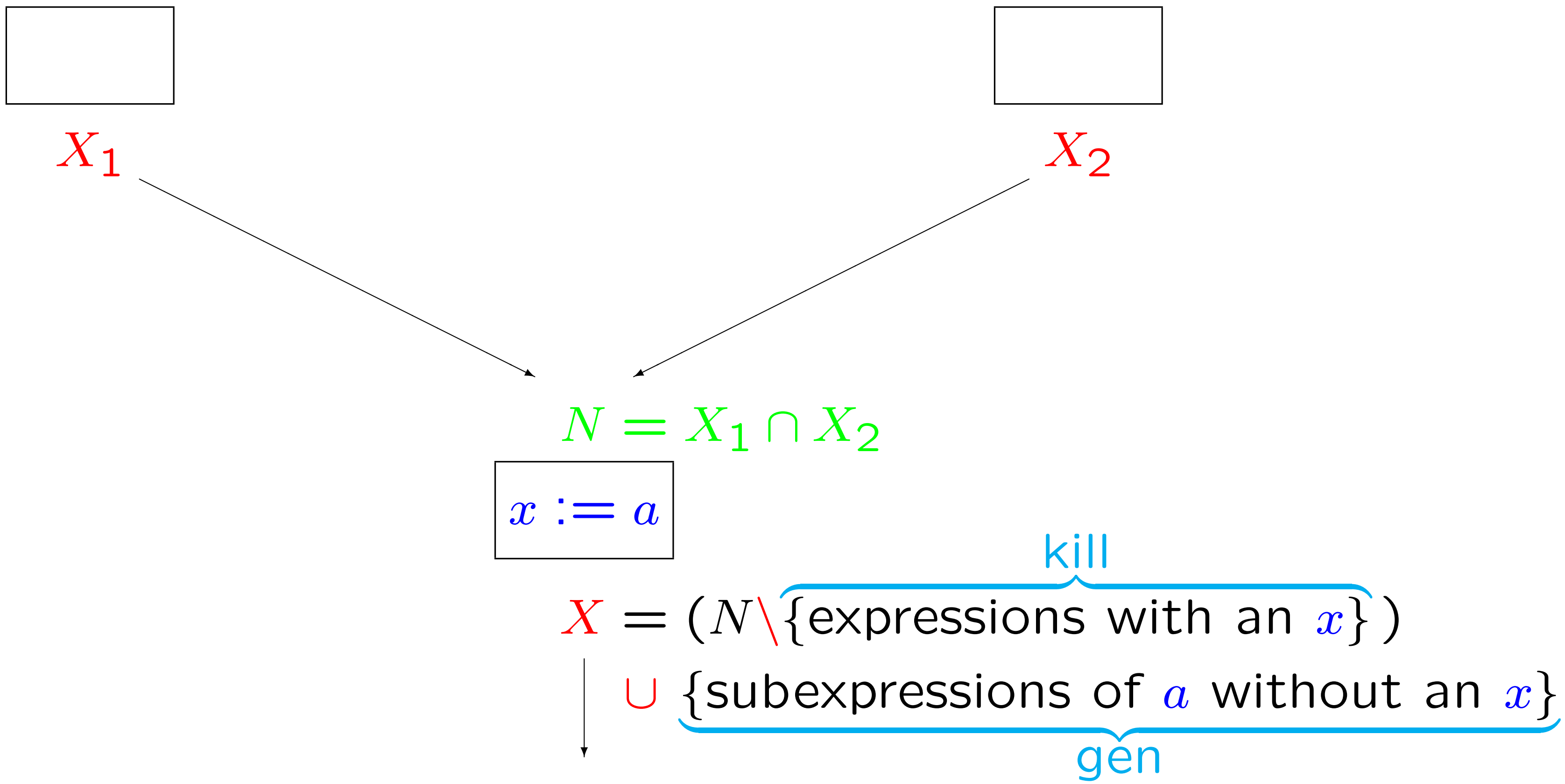
⇓

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

The analysis enables a transformation into

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

Available Expressions Analysis – the basic idea



Available Expressions Analysis

kill and *gen* functions

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

$$\begin{aligned}
gen_{AE}([x := a]^\ell) &= \{a' \in \mathbf{AExp}(a) \mid x \notin FV(a')\} \\
gen_{AE}([\mathtt{skip}]^\ell) &= \emptyset \\
gen_{AE}([b]^\ell) &= \mathbf{AExp}(b)
\end{aligned}$$

data flow equations: $AE^{=}$

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

$$AE_{exit}(\ell) = (AE_{entry}(\ell) \backslash kill_{AE}(B^\ell)) \cup gen_{AE}(B^\ell)$$

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

Example:

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

kill and *gen* functions:

ℓ	$kill_{AE}(\ell)$	$gen_{AE}(\ell)$
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.):

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

Equations:

$$
\begin{aligned}
\mathsf{AE}_{entry}(1) &= \emptyset \\
\mathsf{AE}_{entry}(2) &= \mathsf{AE}_{exit}(1) \\
\mathsf{AE}_{entry}(3) &= \mathsf{AE}_{exit}(2) \cap \mathsf{AE}_{exit}(5) \\
\mathsf{AE}_{entry}(4) &= \mathsf{AE}_{exit}(3) \\
\mathsf{AE}_{entry}(5) &= \mathsf{AE}_{exit}(4)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{AE}_{exit}(1) &= \mathsf{AE}_{entry}(1) \cup \{\texttt{a+b}\} \\
\mathsf{AE}_{exit}(2) &= \mathsf{AE}_{entry}(2) \cup \{\texttt{a*b}\} \\
\mathsf{AE}_{exit}(3) &= \mathsf{AE}_{entry}(3) \cup \{\texttt{a+b}\} \\
\mathsf{AE}_{exit}(4) &= \mathsf{AE}_{entry}(4) \backslash \{\texttt{a+b}, \texttt{a*b}, \texttt{a+1}\} \\
\mathsf{AE}_{exit}(5) &= \mathsf{AE}_{entry}(5) \cup \{\texttt{a+b}\}
\end{aligned}
$$

Example (cont.):

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

Largest solution:

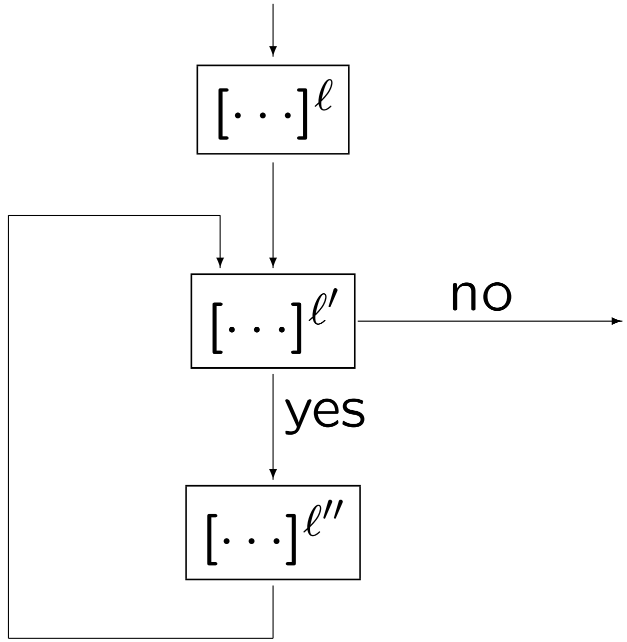
ℓ	$\mathsf{AE}_{entry}(\ell)$	$\mathsf{AE}_{exit}(\ell)$
1	$\emptyset$	$\{\texttt{a+b}\}$
2	$\{\texttt{a+b}\}$	$\{\texttt{a+b}, \texttt{a*b}\}$
3	$\{\texttt{a+b}\}$	$\{\texttt{a+b}\}$
4	$\{\texttt{a+b}\}$	$\emptyset$
5	$\emptyset$	$\{\texttt{a+b}\}$

Why largest solution?

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

Equations:

$$\begin{aligned}
\mathsf{AE}_{entry}(\ell) &= \emptyset \\
\mathsf{AE}_{entry}(\ell') &= \mathsf{AE}_{exit}(\ell)\ \cap\ \mathsf{AE}_{exit}(\ell'') \\
\mathsf{AE}_{entry}(\ell'') &= \mathsf{AE}_{exit}(\ell') \\
\mathsf{AE}_{exit}(\ell) &= \mathsf{AE}_{entry}(\ell) \cup \{\texttt{x+y}\} \\
\mathsf{AE}_{exit}(\ell') &= \mathsf{AE}_{entry}(\ell') \\
\mathsf{AE}_{exit}(\ell'') &= \mathsf{AE}_{entry}(\ell'')
\end{aligned}$$



After some simplification: $\mathsf{AE}_{entry}(\ell') = \{\texttt{x+y}\} \cap \mathsf{AE}_{entry}(\ell')$

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

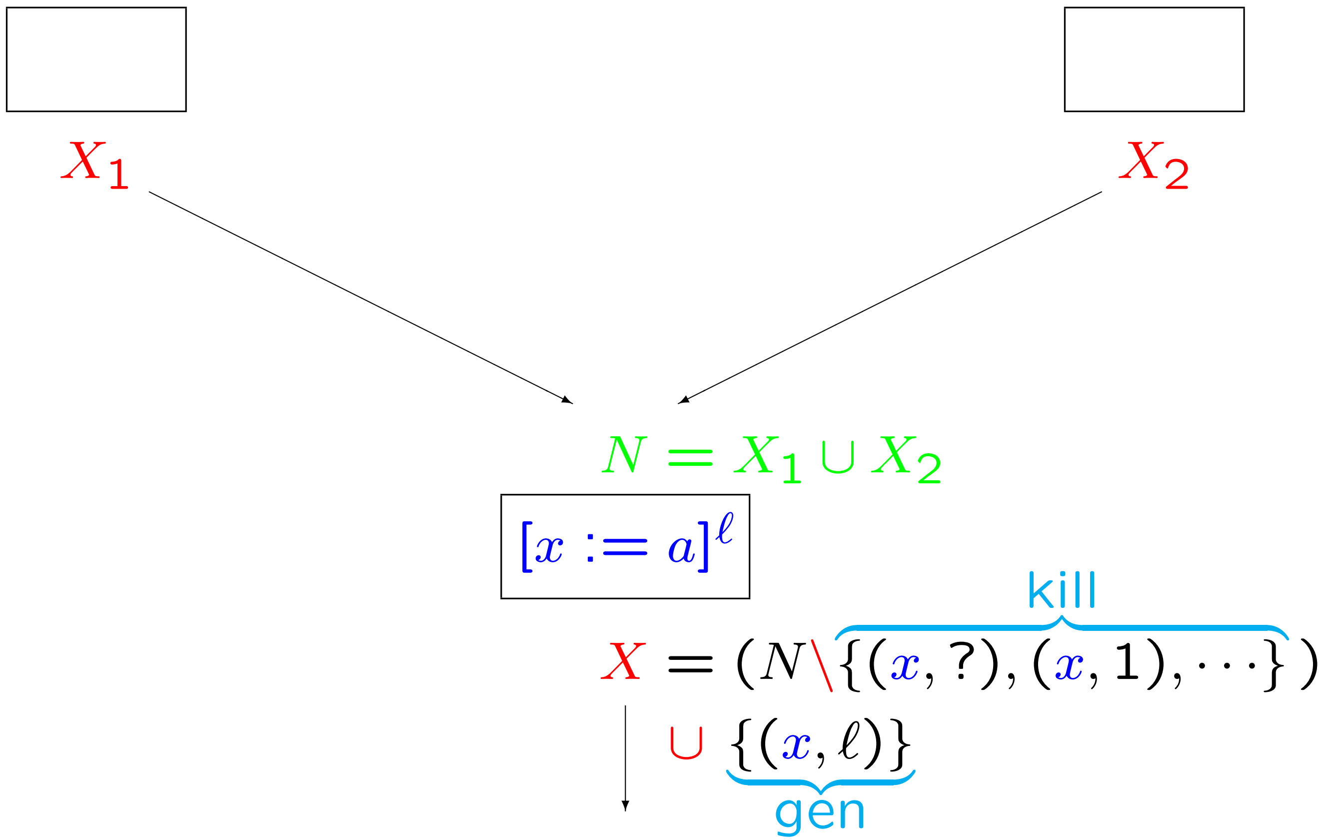
point of interest

$\Downarrow$

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

useful for definition-use chains and use-definition chains

Reaching Definitions Analysis – the basic idea



Reaching Definitions Analysis

kill and *gen* functions

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

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

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

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

Example:

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

kill and *gen* functions:

ℓ	$kill_{\mathsf{RD}}(\ell)$	$gen_{\mathsf{RD}}(\ell)$
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.):

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

Equations:

$$\begin{aligned}
\mathsf{RD}_{entry}(1) &= \{(x,?),(y,?)\}\\
\mathsf{RD}_{entry}(2) &= \mathsf{RD}_{exit}(1)\\
\mathsf{RD}_{entry}(3) &= \mathsf{RD}_{exit}(2) \cup \mathsf{RD}_{exit}(5)\\
\mathsf{RD}_{entry}(4) &= \mathsf{RD}_{exit}(3)\\
\mathsf{RD}_{entry}(5) &= \mathsf{RD}_{exit}(4)
\end{aligned}$$

$$\begin{aligned}
\mathsf{RD}_{exit}(1) &= (\mathsf{RD}_{entry}(1)\backslash\{(x,?),(x,1),(x,5)\}) \cup \{(x,1)\}\\
\mathsf{RD}_{exit}(2) &= (\mathsf{RD}_{entry}(2)\backslash\{(y,?),(y,2),(y,4)\}) \cup \{(y,2)\}\\
\mathsf{RD}_{exit}(3) &= \mathsf{RD}_{entry}(3)\\
\mathsf{RD}_{exit}(4) &= (\mathsf{RD}_{entry}(4)\backslash\{(y,?),(y,2),(y,4)\}) \cup \{(y,4)\}\\
\mathsf{RD}_{exit}(5) &= (\mathsf{RD}_{entry}(5)\backslash\{(x,?),(x,1),(x,5)\}) \cup \{(x,5)\}
\end{aligned}$$

Example (cont.):

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

Smallest solution:

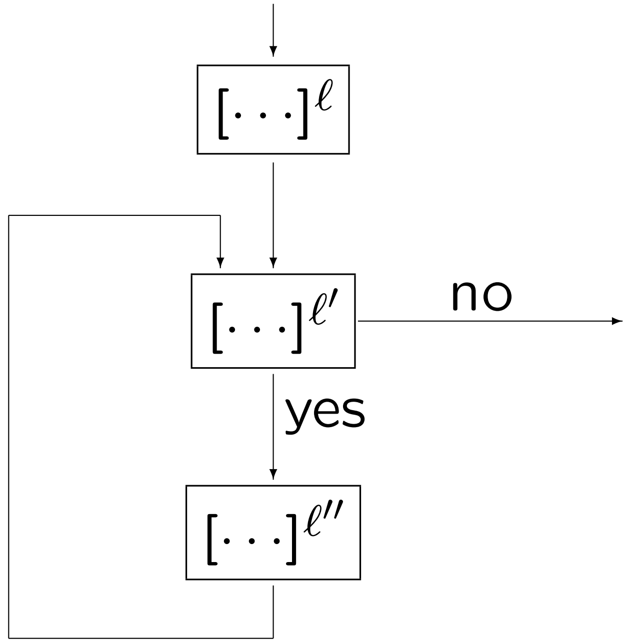
ℓ	$\mathsf{RD}_{entry}(\ell)$	$\mathsf{RD}_{exit}(\ell)$
1	$\{(x,?),(y,?)\}$	$\{(y,?),(x,1)\}$
2	$\{(y,?),(x,1)\}$	$\{(x,1),(y,2)\}$
3	$\{(x,1),(y,2),(y,4),(x,5)\}$	$\{(x,1),(y,2),(y,4),(x,5)\}$
4	$\{(x,1),(y,2),(y,4),(x,5)\}$	$\{(x,1),(y,4),(x,5)\}$
5	$\{(x,1),(y,4),(x,5)\}$	$\{(y,4),(x,5)\}$

Why smallest solution?

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

Equations:

$$\begin{aligned}
\mathsf{RD}_{entry}(\ell) &= \{(\mathtt{x},?),(\mathtt{y},?),(\mathtt{z},?)\}\\
\mathsf{RD}_{entry}(\ell') &= \mathsf{RD}_{exit}(\ell)\cup\mathsf{RD}_{exit}(\ell'')\\
\mathsf{RD}_{entry}(\ell'') &= \mathsf{RD}_{exit}(\ell')\\
\mathsf{RD}_{exit}(\ell) &= (\mathsf{RD}_{entry}(\ell)\setminus\{(\mathtt{z},?)\})\cup\{(\mathtt{z},\ell)\}\\
\mathsf{RD}_{exit}(\ell') &= \mathsf{RD}_{entry}(\ell')\\
\mathsf{RD}_{exit}(\ell'') &= \mathsf{RD}_{entry}(\ell'')
\end{aligned}$$



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

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

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:

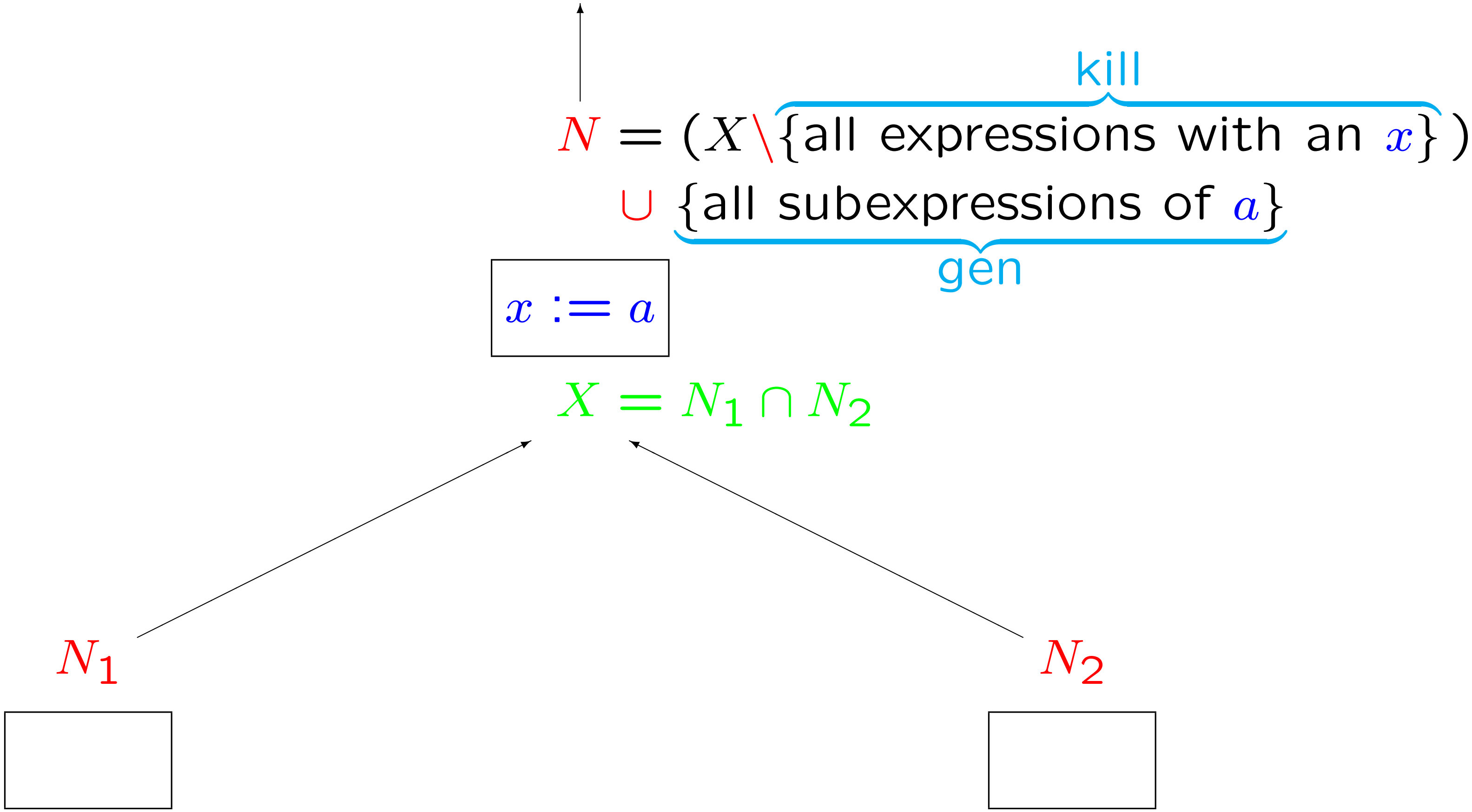
point of interest

⇓ `if` $[\texttt{a>b}]^1$ `then` $([\texttt{x:=b-a}]^2; [\texttt{y:=a-b}]^3)$ `else` $([\texttt{y:=b-a}]^4; [\texttt{x:=a-b}]^5)$

The analysis enables a transformation into

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

Very Busy Expressions Analysis – the basic idea



Very Busy Expressions Analysis

kill and *gen* functions

$$
\begin{aligned}
kill_{VB}([x := a]^\ell) &= \{a' \in \mathbf{AExp}_\star \mid x \in FV(a')\} \\
kill_{VB}([\texttt{skip}]^\ell) &= \emptyset \\
kill_{VB}([b]^\ell) &= \emptyset \\
\\
gen_{VB}([\texttt{x} := a]^\ell) &= \mathbf{AExp}(a) \\
gen_{VB}([\texttt{skip}]^\ell) &= \emptyset \\
gen_{VB}([b]^\ell) &= \mathbf{AExp}(b)
\end{aligned}
$$

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

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

$$
\mathsf{VB}_{entry}(\ell) = (\mathsf{VB}_{exit}(\ell) \backslash kill_{VB}(B^\ell)) \cup gen_{VB}(B^\ell)
$$

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

Example:

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

kill and *gen* function:

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

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:

$$
\begin{aligned}
\mathsf{VB}_{entry}(1) &= \mathsf{VB}_{exit}(1) \\
\mathsf{VB}_{entry}(2) &= \mathsf{VB}_{exit}(2) \cup \{\texttt{b-a}\} \\
\mathsf{VB}_{entry}(3) &= \{\texttt{a-b}\} \\
\mathsf{VB}_{entry}(4) &= \mathsf{VB}_{exit}(4) \cup \{\texttt{b-a}\} \\
\mathsf{VB}_{entry}(5) &= \{\texttt{a-b}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{VB}_{exit}(1) &= \mathsf{VB}_{entry}(2) \cap \mathsf{VB}_{entry}(4) \\
\mathsf{VB}_{exit}(2) &= \mathsf{VB}_{entry}(3) \\
\mathsf{VB}_{exit}(3) &= \emptyset \\
\mathsf{VB}_{exit}(4) &= \mathsf{VB}_{entry}(5) \\
\mathsf{VB}_{exit}(5) &= \emptyset
\end{aligned}
$$

Example (cont.):

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

Largest solution:

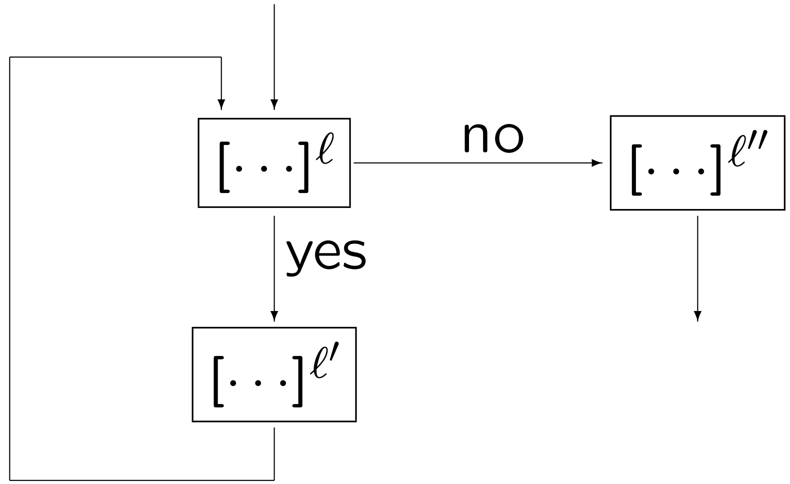
ℓ	$\mathsf{VB}_{entry}(\ell)$	$\mathsf{VB}_{exit}(\ell)$
1	{a-b, b-a}	{a-b, b-a}
2	{a-b, b-a}	{a-b}
3	{a-b}	$\emptyset$
4	{a-b, b-a}	{a-b}
5	{a-b}	$\emptyset$

Why largest solution?

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

Equations:

$$\begin{aligned}
\mathsf{VB}_{entry}(\ell) &= \mathsf{VB}_{exit}(\ell) \\
\mathsf{VB}_{entry}(\ell') &= \mathsf{VB}_{exit}(\ell') \\
\mathsf{VB}_{entry}(\ell'') &= \{\texttt{x+1}\} \\
\mathsf{VB}_{exit}(\ell) &= \mathsf{VB}_{entry}(\ell') \cap \mathsf{VB}_{entry}(\ell'') \\
\mathsf{VB}_{exit}(\ell') &= \mathsf{VB}_{entry}(\ell) \\
\mathsf{VB}_{exit}(\ell'') &= \emptyset
\end{aligned}$$



After some simplifications: $\mathsf{VB}_{exit}(\ell) = \mathsf{VB}_{exit}(\ell) \cap \{\texttt{x+1}\}$

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

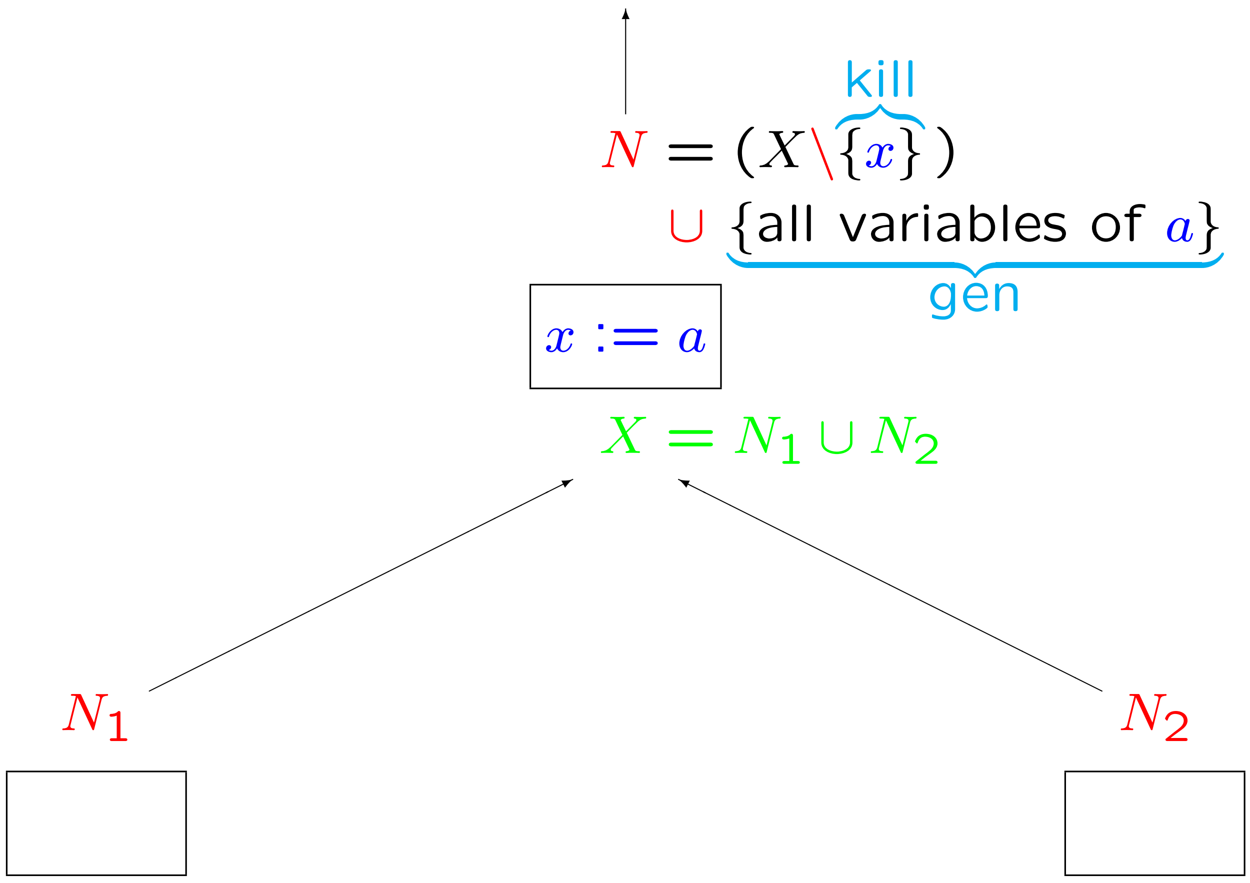
⇓

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

The analysis enables a transformation into

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

Live Variables Analysis – the basic idea



Live Variables Analysis

kill and *gen* functions

$$\begin{aligned}
kill_{\mathsf{LV}}([x := a]^\ell) &= \{x\} \\
kill_{\mathsf{LV}}([\texttt{skip}]^\ell) &= \emptyset \\
kill_{\mathsf{LV}}([b]^\ell) &= \emptyset \\
\\
gen_{\mathsf{LV}}([x := a]^\ell) &= FV(a) \\
gen_{\mathsf{LV}}([\texttt{skip}]^\ell) &= \emptyset \\
gen_{\mathsf{LV}}([b]^\ell) &= FV(b)
\end{aligned}$$

data flow equations: $\mathsf{LV}^=$

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

$$\mathsf{LV}_{entry}(\ell) = (\mathsf{LV}_{exit}(\ell) \backslash kill_{\mathsf{LV}}(B^\ell)) \cup gen_{\mathsf{LV}}(B^\ell)$$

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

Example:

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

kill and *gen* functions:

ℓ	$kill_{\mathsf{LV}}(\ell)$	$gen_{\mathsf{LV}}(\ell)$
1	$\{\mathtt{x}\}$	$\emptyset$
2	$\{\mathtt{y}\}$	$\emptyset$
3	$\{\mathtt{x}\}$	$\emptyset$
4	$\emptyset$	$\{\mathtt{x}, \mathtt{y}\}$
5	$\{\mathtt{z}\}$	$\{\mathtt{y}\}$
6	$\{\mathtt{z}\}$	$\{\mathtt{y}\}$
7	$\{\mathtt{x}\}$	$\{\mathtt{z}\}$

Example (cont.):

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

Equations:

$$
\begin{aligned}
\mathsf{LV}_{entry}(1) &= \mathsf{LV}_{exit}(1)\backslash\{\texttt{x}\} & \mathsf{LV}_{exit}(1) &= \mathsf{LV}_{entry}(2)\\
\mathsf{LV}_{entry}(2) &= \mathsf{LV}_{exit}(2)\backslash\{\texttt{y}\} & \mathsf{LV}_{exit}(2) &= \mathsf{LV}_{entry}(3)\\
\mathsf{LV}_{entry}(3) &= \mathsf{LV}_{exit}(3)\backslash\{\texttt{x}\} & \mathsf{LV}_{exit}(3) &= \mathsf{LV}_{entry}(4)\\
\mathsf{LV}_{entry}(4) &= \mathsf{LV}_{exit}(4) \cup \{\texttt{x},\texttt{y}\} & \mathsf{LV}_{exit}(4) &= \mathsf{LV}_{entry}(5) \cup \mathsf{LV}_{entry}(6)\\
\mathsf{LV}_{entry}(5) &= (\mathsf{LV}_{exit}(5)\backslash\{\texttt{z}\}) \cup \{\texttt{y}\} & \mathsf{LV}_{exit}(5) &= \mathsf{LV}_{entry}(7)\\
\mathsf{LV}_{entry}(6) &= (\mathsf{LV}_{exit}(6)\backslash\{\texttt{z}\}) \cup \{\texttt{y}\} & \mathsf{LV}_{exit}(6) &= \mathsf{LV}_{entry}(7)\\
\mathsf{LV}_{entry}(7) &= \{\texttt{z}\} & \mathsf{LV}_{exit}(7) &= \emptyset
\end{aligned}
$$

Example (cont.):

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

Smallest solution:

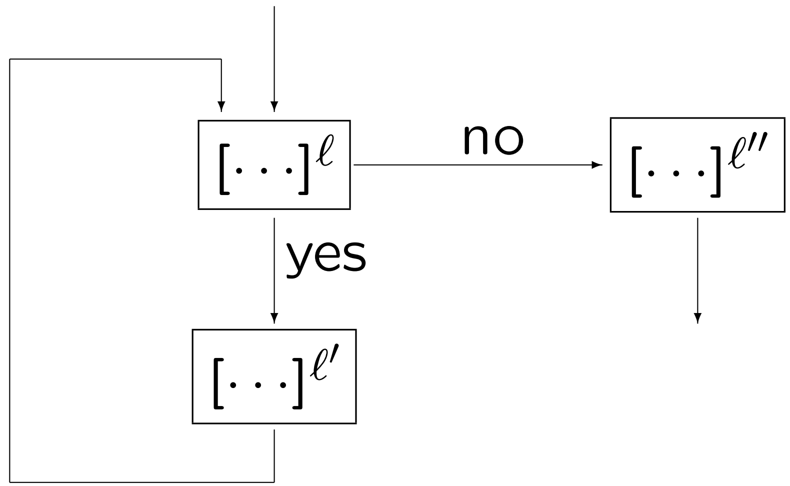
ℓ	$\mathsf{LV}_{entry}(\ell)$	$\mathsf{LV}_{exit}(\ell)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{\mathtt{y}\}$
3	$\{\mathtt{y}\}$	$\{\mathtt{x},\mathtt{y}\}$
4	$\{\mathtt{x},\mathtt{y}\}$	$\{\mathtt{y}\}$
5	$\{\mathtt{y}\}$	$\{\mathtt{z}\}$
6	$\{\mathtt{y}\}$	$\{\mathtt{z}\}$
7	$\{\mathtt{z}\}$	$\emptyset$

Why smallest solution?

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

Equations:

$$\begin{aligned}
\mathsf{LV}_{entry}(\ell) &= \mathsf{LV}_{exit}(\ell) \cup \{\texttt{x}\}\\
\mathsf{LV}_{entry}(\ell') &= \mathsf{LV}_{exit}(\ell')\\
\mathsf{LV}_{entry}(\ell'') &= \{\texttt{x}\}\\
\mathsf{LV}_{exit}(\ell) &= \mathsf{LV}_{entry}(\ell') \cup \mathsf{LV}_{entry}(\ell'')\\
\mathsf{LV}_{exit}(\ell') &= \mathsf{LV}_{entry}(\ell)\\
\mathsf{LV}_{exit}(\ell'') &= \emptyset
\end{aligned}$$



After some calculations: $\mathsf{LV}_{exit}(\ell) = \mathsf{LV}_{exit}(\ell) \cup \{\texttt{x}\}$

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

Derived Data Flow Information

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

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

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

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

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

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

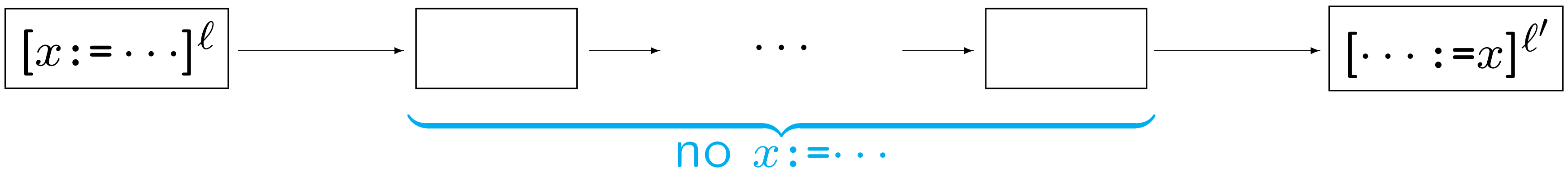
ud chains

$$ud : \mathbf{Var}_\star \times \mathbf{Lab}_\star \to \mathcal{P}(\mathbf{Lab}_\star)$$

given by

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

where



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

ud chains - an alternative definition

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

is defined by:

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

One can show that:

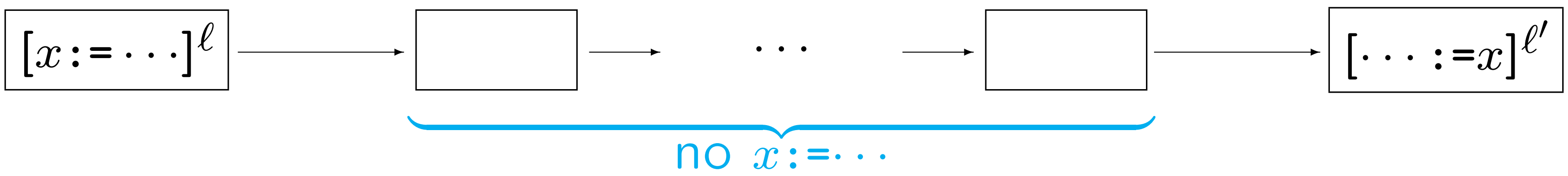
$$\mathit{ud}(x,\ell) = \mathsf{UD}(x,\ell)$$

du chains

$$du : \mathbf{Var}_\star \times \mathbf{Lab}_\star \to \mathcal{P}(\mathbf{Lab}_\star)$$

given by

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



One can show that:

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

Example:

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

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

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

$$\sigma \in \mathbf{State} \;=\; \mathbf{Var} \to \mathbf{Z}$$

The semantics of arithmetic and boolean expressions

$$\mathcal{A} : \;\; \mathbf{AExp} \to (\mathbf{State} \to \mathbf{Z}) \qquad \text{(no errors allowed)}$$

$$\mathcal{B} : \;\; \mathbf{BExp} \to (\mathbf{State} \to \mathbf{T}) \qquad \text{(no errors allowed)}$$

The *transitions* of the semantics are of the form

$$\langle S, \sigma \rangle \to \sigma' \qquad \text{and} \qquad \langle S, \sigma \rangle \to \langle S', \sigma' \rangle$$

Transitions

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

$$\langle [\texttt{skip}]^\ell, \sigma\rangle \to \sigma$$

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

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

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

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

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

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

Example:

$$\langle [\texttt{y:=x}]^1; [\texttt{z:=1}]^2; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{300}\rangle$$

$$\rightarrow \langle [\texttt{z:=1}]^2; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{330}\rangle$$

$$\rightarrow \langle \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{331}\rangle$$

$$\rightarrow \langle [\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{331}\rangle$$

$$\rightarrow \langle [\texttt{y:=y-1}]^5; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{333}\rangle$$

$$\rightarrow \langle \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{323}\rangle$$

$$\rightarrow \langle [\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{323}\rangle$$

$$\rightarrow \langle [\texttt{y:=y-1}]^5; \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{326}\rangle$$

$$\rightarrow \langle \texttt{while } [\texttt{y>1}]^3 \texttt{ do } ([\texttt{z:=z*y}]^4; [\texttt{y:=y-1}]^5); [\texttt{y:=0}]^6, \sigma_{316}\rangle$$

$$\rightarrow \langle [\texttt{y:=0}]^6, \sigma_{316}\rangle$$

$$\rightarrow \sigma_{306}$$

Equations and Constraints

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

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

$$\mathsf{LV}_{entry}(\ell) \quad = \quad (\mathsf{LV}_{exit}(\ell) \backslash \mathit{kill}_{\mathsf{LV}}(B^\ell)) \cup \mathit{gen}_{\mathsf{LV}}(B^\ell)$$

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

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

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

$$\mathsf{LV}_{entry}(\ell) \quad \supseteq \quad (\mathsf{LV}_{exit}(\ell) \backslash \mathit{kill}_{\mathsf{LV}}(B^\ell)) \cup \mathit{gen}_{\mathsf{LV}}(B^\ell)$$

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

Lemma

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

Proof: Trivial.

Lemma

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

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

$$\langle S, \sigma_1 \rangle \quad \rightarrow \quad \langle S', \sigma_1' \rangle \quad \rightarrow \quad \cdots \quad \rightarrow \quad \langle S'', \sigma_1'' \rangle \quad \rightarrow \quad \sigma_1'''$$

$$\updownarrow \models \mathsf{LV}^{\subseteq} \qquad\qquad \updownarrow \models \mathsf{LV}^{\subseteq} \qquad\qquad\qquad\qquad \updownarrow \models \mathsf{LV}^{\subseteq}$$

$$\mathit{live} \qquad\qquad\qquad \mathit{live} \qquad\qquad\qquad \cdots \qquad\qquad \mathit{live}$$

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 σ_1 and σ_2 are equal: only the values of the live variables of V matters and here the two states are equal.

Example:

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

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

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

Correctness Theorem

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

$$
\begin{array}{ccccccccc}
\langle S, \sigma_1 \rangle & \to & \langle S', \sigma_1' \rangle & \to & \cdots & \to & \langle S'', \sigma_1'' \rangle & \to & \sigma_1''' \\
\updownarrow \sim_V & & \updownarrow \sim_{V'} & & & & \updownarrow \sim_{V''} & & \updownarrow \sim_{V'''} \\
\langle S, \sigma_2 \rangle & \to & \langle S', \sigma_2' \rangle & \to & \cdots & \to & \langle S'', \sigma_2'' \rangle & \to & \sigma_2'''
\end{array}
$$

$V = \mathit{live}_{entry}(\mathit{init}(S))$

$V'' = \mathit{live}_{entry}(\mathit{init}(S''))$

$V' = \mathit{live}_{entry}(\mathit{init}(S'))$

$$
\begin{aligned}
V''' &= \mathit{live}_{exit}(\mathit{init}(S'')) \\
&= \mathit{live}_{exit}(\ell)
\end{aligned}
$$

for some $\ell \in \mathit{final}(S)$

Monotone Frameworks

- Monotone and Distributive Frameworks

- Instances of Frameworks

- Constant Propagation Analysis

The Overall Pattern

Each of the four classical analyses take the form

$$\begin{array}{rcl} \mathit{Analysis}_\circ(\ell) & = & \begin{cases} \iota & \text{if } \ell \in E \\ \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\} & \text{otherwise} \end{cases} \\ \mathit{Analysis}_\bullet(\ell) & = & f_\ell(\mathit{Analysis}_\circ(\ell)) \end{array}$$

where

- $\bigsqcup$ is $\bigcap$ or $\bigcup$ (and $\sqcup$ is $\cup$ or $\cap$),
- F is either $\mathit{flow}(S_\star)$ or $\mathit{flow}^R(S_\star)$,
- E is $\{\mathit{init}(S_\star)\}$ or $\mathit{final}(S_\star)$,
- ι specifies the initial or final analysis information, and
- f_ℓ is the transfer function associated with $B^\ell \in \mathit{blocks}(S_\star)$.

The Principle: forward versus backward

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

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

The Principle: union versus intersecton

- When $\bigsqcup$ is $\bigcap$ 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 $\bigsqcup$ is $\bigcup$ 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*, $\bigsqcup$: $\mathcal{P}(L) \rightarrow L$, 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, $\bigsqcup Y$.

- L satisfies the *Ascending Chain Condition*; that is, each ascending chain eventually stabilises (meaning that if $(l_n)_n$ is such that $l_1 \sqsubseteq l_2 \sqsubseteq l_3 \sqsubseteq \cdots$,then there exists n such that $l_n = l_{n+1} = \cdots$).

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 $\bigsqcup$ is $\bigcup$ and the least element $\bot$ 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 $\bigsqcup$ is $\bigcap$ and the least element $\bot$ 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 $f_\ell : L \to L$ 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 $\bigsqcup$ 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 *flow*$(S_\star)$ or *flow*$^R(S_\star)$)

– a finite set of *extremal labels*, E (typically $\{$*init*$(S_\star)\}$ or *final*$(S_\star)$)

– an *extremal value*, $\iota \in L$, for the extremal labels

– a mapping, $f_\cdot$, from the labels $\mathbf{Lab}_\star$ to transfer functions in $\mathcal{F}$

Equations of the Instance:

$$\mathit{Analysis}_\circ(\ell) = \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\} \sqcup \iota_E^\ell$$

$$\text{where } \iota_E^\ell = \begin{cases} \iota & \text{if } \ell \in E \\ \bot & \text{if } \ell \notin E \end{cases}$$

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

Constraints of the Instance:

$$\mathit{Analysis}_\circ(\ell) \sqsupseteq \bigsqcup\{\mathit{Analysis}_\bullet(\ell') \mid (\ell',\ell) \in F\} \sqcup \iota_E^\ell$$

$$\text{where } \iota_E^\ell = \begin{cases} \iota & \text{if } \ell \in E \\ \bot & \text{if } \ell \notin E \end{cases}$$

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

The Examples Revisited

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

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

$$
\begin{array}{lcl}
f(l_1 \sqcup l_2) & = & \left\{\begin{array}{l} f(l_1 \cup l_2) \\ f(l_1 \cap l_2) \end{array}\right. \quad = \left\{\begin{array}{l} ((l_1 \cup l_2) \setminus l_k) \cup l_g \\ ((l_1 \cap l_2) \setminus l_k) \cup l_g \end{array}\right. \\
& = & \left\{\begin{array}{l} ((l_1 \setminus l_k) \cup (l_2 \setminus l_k)) \cup l_g \\ ((l_1 \setminus l_k) \cap (l_2 \setminus l_k)) \cup l_g \end{array}\right. \quad = \left\{\begin{array}{l} ((l_1 \setminus l_k) \cup l_g) \cup ((l_2 \setminus l_k) \cup l_g) \\ ((l_1 \setminus l_k) \cup l_g) \cap ((l_2 \setminus l_k) \cup l_g) \end{array}\right. \\
& = & \left\{\begin{array}{l} f(l_1) \cup f(l_2) \\ f(l_1) \cap f(l_2) \end{array}\right. \quad = \quad f(l_1) \sqcup f(l_2)
\end{array}
$$

- $id(l) = (l \setminus \emptyset) \cup \emptyset$
- $f_2(f_1(l)) = (((l \setminus l_k^1) \cup l_g^1) \setminus l_k^2) \cup l_g^2 = (l \setminus (l_k^1 \cup l_k^2)) \cup ((l_g^1 \setminus l_k^2) \cup l_g^2)$
- 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:

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

The analysis enables a transformation into

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

Elements of L

$$\widehat{\mathbf{State}}_{\mathsf{CP}} = ((\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot, \sqsubseteq)$$

Idea:

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

Partial Ordering on L

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

$$\forall \widehat{\sigma} \in (\mathbf{Var}_\star \to \mathbf{Z}^\top)_\bot : \quad \bot \sqsubseteq \widehat{\sigma}$$

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

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 $\mathcal{F}$

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

Lemma

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

Instances

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

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

Constant Propagation Analysis

$$\mathcal{A}_{\mathsf{CP}} : \mathbf{AExp} \to (\widehat{\mathbf{State}}_{\mathsf{CP}} \to \mathbf{Z}^{\top}_{\bot})$$

$$\mathcal{A}_{\mathsf{CP}}[\![x]\!]\widehat{\sigma} = \begin{cases} \bot & \text{if } \widehat{\sigma} = \bot \\ \widehat{\sigma}(x) & \text{otherwise} \end{cases}$$

$$\mathcal{A}_{\mathsf{CP}}[\![n]\!]\widehat{\sigma} = \begin{cases} \bot & \text{if } \widehat{\sigma} = \bot \\ n & \text{otherwise} \end{cases}$$

$$\mathcal{A}_{\mathsf{CP}}[\![a_1 \; op_a \; a_2]\!]\widehat{\sigma} = \mathcal{A}_{\mathsf{CP}}[\![a_1]\!]\widehat{\sigma} \; \widehat{\mathsf{op}}_a \; \mathcal{A}_{\mathsf{CP}}[\![a_2]\!]\widehat{\sigma}$$

transfer functions: f^{CP}_{ℓ}

$$[x := a]^{\ell} : \quad f^{\mathsf{CP}}_{\ell}(\widehat{\sigma}) = \begin{cases} \bot & \text{if } \widehat{\sigma} = \bot \\ \widehat{\sigma}[x \mapsto \mathcal{A}_{\mathsf{CP}}[\![a]\!]\widehat{\sigma}] & \text{otherwise} \end{cases}$$

$$[\texttt{skip}]^{\ell} : \quad f^{\mathsf{CP}}_{\ell}(\widehat{\sigma}) = \widehat{\sigma}$$

$$[b]^{\ell} : \quad f^{\mathsf{CP}}_{\ell}(\widehat{\sigma}) = \widehat{\sigma}$$

Lemma

Constant Propagation is not a Distributive Framework

Proof

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

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

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

Both $f_\ell^{\mathsf{CP}}(\widehat{\sigma}_1)$ and $f_\ell^{\mathsf{CP}}(\widehat{\sigma}_2)$ map $\mathtt{y}$ to 1 — $f_\ell^{\mathsf{CP}}(\widehat{\sigma}_1) \sqcup f_\ell^{\mathsf{CP}}(\widehat{\sigma}_2)$ maps $\mathtt{y}$ 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 $(L, \mathcal{F}, F, E, \iota, f_\cdot)$ 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 (ℓ, ℓ') indicating that the current analysis result has changed at the entry (or exit) to the block ℓ and hence the entry (or exit) information must be recomputed for ℓ'

Worklist Algorithm

Step 1 Initialisation (of W and Analysis)

W := nil;
for all (ℓ, ℓ') in F do W := cons((ℓ, ℓ'),W);
for all ℓ in F or E do
  if $\ell \in E$ then Analysis[ℓ] := ι else Analysis[ℓ] := $\bot_L$;

Step 2 Iteration (updating W and Analysis)

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

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

for all ℓ in F or E do
  $MFP_\circ(\ell)$:= Analysis[ℓ];
  $MFP_\bullet(\ell)$:= f_ℓ(Analysis[ℓ])

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_ℓ, 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 ℓ:

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

The paths up to and including ℓ:

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

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

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

The MOP Solution

The solution up to but not including ℓ:

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

The solution up to and including ℓ:

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

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 $(L, \mathcal{F}, F, B, \iota, f_\cdot)$ 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 ℓ 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 $u_1, \cdots, u_n$ and $v_1, \cdots, v_n$ be strings over the alphabet {1,···,9}; let $|\ u\ |$ denote the length of u; let $[\![u]\!]$ be the natural number denoted.

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

```
x:=[[u1]]; y:=[[v1]];
while [···] do
   (if [···] then x:=x * 10^|u1| + [[u1]]; y:=y * 10^|v1| + [[v1]] else
    ⋮
    if [···] then x:=x * 10^|un| + [[un]]; y:=y * 10^|vn| + [[vn]] else skip)
[z:=abs((x-y)*(x-y))]^ℓ
```

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

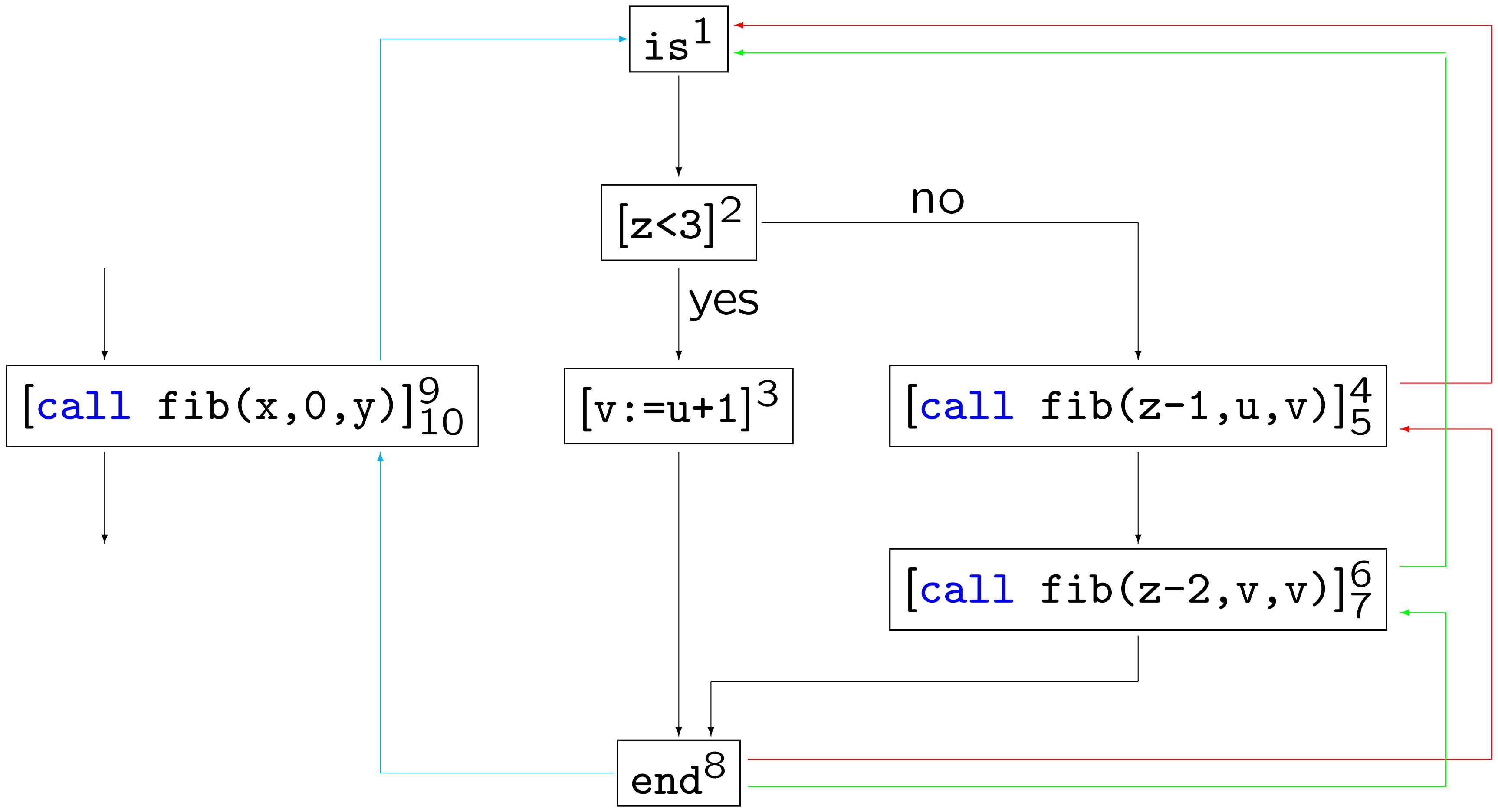
Interprocedural Analysis

- 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: $P_\star = \texttt{begin}\ D_\star\ S_\star\ \texttt{end}$

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

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

Example:

$$
\begin{array}{l}
\texttt{begin}\ \ \texttt{proc fib(val z, u; res v) is}^{1} \\
\qquad\qquad\quad \texttt{if [z<3]}^{2}\ \texttt{then [v:=u+1]}^{3} \\
\qquad\qquad\quad \texttt{else ([call fib(z-1,u,v)]}^{4}_{5}\texttt{; [call fib(z-2,v,v)]}^{6}_{7}\texttt{)} \\
\qquad\quad \texttt{end}^{8}\texttt{;} \\
\qquad\quad \texttt{[call fib(x,0,y)]}^{9}_{10} \\
\texttt{end}
\end{array}
$$

Flow graphs for procedure calls

$$
\begin{array}{rcl}
\mathit{init}([\texttt{call}\ p(a,z)]_{\ell_r}^{\ell_c}) & = & \ell_c \\
\mathit{final}([\texttt{call}\ p(a,z)]_{\ell_r}^{\ell_c}) & = & \{\ell_r\} \\
\mathit{blocks}([\texttt{call}\ p(a,z)]_{\ell_r}^{\ell_c}) & = & \{[\texttt{call}\ p(a,z)]_{\ell_r}^{\ell_c}\} \\
\mathit{labels}([\texttt{call}\ p(a,z)]_{\ell_r}^{\ell_c}) & = & \{\ell_c, \ell_r\} \\
\mathit{flow}([\texttt{call}\ p(a,z)]_{\ell_r}^{\ell_c}) & = & \{(\ell_c;\ell_n), (\ell_x;\ell_r)\} \\
 & \text{if} & \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}\ \text{is in}\ D_\star
\end{array}
$$

- $(\ell_c;\ell_n)$ is the flow corresponding to *calling* a procedure at ℓ_c and entering the procedure body at ℓ_n, and

- $(\ell_x;\ell_r)$ is the flow corresponding to exiting a procedure body at ℓ_x and *returning* to the call at ℓ_r.

Flow graphs for procedure declarations

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

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

Flow graphs for programs

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

$$
\begin{array}{rcl}
\mathit{init}_\star & = & \mathit{init}(S_\star) \\
\mathit{final}_\star & = & \mathit{final}(S_\star) \\
\mathit{blocks}_\star & = & \bigcup\{\mathit{blocks}(p) \mid \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}\ \text{is in}\ D_\star\} \\
 & & \cup\,\mathit{blocks}(S_\star) \\
\mathit{labels}_\star & = & \bigcup\{\mathit{labels}(p) \mid \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}\ \text{is in}\ D_\star\} \\
 & & \cup\,\mathit{labels}(S_\star) \\
\mathit{flow}_\star & = & \bigcup\{\mathit{flow}(p) \mid \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}\ \text{is in}\ D_\star\} \\
 & & \cup\,\mathit{flow}(S_\star) \\
\mathit{interflow}_\star & = & \{(\ell_c, \ell_n, \ell_x, \ell_r) \mid \texttt{proc}\ p(\texttt{val}\ x; \texttt{res}\ y)\ \texttt{is}^{\ell_n}\ S\ \texttt{end}^{\ell_x}\ \text{is in}\ D_\star \\
 & & \qquad\qquad\qquad \text{and}\ [\texttt{call}\ p(a, z)]^{\ell_c}_{\ell_r}\ \text{is in}\ S_\star\}
\end{array}
$$

Example:

```
begin  proc fib(val z, u; res v) is^1
             if [z<3]^2 then [v:=u+1]^3
             else ([call fib(z-1,u,v)]^4_5; [call fib(z-2,v,v)]^6_7)
       end^8;
       [call fib(x,0,y)]^9_10
end
```

We have

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

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

A naive formulation

Treat the three kinds of flow in the same way:

flow	treat as
(ℓ_1, ℓ_2)	(ℓ_1, ℓ_2)
$(\ell_c; \ell_n)$	(ℓ_c, ℓ_n)
$(\ell_x; \ell_r)$	(ℓ_x, ℓ_r)

Equation system:

$$\begin{array}{rcl} A_\bullet(\ell) & = & f_\ell(A_\circ(\ell)) \\ A_\circ(\ell) & = & \bigsqcup\{A_\bullet(\ell') \mid (\ell', \ell) \in F \text{ or } (\ell', \ell) \in F \text{ or } (\ell', \ell) \in F\} \sqcup \iota_E^\ell \end{array}$$

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 ℓ_1 to ℓ_2 in $P_\star$ has proper nesting of procedure entries and exits; and a procedure returns to the point where it was called:

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

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

Valid Paths

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

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

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

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

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

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 $(L, \mathcal{F}, F, E, \iota, f_\cdot)$ of a Monotone Framework

- the analysis is forwards, i.e. $F = \mathit{flow}_\star$ and $E = \{\mathit{init}_\star\}$;

- the complete lattice is a powerset, i.e. $L = \mathcal{P}(D)$;

- the transfer functions in $\mathcal{F}$ are completely additive; and

- each f_ℓ is given by $f_\ell(Y) = \bigcup\{\phi_\ell(d) \mid d \in Y\}$ where $\phi_\ell : D \to \mathcal{P}(D)$.

(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'_ℓ 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

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

for all labels (including those that label procedure calls)

Example:

Detection of Signs Analysis as a Monotone Framework:

$(L_{\mathsf{sign}}, \mathcal{F}_{\mathsf{sign}}, F, E, \iota_{\mathsf{sign}}, f_{\cdot}^{\mathsf{sign}})$ where $\mathbf{Sign} = \{\texttt{-}, \texttt{0}, \texttt{+}\}$ and

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

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

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

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

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

Example (cont.):

Detection of Signs Analysis as an embellished monotone framework

$$L'_{\mathsf{sign}} = \mathcal{P}(\boldsymbol{\Delta} \times \mathbf{(Var_\star \rightarrow Sign)})$$

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

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

Transfer functions for procedure declarations

Procedure declarations

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

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

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

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

For the *procedure call*

$$f^1_{\ell_c} : \mathcal{P}(\Delta \times D) \to \mathcal{P}(\Delta \times D)$$

and it is used in the equation:

$$A_\bullet(\ell_c) \;=\; f^1_{\ell_c}(A_\circ(\ell_c)) \quad \text{for all procedure calls } [\texttt{call}\ p(a,z)]_{\ell_r}^{\ell_c}$$

For the *procedure return*

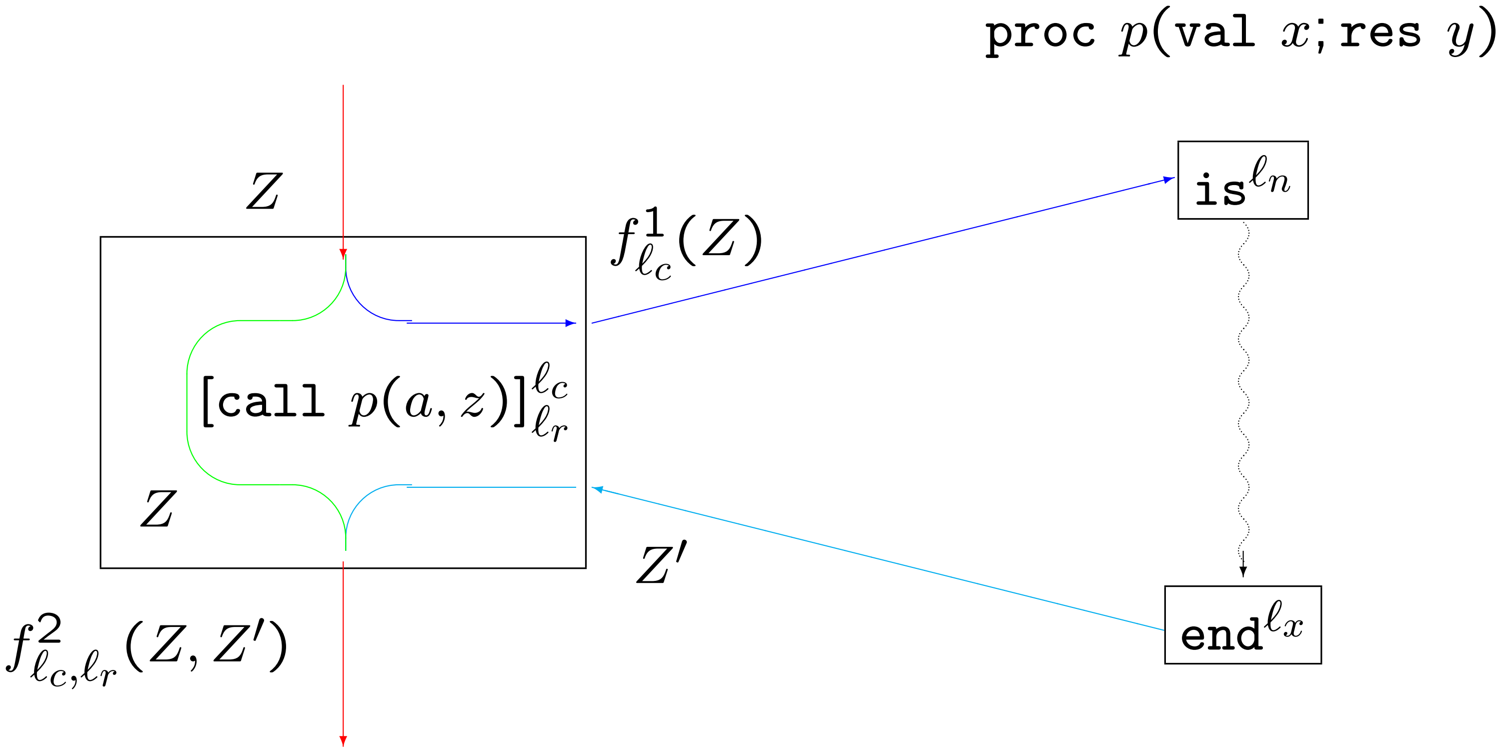
$$f^2_{\ell_c,\ell_r} : \boxed{\mathcal{P}(\Delta \times D)} \times \mathcal{P}(\Delta \times D) \to \mathcal{P}(\Delta \times D)$$

and it is used in the equation:

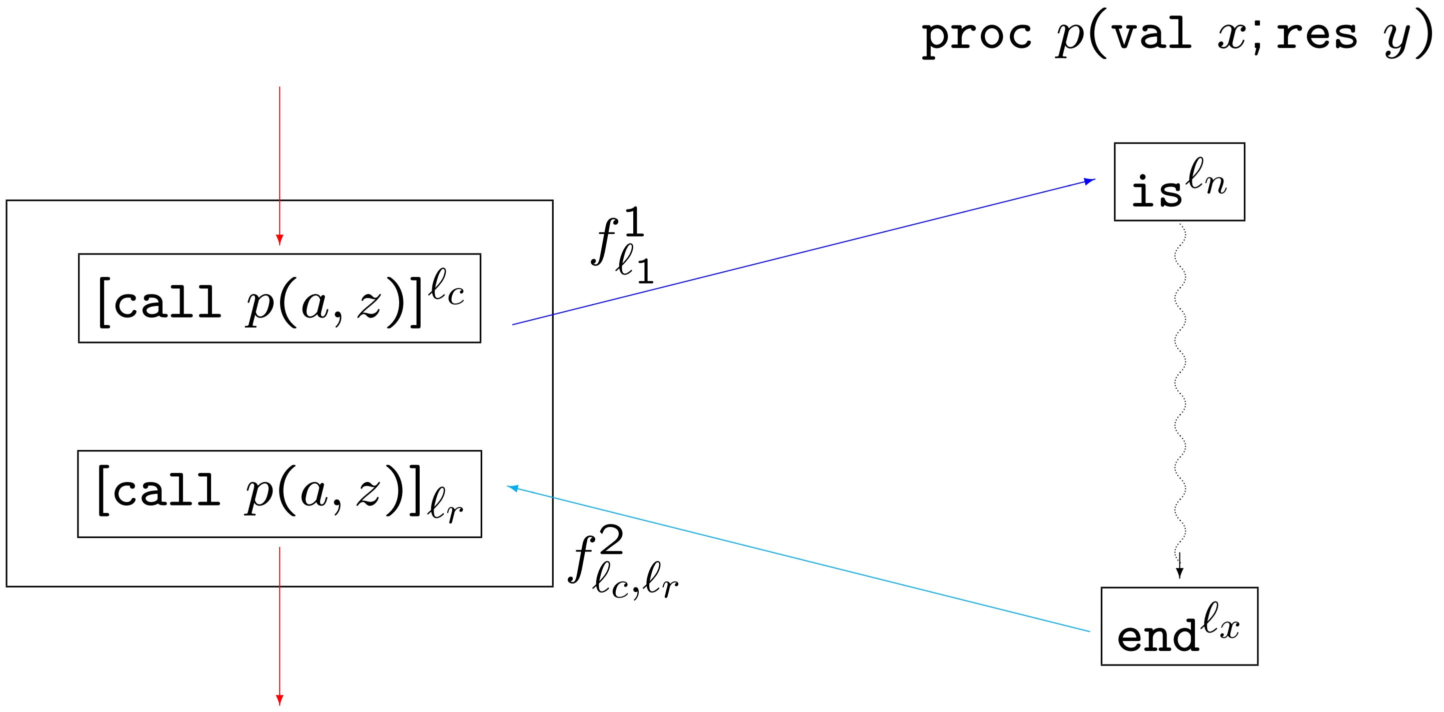
$$A_\bullet(\ell_r) \;=\; f^2_{\ell_c,\ell_r}(\boxed{A_\circ(\ell_c)}, A_\circ(\ell_r)) \quad \text{for all procedure calls } [\texttt{call}\ p(a,z)]_{\ell_r}^{\ell_c}$$

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

Procedure calls and returns



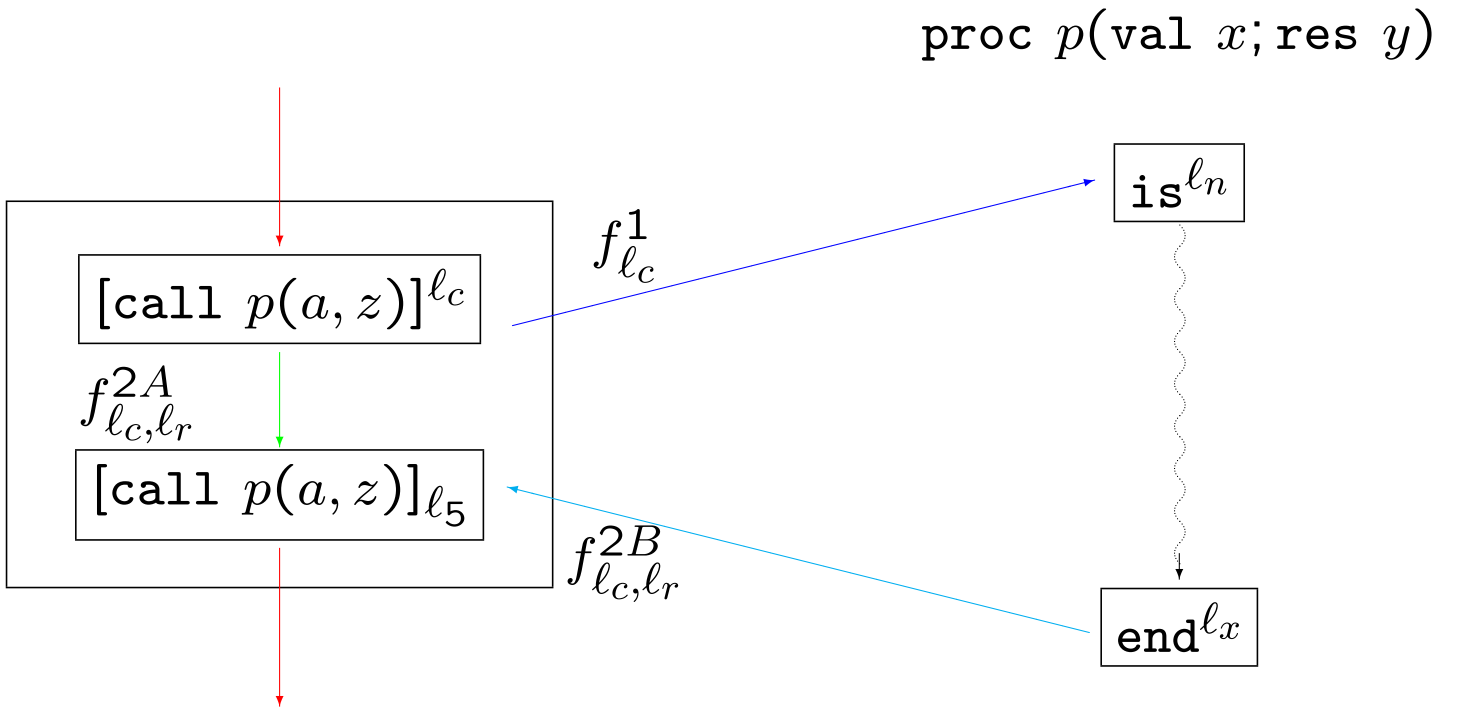
Variation 1: ignore calling context upon return



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

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

Variation 2: joining contexts upon return



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

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

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

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

Transfer functions for procedure call

$$f^1_{\ell_c}(Z) = \bigcup\{\{\delta'\} \times \phi^1_{\ell_c}(d) \mid \begin{array}{l}(\delta, d) \in Z \;\wedge \\ \delta' = [\delta, \ell_c]\}\end{array}$$

$$f^2_{\ell_c,\ell_r}(Z, Z') = \bigcup\{\{\delta\} \times \phi^2_{\ell_c,\ell_r}(d, d') \mid \begin{array}{l}(\delta, d) \in Z \;\wedge \\ (\delta', d') \in Z' \;\wedge\; \delta' = [\delta, \ell_c]\}\end{array}$$

Example:

Recalling the statements:

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

Detection of Signs Analysis:

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

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

Call Strings of Bounded Length

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

Transfer functions for procedure call

$$f^1_{\ell_c}(Z) = \bigcup\{\{\delta'\} \times \phi^1_{\ell_c}(d) \mid \begin{array}{l}(\delta, d) \in Z \ \wedge \\ \delta' = \lceil \delta, \ell_c \rceil_k\end{array}\}$$

$$f^2_{\ell_c,\ell_r}(Z, Z') = \bigcup\{\{\delta\} \times \phi^2_{\ell_c,\ell_r}(d, d') \mid \begin{array}{l}(\delta, d) \in Z \ \wedge \\ (\delta', d') \in Z' \ \wedge \ \delta' = \lceil \delta, \ell_c \rceil_k\end{array}\}$$

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

$$\Delta = \{\Lambda\}$$

Note: this is equivalent to having no context information!

Specialising the transfer functions:

$$f^1_{\ell_c}(Y) = \bigcup\{\phi^1_{\ell_c}(d) \mid d \in Y\}$$

$$f^2_{\ell_c,\ell_r}(Y, Y') = \bigcup\{\phi^2_{\ell_c,\ell_r}(d, d') \mid d \in Y \ \wedge \ d' \in Y'\}$$

(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:

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

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

Large Assumption Sets $(k = 1)$

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

Transfer functions for procedure call

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

$$f^2_{\ell_c,\ell_r}(Z, Z') = \bigcup\{\{\delta\} \times \phi^2_{\ell_c,\ell_r}(d, d') \mid \; (\delta, d) \in Z \;\wedge \\ (\delta', d') \in Z' \wedge \delta' = \{\, d'' \mid (\delta, d'') \in Z\}\}$$

Small Assumption Sets $(k = 1)$

$$\Delta = D$$

Transfer function for procedure call

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

$$f^2_{\ell_c,\ell_r}(Z, Z') = \bigcup\{\{\delta\} \times \phi^2_{\ell_c,\ell_r}(d, d') \mid \begin{array}{l}(\delta, d) \in Z \;\wedge \\ (d, d') \in Z'\}\end{array}$$

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

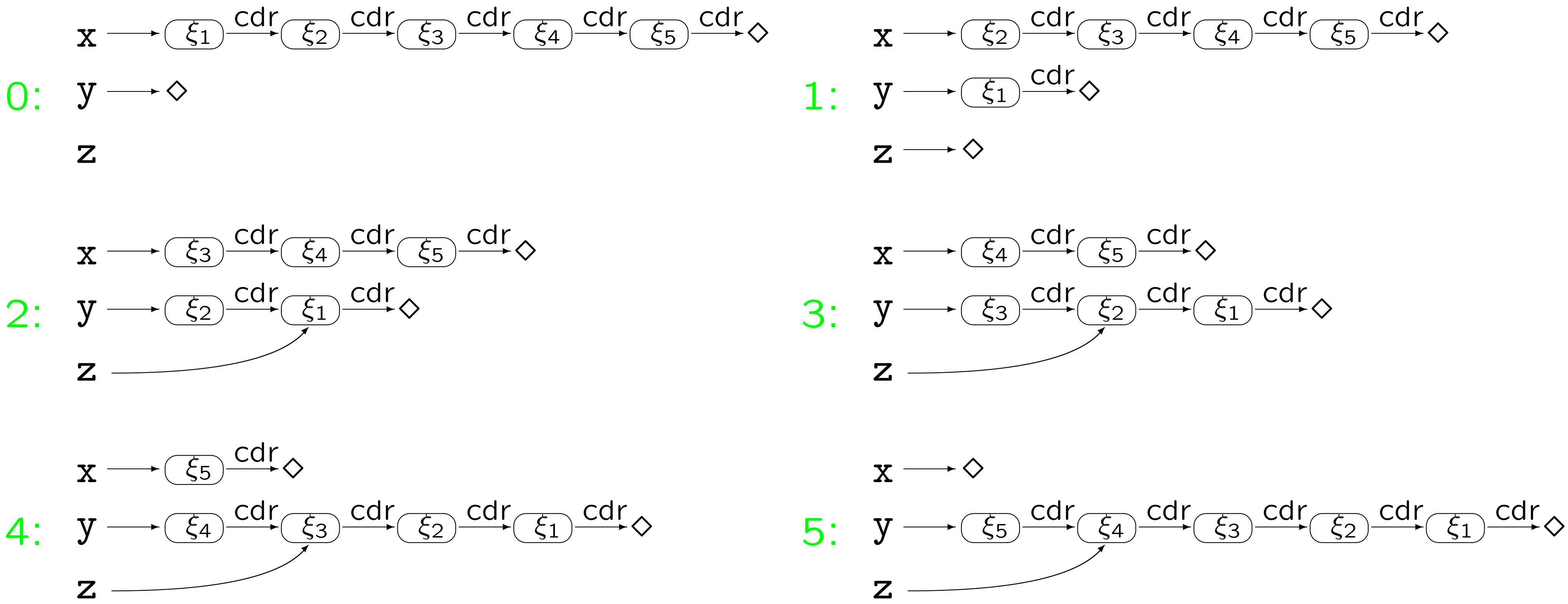
Syntax of the pointer language

$$\begin{array}{rcl} a & ::= & p \mid n \mid a_1\ op_a\ a_2 \mid \texttt{nil} \\ p & ::= & x \mid x.sel \\ b & ::= & \texttt{true} \mid \texttt{false} \mid \texttt{not}\ b \mid b_1\ op_b\ b_2 \mid a_1\ op_r\ a_2 \mid op_p\ p \\ S & ::= & [p\texttt{:=}a]^\ell \mid [\texttt{skip}]^\ell \mid S_1;\ S_2\ \mid \\ & & \texttt{if}\ [b]^\ell\ \texttt{then}\ S_1\ \texttt{else}\ S_2 \mid \texttt{while}\ [b]^\ell\ \texttt{do}\ S \mid \\ & & [\texttt{malloc}\ p]^\ell \end{array}$$

Example

$$\begin{array}{l} [\texttt{y:=nil}]^1; \\ \texttt{while}\ [\texttt{not is-nil(x)}]^2\ \texttt{do} \\ \quad ([\texttt{z:=y}]^3; [\texttt{y:=x}]^4; [\texttt{x:=x.cdr}]^5; [\texttt{y.cdr:=z}]^6); \\ [\texttt{z:=nil}]^7 \end{array}$$

Reversal of a list



Structural Operational Semantics

A configurations consists of

- a state $\sigma \in \mathbf{State} = \mathbf{Var}_\star \to (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$

  mapping variables to values, locations (in the heap) or the nil-value

- a heap $\mathcal{H} \in \mathbf{Heap} = (\mathbf{Loc} \times \mathbf{Sel}) \to_{\mathsf{fin}} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$

  mapping pairs of locations and selectors to values, locations in the heap or the nil-value

Pointer expressions

$$\wp \ : \ \mathbf{PExp} \to (\mathbf{State} \times \mathbf{Heap}) \to_{\mathsf{fin}} (\mathbf{Z} + \{\diamond\} + \mathbf{Loc})$$

is defined by

$$\wp[\![x]\!](\sigma, \mathcal{H}) \ = \ \sigma(x)$$

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

Arithmetic and boolean expressions

$$\mathcal{A} \ : \ \mathbf{AExp} \to (\mathbf{State} \times \mathbf{Heap}) \to_{\mathsf{fin}} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$$

$$\mathcal{B} \ : \ \mathbf{BExp} \to (\mathbf{State} \times \mathbf{Heap}) \to_{\mathsf{fin}} \mathbf{T}$$

Statements

Clauses for assignments:

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

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

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

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

Clauses for malloc:

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

where ξ does not occur in σ or $\mathcal{H}$

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

where ξ does not occur in σ 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.

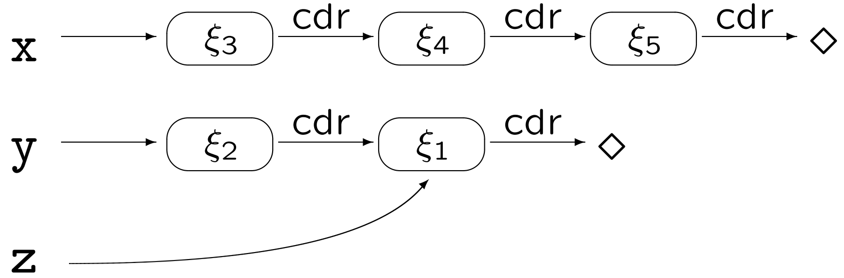
The nodes of the shape graphs are abstract locations:

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

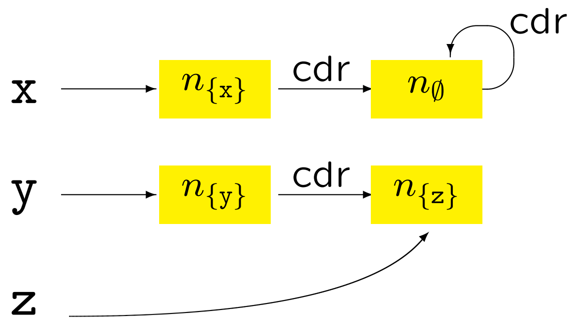
Note: there will only be *finitely* many abstract locations

Example

In the semantics:



In the analysis:



Abstract Locations

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

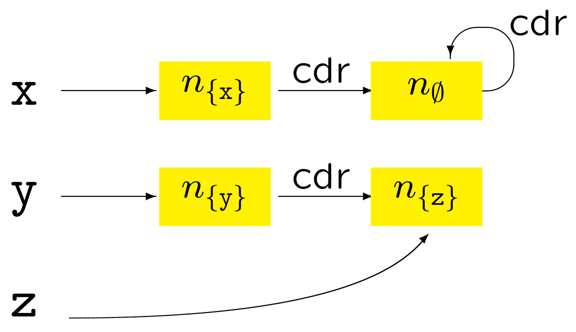
The abstract location $n_\emptyset$ is called the *abstract summary location*: $n_\emptyset$ represents all the locations that cannot be reached directly from the state without consulting the heap

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

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

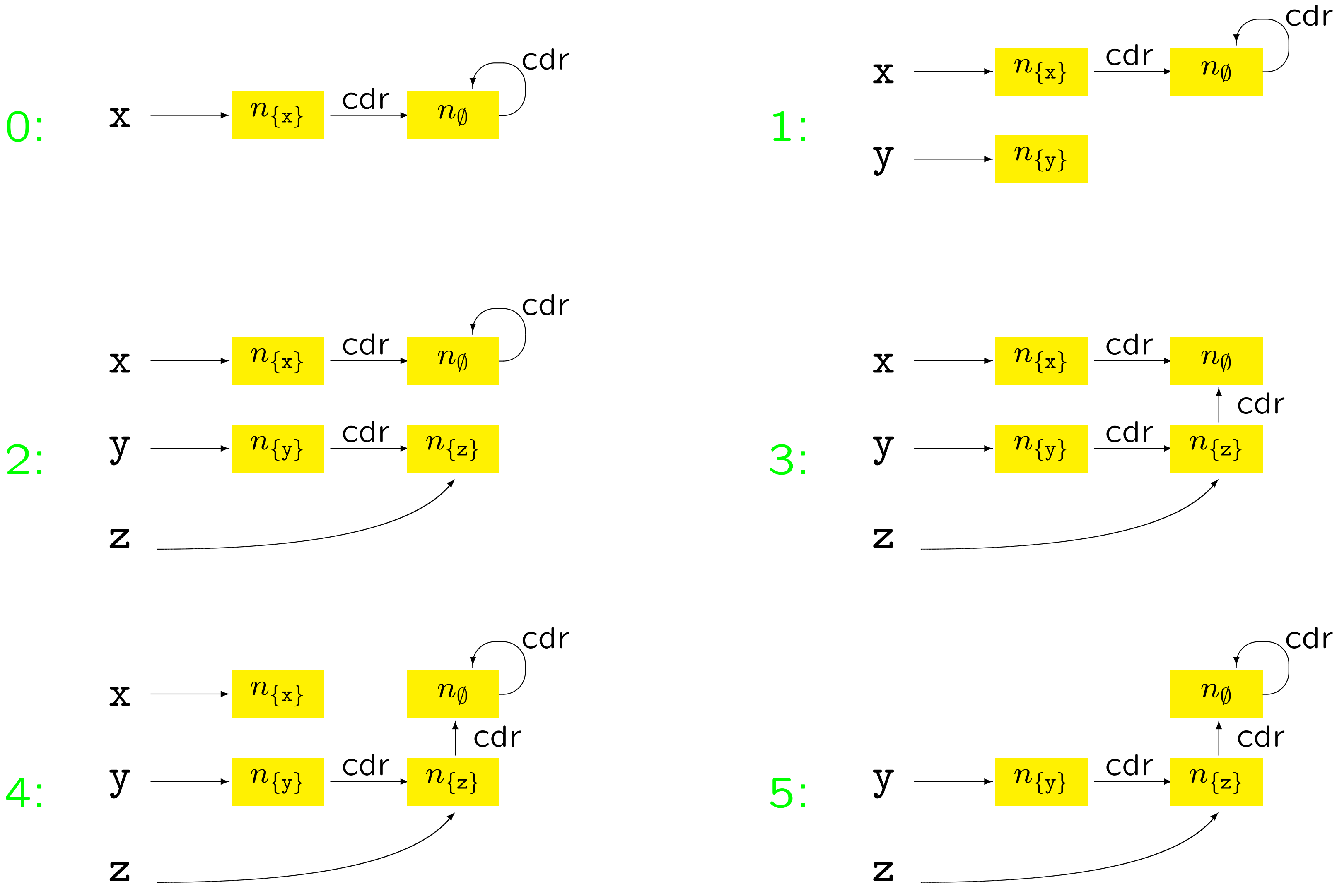
$$\mathsf{H} \in \mathbf{AHeap} = \mathcal{P}(\mathbf{ALoc} \times \mathbf{Sel} \times \mathbf{ALoc}) \quad \text{abstract heap}$$



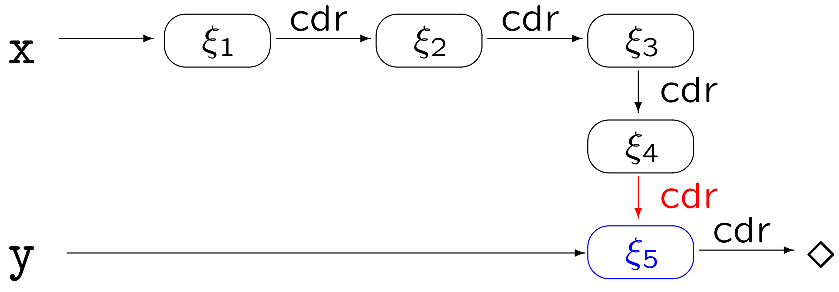
Invariant 2 If x is mapped to n_X by the abstract state S then $x \in X$

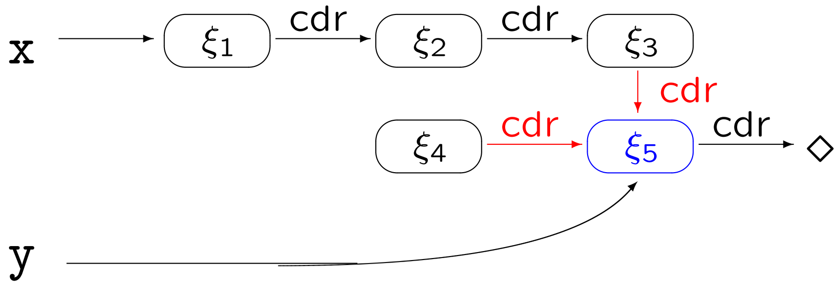
Invariant 3 Whenever (n_V, sel, n_W) and $(n_V, \mathit{sel}, n_{W'})$ are in the abstract heap H then either $V = \emptyset$ or $W = W'$

Reversal of a list

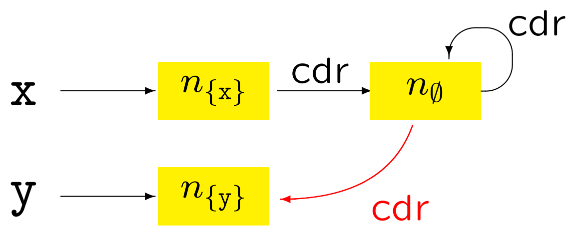


Sharing in the heap





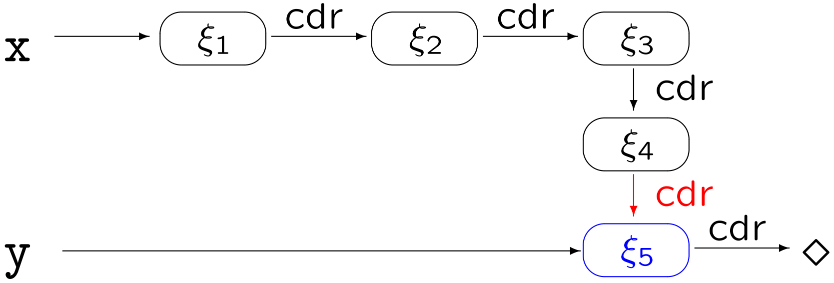
Give rise to the same shape graph:

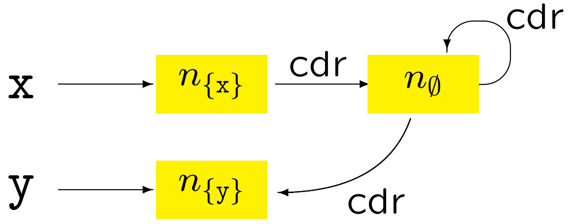


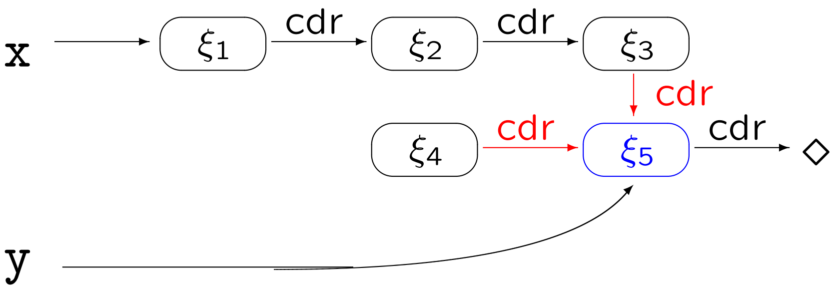
is: the abstract locations that *might* be shared due to pointers in the heap:

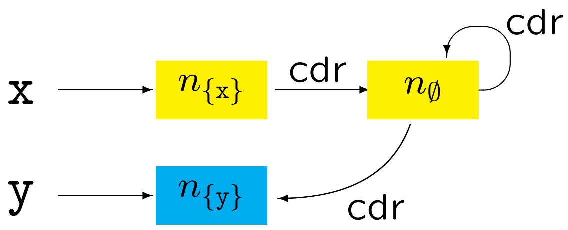
n_X is included in **is** if it might represents a location that is the target of more than one pointer in the heap

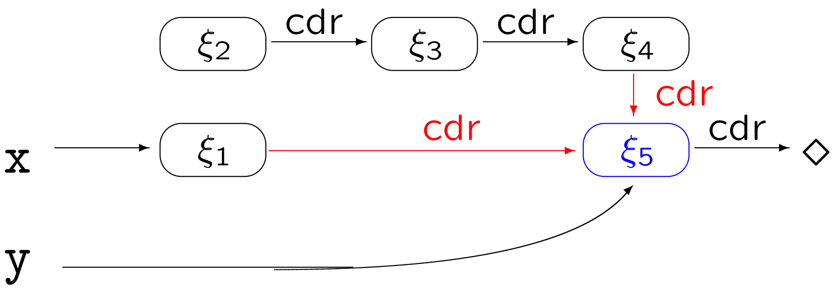
Examples: sharing in the heap

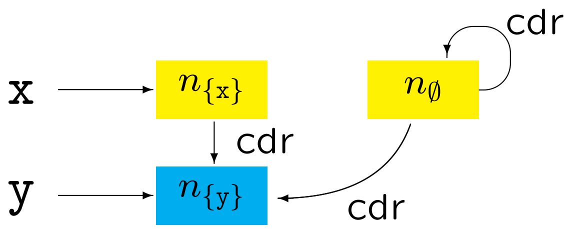






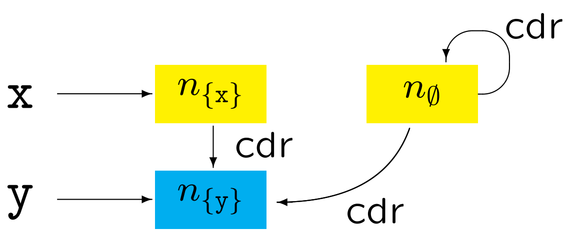






Sharing information

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



Invariant 4 If $n_X \in \mathsf{is}$ then either

- $(n_\emptyset, \mathit{sel}, n_X)$ is in the abstract heap for some *sel*, or
- there are two distinct triples $(n_V, \mathit{sel}_1, n_X)$ and $(n_W, \mathit{sel}_2, n_X)$ in the abstract heap

Invariant 5 Whenever there are two distinct triples $(n_V, \mathit{sel}_1, n_X)$ and $(n_W, \mathit{sel}_2, n_X)$ in the abstract heap and $X \neq \emptyset$ then $n_X \in \mathsf{is}$

The complete lattice of shape graphs

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

$$\begin{array}{rcl} \mathsf{S} \in \mathbf{AState} & = & \mathcal{P}(\mathbf{Var}_\star \times \mathbf{ALoc}) \\ \mathsf{H} \in \mathbf{AHeap} & = & \mathcal{P}(\mathbf{ALoc} \times \mathbf{Sel} \times \mathbf{ALoc}) \\ \mathsf{is} \in \mathbf{IsShared} & = & \mathcal{P}(\mathbf{ALoc}) \end{array}$$

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

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

The analysis computes over *sets of compatible shape graphs*

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

The analysis

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

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

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 $(\sigma, \mathcal{H})$

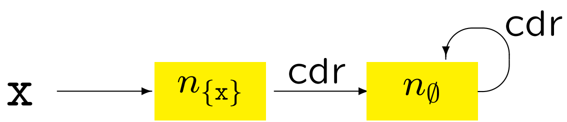
The analysis

Equations:

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

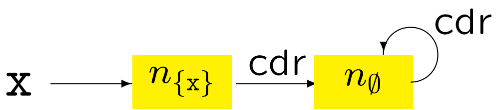
$$\mathit{Shape}_\bullet(\ell) = f_\ell^{\mathsf{SA}}(\mathit{Shape}_\circ(\ell))$$

Example: The extremal value ι for the list reversal program



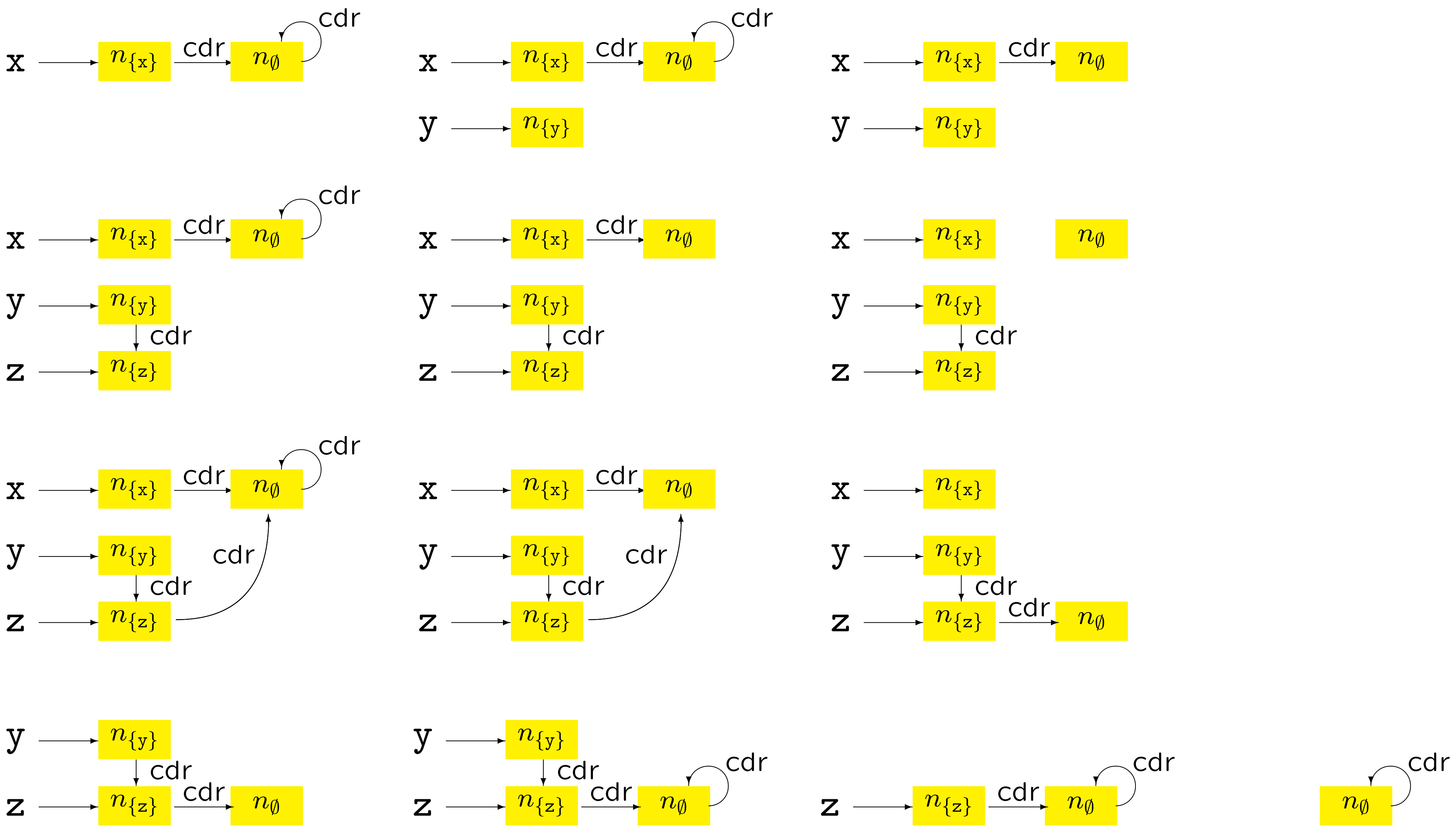
– x points to a non-cyclic list with at least three elements

Shape$_\bullet$(1) for $[\texttt{y:=nil}]^1$

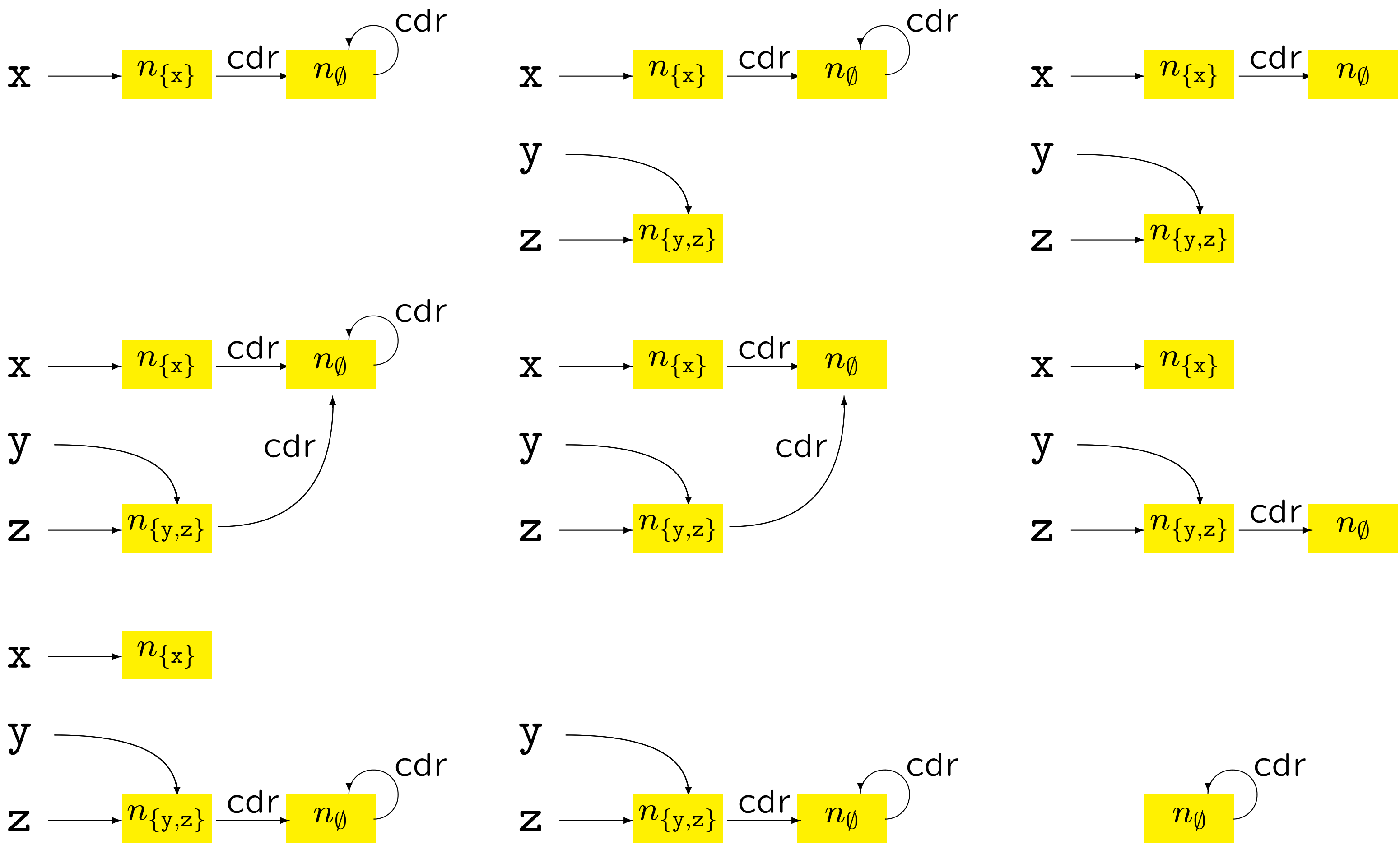


Note: we do not record `nil`-values in the analysis

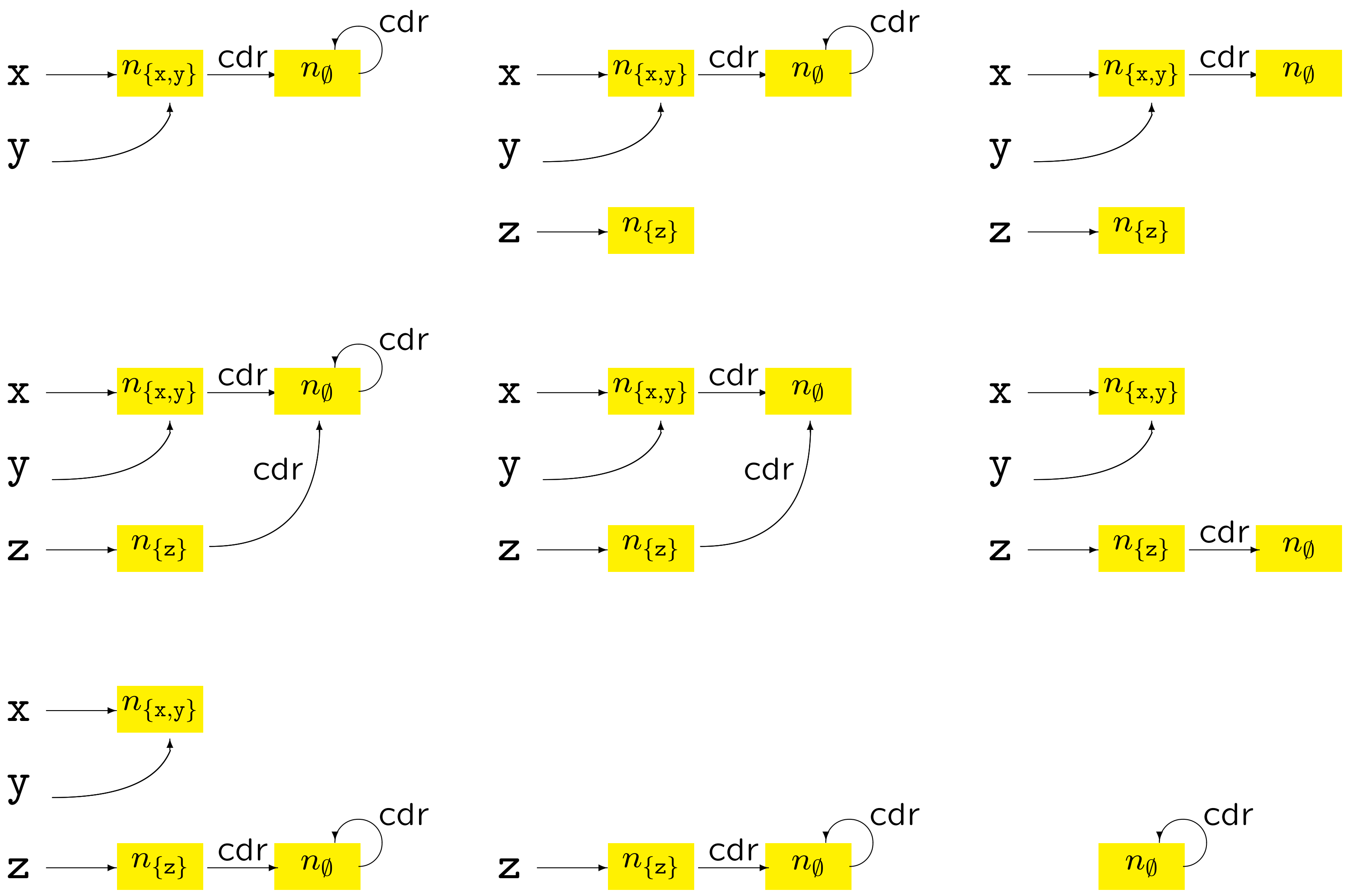
$\mathit{Shape}_\bullet(2)$ for [not is-nil(x)]^2



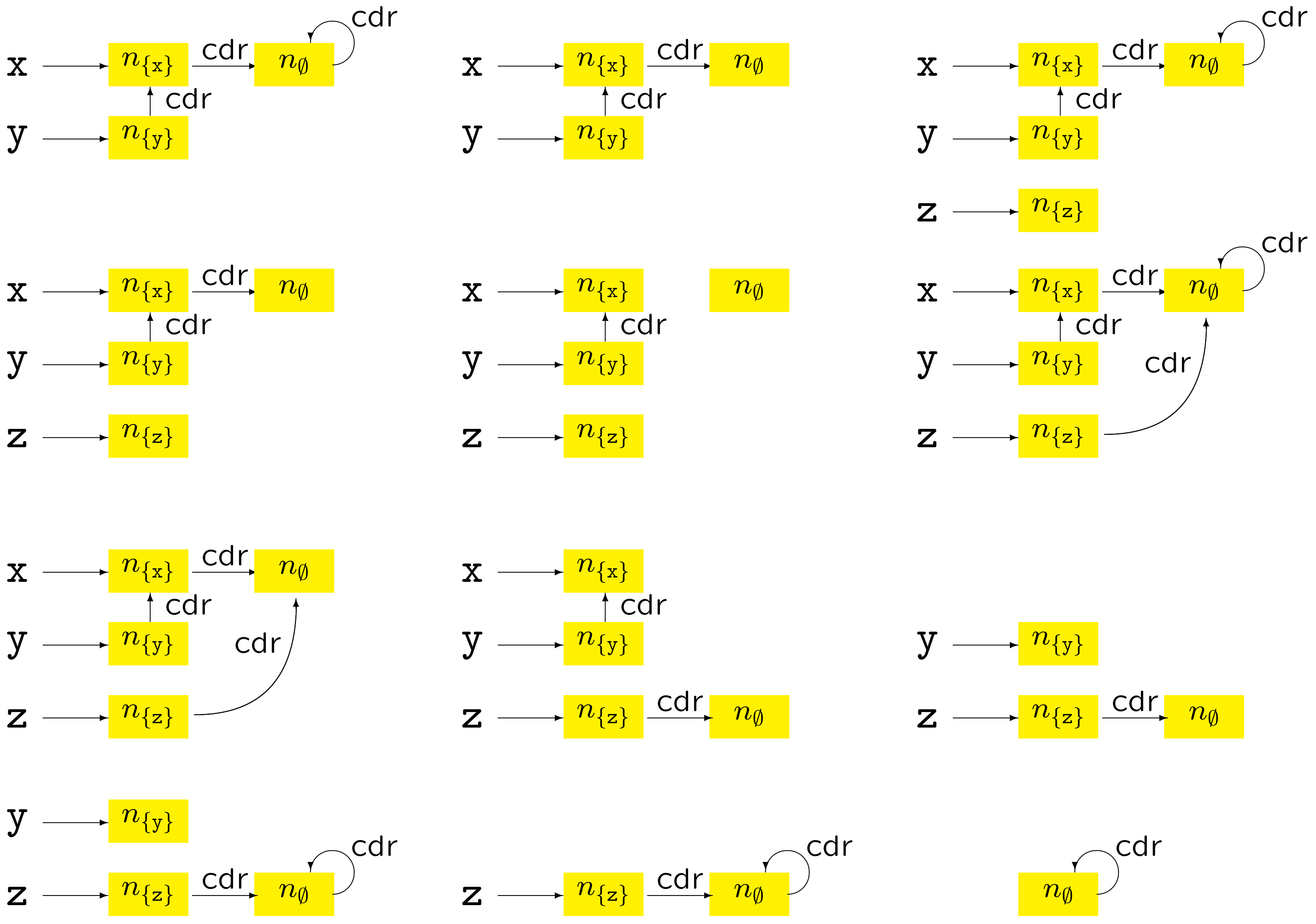
Shape$_\bullet$(3) for [z:=y]^3



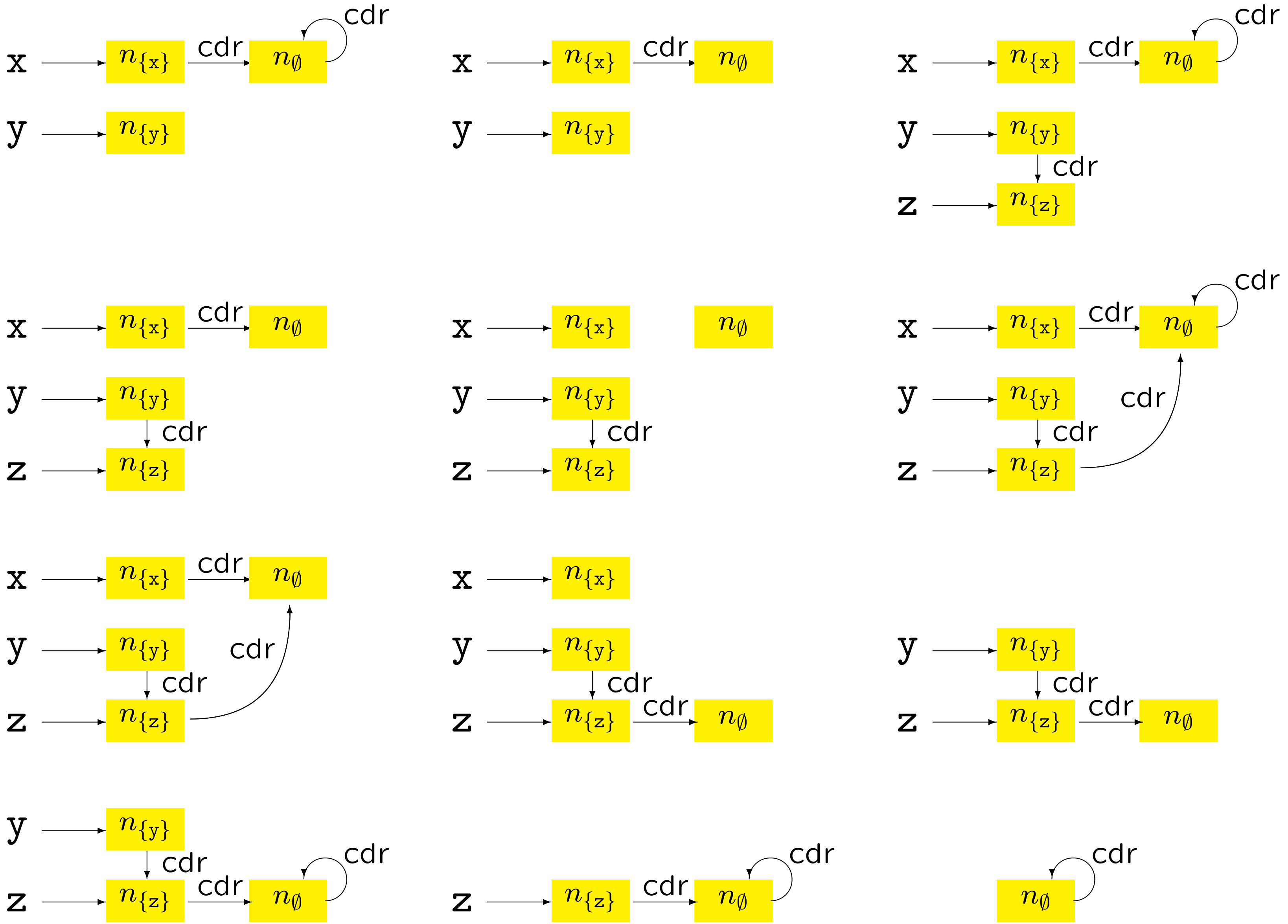
Shape$_\bullet$(4) for [y:=x]^4



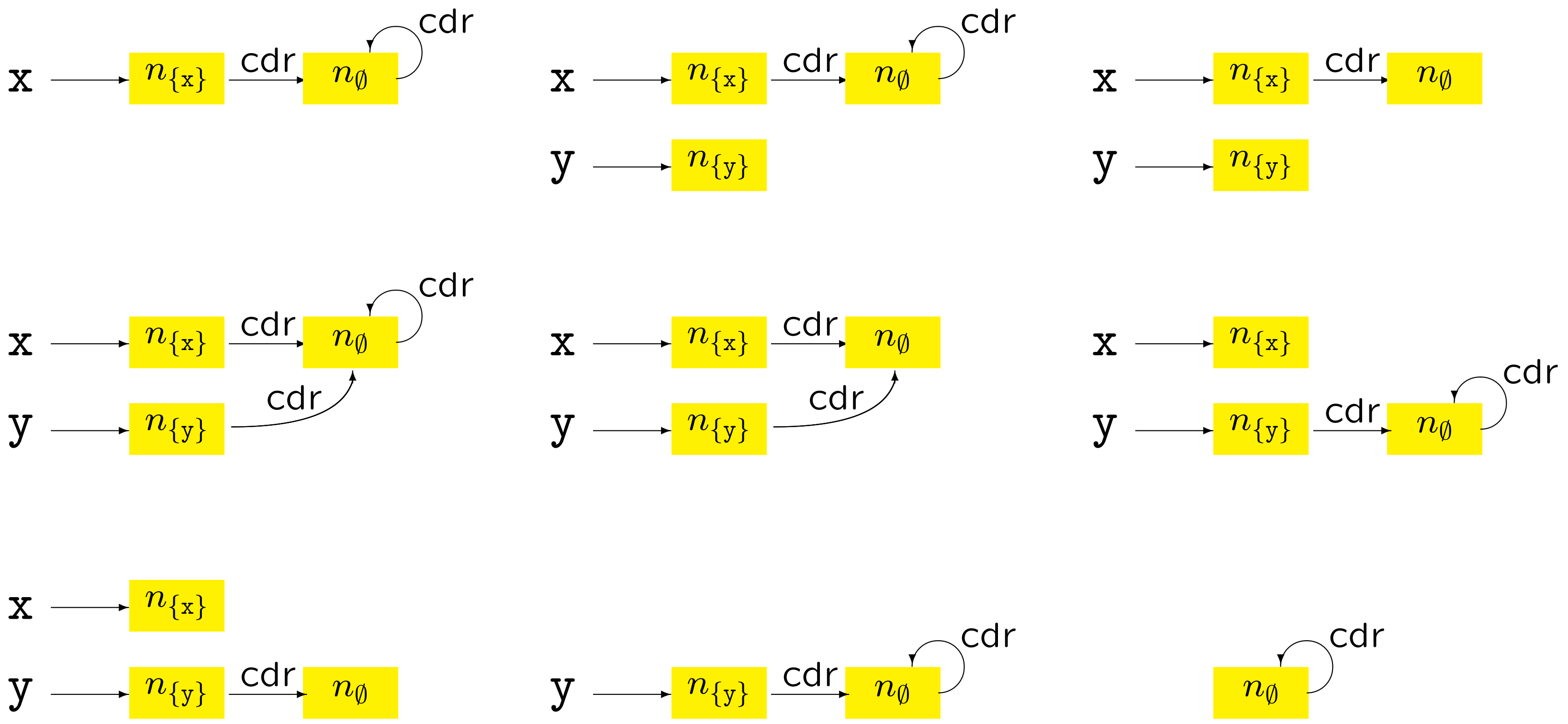
Shape$_\bullet$(5) for [x:=x.cdr]^5



Shape$_\bullet$(6) for $[\texttt{y.cdr:=z}]^6$



Shape$_\bullet$(7) for [z:=nil]^7



– upon termination y points to a non-circular list

– a more precise analysis taking tests into account will know that x is nil upon termination

Transfer functions

$$f_\ell^{\mathsf{SA}} : \mathcal{P}(\mathbf{SG}) \to \mathcal{P}(\mathbf{SG})$$

has the form:

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

where

$$\phi_\ell^{\mathsf{SA}} : \mathbf{SG} \to \mathcal{P}(\mathbf{SG})$$

specifies how a *single* shape graph (in $\mathit{Shape}_\circ(\ell)$) may be transformed into a *set* of shape graphs (in $\mathit{Shape}_\bullet(\ell)$) by the elementary block.

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

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

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

Transfer function for $[x\texttt{:=}a]^\ell$

— where a is of the form n, a_1 op_a a_2 or `nil`

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

where $kill_x((\mathsf{S},\mathsf{H},\mathsf{is})) = (\mathsf{S}',\mathsf{H}',\mathsf{is}')$ is

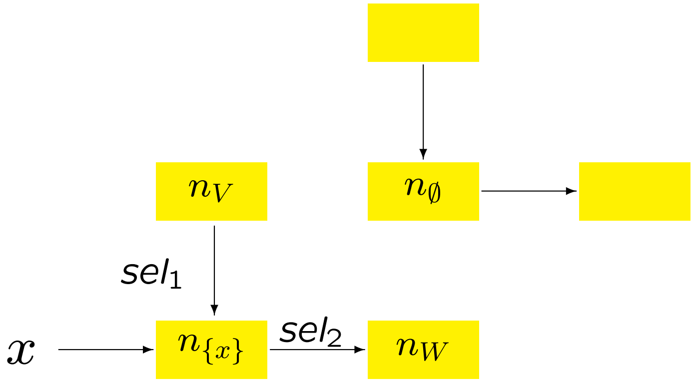
$$\begin{array}{rcl} \mathsf{S}' & = & \{(z, k_x(n_Z)) \mid (z, n_Z) \in S \ \wedge \ z \neq x\} \\ \mathsf{H}' & = & \{(k_x(n_V), \mathit{sel}, k_x(n_W)) \mid (n_V, \mathit{sel}, n_W) \in \mathsf{H}\} \\ \mathsf{is}' & = & \{k_x(n_X) \mid n_X \in \mathsf{is}\} \end{array}$$

and

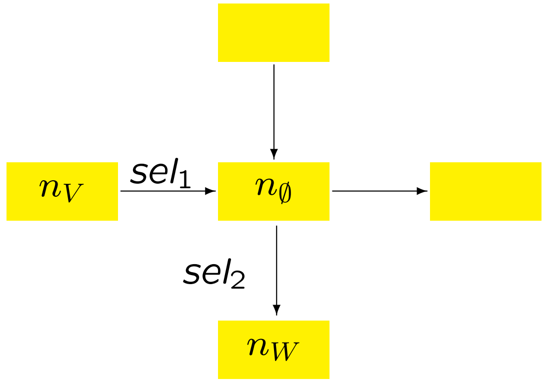
$$k_x(n_Z) = n_{Z\setminus\{x\}}$$

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

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



$(\mathsf{S}, \mathsf{H}, \mathsf{is})$



$(\mathsf{S}', \mathsf{H}', \mathsf{is}')$

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

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

where $(\mathsf{S}',\mathsf{H}',\mathsf{is}') = kill_x((\mathsf{S},\mathsf{H},\mathsf{is}))$ and

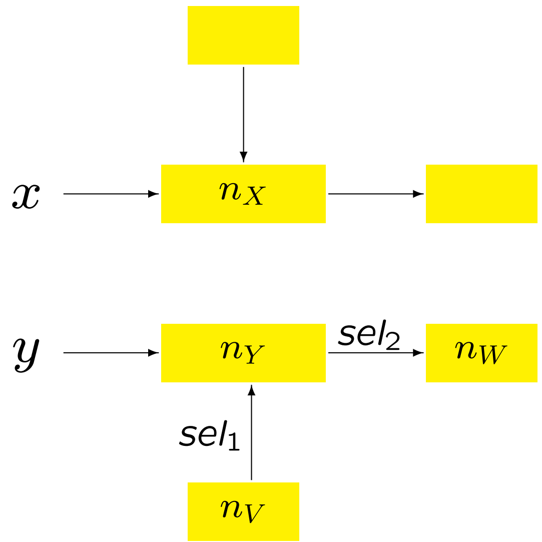
$$\begin{array}{rcl} \mathsf{S}'' & = & \{(z, g_x^y(n_Z)) \mid (z, n_Z) \in \mathsf{S}'\} \\ & & \cup\ \{(x, g_x^y(n_Y)) \mid (y', n_Y) \in \mathsf{S}' \wedge y' = y\} \\ \mathsf{H}'' & = & \{(g_x^y(n_V), sel, g_x^y(n_W)) \mid (n_V, sel, n_W) \in \mathsf{H}'\} \\ \mathsf{is}'' & = & \{g_x^y(n_Z) \mid n_Z \in \mathsf{is}'\} \end{array}$$

and

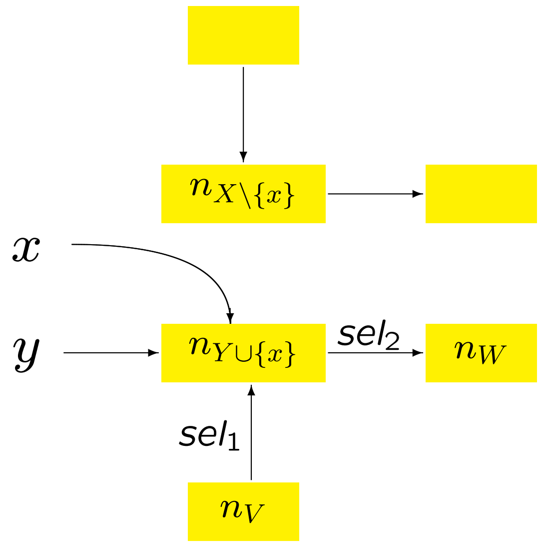
$$g_x^y(n_Z) = \begin{cases} n_{Z\cup\{x\}} & \text{if } y \in Z \\ n_Z & \text{otherwise} \end{cases}$$

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 $x \neq y$



$(\mathsf{S}, \mathsf{H}, \mathsf{is})$



$(\mathsf{S}'', \mathsf{H}'', \mathsf{is}'')$

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

Remove the old binding for x: strong nullification

$$(\mathsf{S}', \mathsf{H}', \mathsf{is}') = kill_x((\mathsf{S}, \mathsf{H}, \mathsf{is}))$$

Establish the new binding for x:

1. There is no abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ – or there is an abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ but no n_Z such that $(n_Y, sel, n_Z) \in \mathsf{H}'$

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

3. There is an abstract location n_Y such that $(y, n_Y) \in \mathsf{S}'$ and $(n_Y, sel, n_\emptyset) \in \mathsf{H}'$

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

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

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

OBS: dereference of a `nil`-pointer

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

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

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

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

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

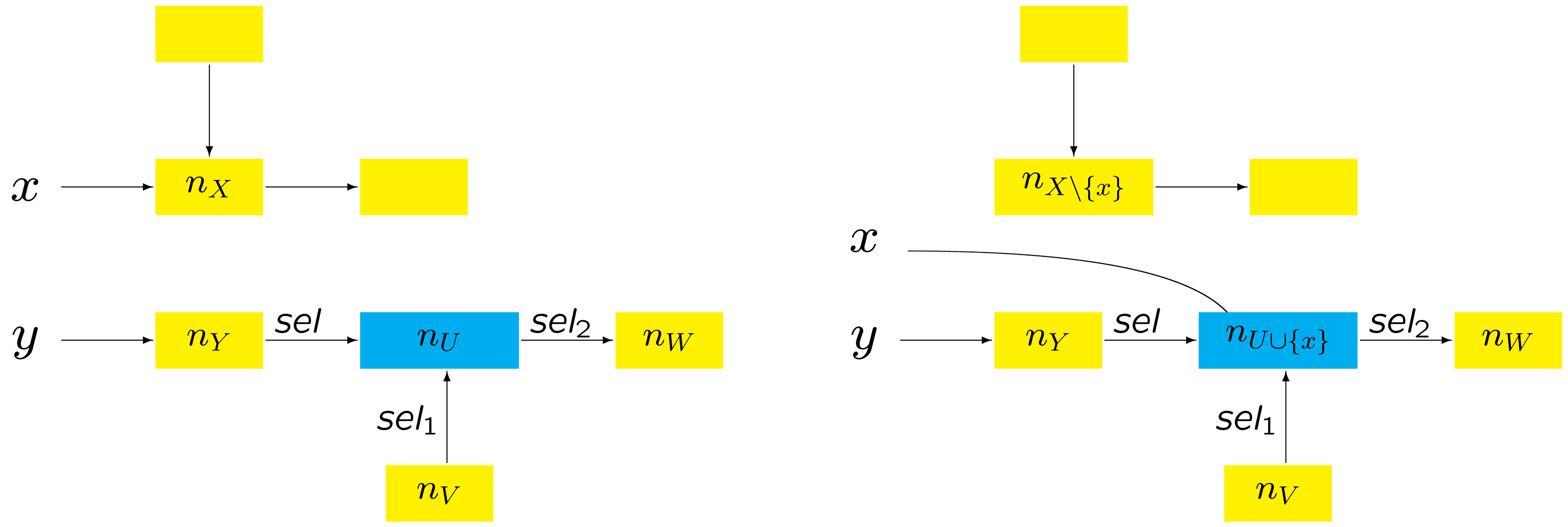
We take

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

where $(\mathsf{S}', \mathsf{H}', \mathsf{is}') = kill_x((\mathsf{S}, \mathsf{H}, \mathsf{is}))$ and

$$\begin{array}{rcl} \mathsf{S}'' & = & \{(z, h_x^U(n_Z)) \mid (z, n_Z) \in \mathsf{S}'\} \cup \{(x, h_x^U(n_U))\} \\ \mathsf{H}'' & = & \{(h_x^U(n_V), sel', h_x^U(n_W)) \mid (n_V, sel', n_W) \in \mathsf{H}'\} \\ \mathsf{is}'' & = & \{h_x^U(n_Z) \mid n_Z \in \mathsf{is}'\} \end{array}$$

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



$(\mathsf{S}, \mathsf{H}, \mathsf{is})$ $(\mathsf{S}'', \mathsf{H}'', \mathsf{is}'')$

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

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

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

$$[x\texttt{:=nil}]^{\cdots};\ [x\texttt{:=}y.sel]^{\ell};\ [x\texttt{:=nil}]^{\cdots}$$

$$(\mathsf{S}, \mathsf{H}, \mathsf{is}) \quad (\mathsf{S}', \mathsf{H}', \mathsf{is}') \quad (\mathsf{S}'', \mathsf{H}'', \mathsf{is}'') \quad (\mathsf{S}''', \mathsf{H}''', \mathsf{is}''')$$

Idea:

$$(\mathsf{S}', \mathsf{H}', \mathsf{is}') = (\mathsf{S}''', \mathsf{H}''', \mathsf{is}''') = kill_x((\mathsf{S}'', \mathsf{H}'', \mathsf{is}''))$$

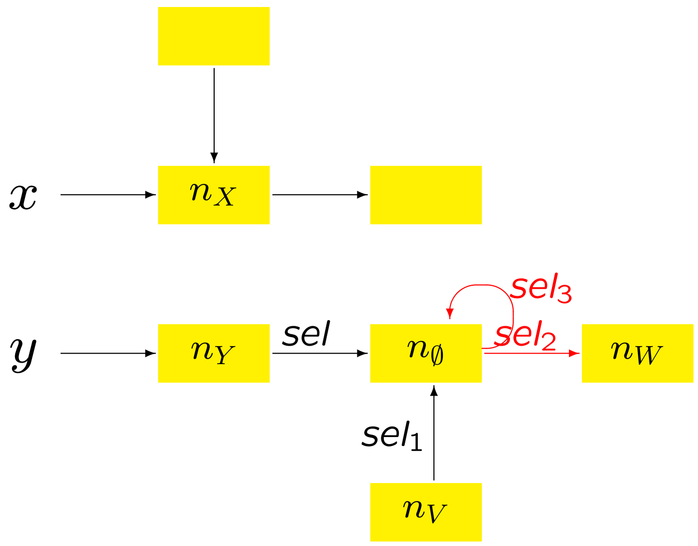
Case 3 for $[x := y.sel]^\ell$ (2)

Transfer function:

$$\phi_\ell^{\mathsf{SA}}((\mathsf{S},\mathsf{H},\mathsf{is})) = \{(\mathsf{S}'',\mathsf{H}'',\mathsf{is}'') \mid (\mathsf{S}'',\mathsf{H}'',\mathsf{is}'') \text{ is compatible} \land kill_x((\mathsf{S}'',\mathsf{H}'',\mathsf{is}'')) = (\mathsf{S}',\mathsf{H}',\mathsf{is}') \land (x, n_{\{x\}}) \in \mathsf{S}'' \land (n_Y, sel, n_{\{x\}}) \in \mathsf{H}''\}$$

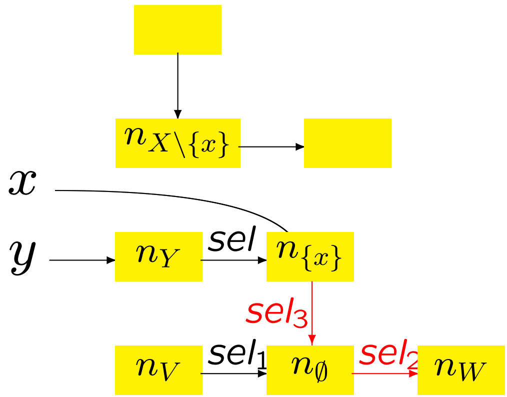
where $(\mathsf{S}',\mathsf{H}',\mathsf{is}') = kill_x((\mathsf{S},\mathsf{H},\mathsf{is}))$.

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

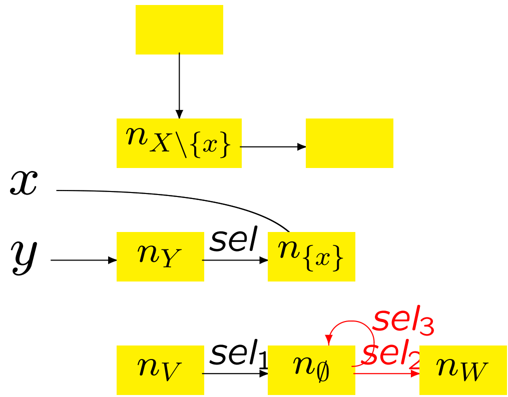


(S, H, is)

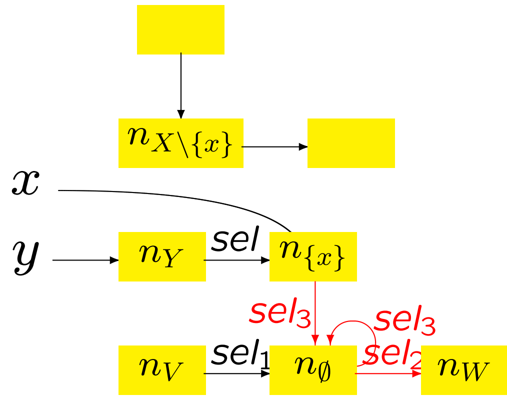
The effect of $[x\!:=\!y.sel]^\ell$ in Case 3 (2)



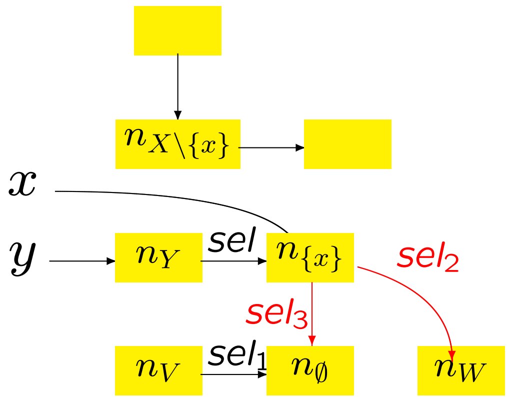
(S''_1, H''_1, is''_1)



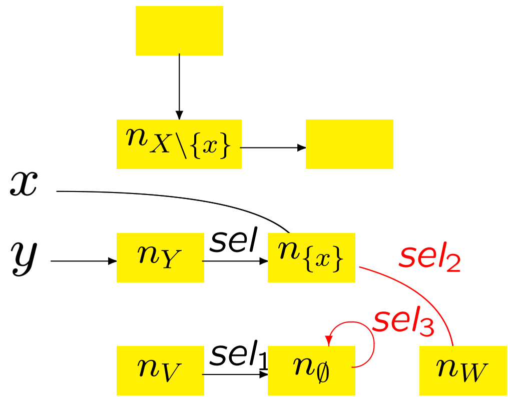
(S''_3, H''_3, is''_3)



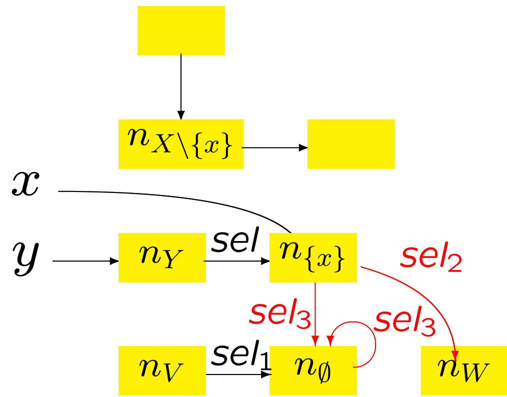
(S''_5, H''_5, is''_5)



(S''_2, H''_2, is''_2)



(S''_4, H''_4, is''_4)



(S''_6, H''_6, is''_6)

Transfer function for $[x.sel\texttt{:=}a]^\ell$

— where a is of the form n, a_1 op_a a_2 or `nil`.

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

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

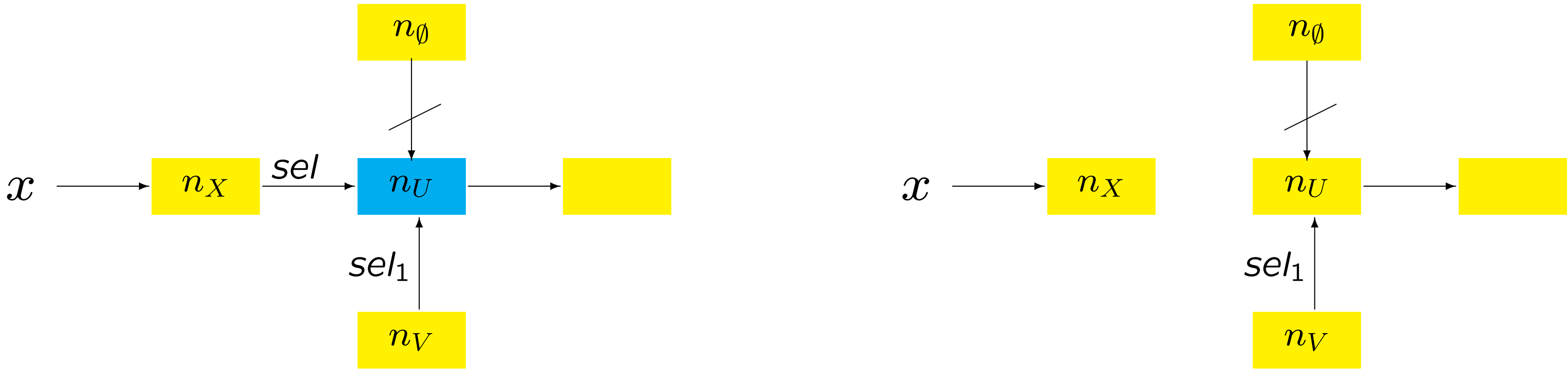
If there are abstract locations n_X and n_U such that $(x, n_X) \in \mathsf{S}$ and $(n_X, sel, n_U) \in \mathsf{H}$ then

$$\phi_\ell^{\mathsf{SA}}((\mathsf{S}, \mathsf{H}, \mathsf{is})) = \{kill_{x.sel}((\mathsf{S}, \mathsf{H}, \mathsf{is}))\}$$

where $kill_{x.sel}((\mathsf{S}, \mathsf{H}, \mathsf{is})) = (\mathsf{S}', \mathsf{H}', \mathsf{is}')$ is given by

$$\begin{aligned}
\mathsf{S}' &= \mathsf{S} \\
\mathsf{H}' &= \{(n_V, sel', n_W) \mid (n_V, sel', n_W) \in \mathsf{H} \;\wedge\; \neg(X = V \;\wedge\; sel = sel')\} \\
\mathsf{is}' &= \begin{cases} \mathsf{is}\backslash\{n_U\} & \text{if } \; n_U \in \mathsf{is} \;\wedge\; \#into(n_U, \mathsf{H}') \leq 1 \;\wedge\; \neg\exists(n_\emptyset, sel', n_U) \in \mathsf{H}' \\ \mathsf{is} & \text{otherwise} \end{cases}
\end{aligned}$$

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



$(\mathsf{S}, \mathsf{H}, \mathsf{is})$ $(\mathsf{S}', \mathsf{H}', \mathsf{is}')$

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

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

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

$$\phi_\ell^{\mathsf{SA}}((\mathsf{S}, \mathsf{H}, \mathsf{is})) = \{kill_{x.sel}((\mathsf{S}, \mathsf{H}, \mathsf{is}))\}$$

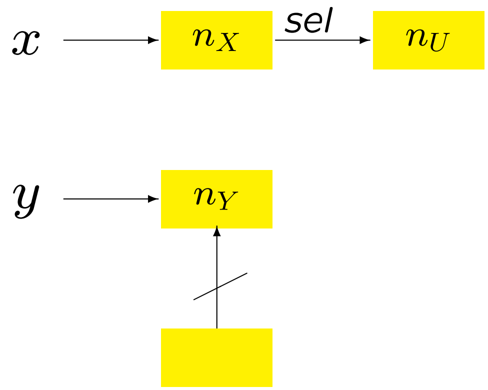
If there is $(x, n_X) \in \mathsf{S}$ and $(y, n_Y) \in \mathsf{S}$ then

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

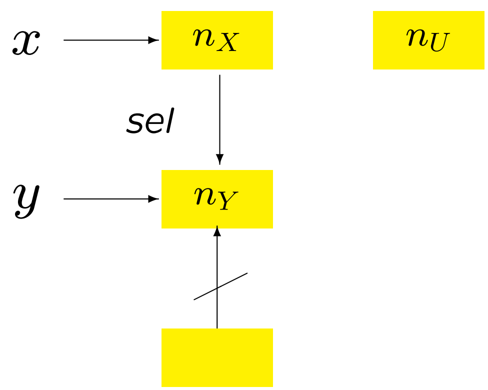
where $(\mathsf{S}', \mathsf{H}', \mathsf{is}') = kill_{x.sel}((\mathsf{S}, \mathsf{H}, \mathsf{is}))$ and

$$\begin{aligned}
\mathsf{S}'' &= \mathsf{S}' \quad (= \mathsf{S}) \\
\mathsf{H}'' &= \mathsf{H}' \cup \{(n_X, \mathit{sel}, n_Y) \mid (x, n_X) \in \mathsf{S}' \ \wedge \ (y, n_Y) \in \mathsf{S}'\} \\
\mathsf{is}'' &= \begin{cases} \mathsf{is}' \cup \{n_Y\} & \text{if } \#\mathit{into}(n_Y, \mathsf{H}') \geq 1 \\ \mathsf{is}' & \text{otherwise} \end{cases}
\end{aligned}$$

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



$(\mathsf{S}, \mathsf{H}, \mathsf{is})$



$(\mathsf{S}', \mathsf{H}'', \mathsf{is}'')$

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

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

where $(\mathsf{S}', \mathsf{H}', \mathsf{is}') = kill_x(\mathsf{S},\mathsf{H},\mathsf{is})$.