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

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

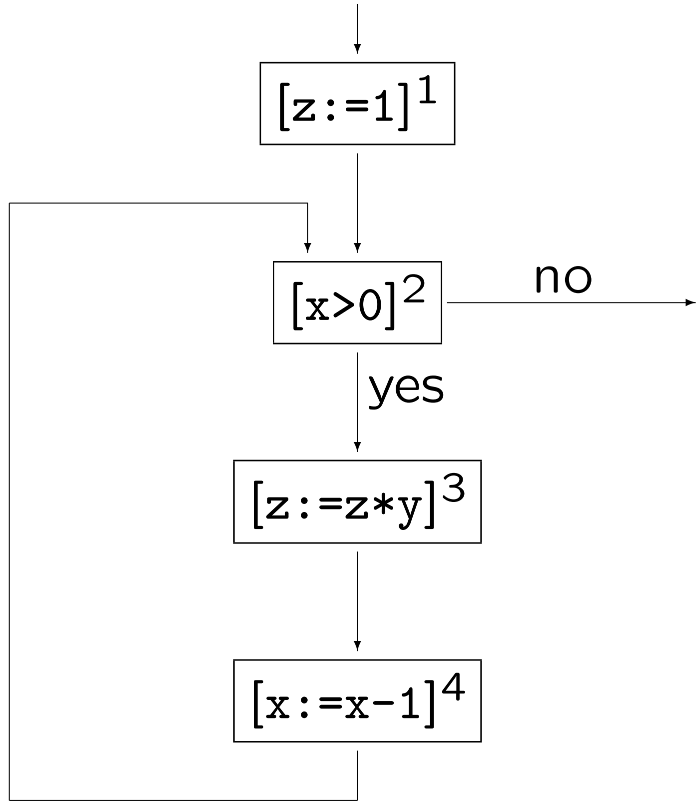
Example: $[\texttt{z:=1}]^1; \texttt{while}\ [\texttt{x>0}]^2\ \texttt{do}\ ([\texttt{z:=z*y}]^3; [\texttt{x:=x-1}]^4)$

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:

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

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

Example:

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

Final labels

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

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

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

Example:

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

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*

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

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

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

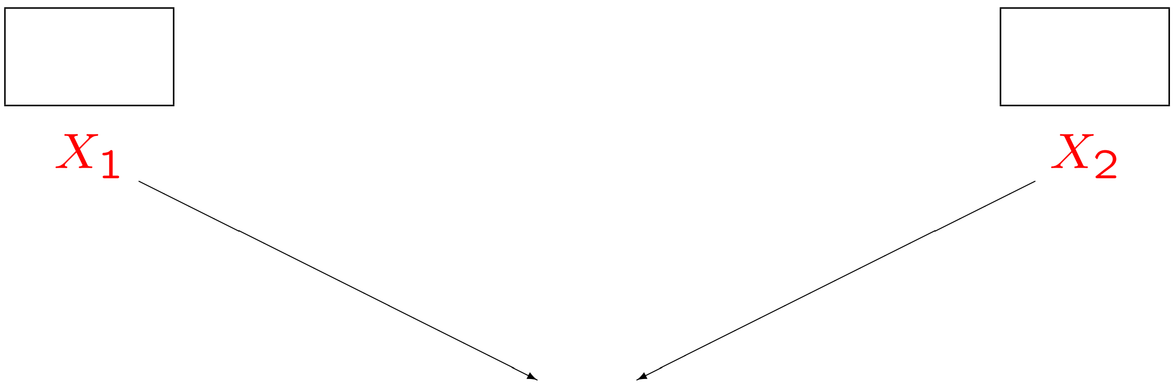
$\Downarrow$

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

The analysis enables a transformation into

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

Available Expressions Analysis – the basic idea



$$N = X_1 \cap X_2$$

$$\boxed{x := a}$$

$$X = (N \backslash \overbrace{\{\text{expressions with an } x\}}^{\text{kill}}\,)$$
$$\cup \underbrace{\{\text{subexpressions of } a \text{ without an } x\}}_{\text{gen}}$$

Available Expressions Analysis

kill and *gen* functions

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

data flow equations: $\boxed{\mathsf{AE}^=}$

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

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

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	$\mathit{kill}_{\mathsf{AE}}(\ell)$	$\mathit{gen}_{\mathsf{AE}}(\ell)$
1	$\emptyset$	$\{\texttt{a+b}\}$
2	$\emptyset$	$\{\texttt{a*b}\}$
3	$\emptyset$	$\{\texttt{a+b}\}$
4	$\{\texttt{a+b}, \texttt{a*b}, \texttt{a+1}\}$	$\emptyset$
5	$\emptyset$	$\{\texttt{a+b}\}$

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:

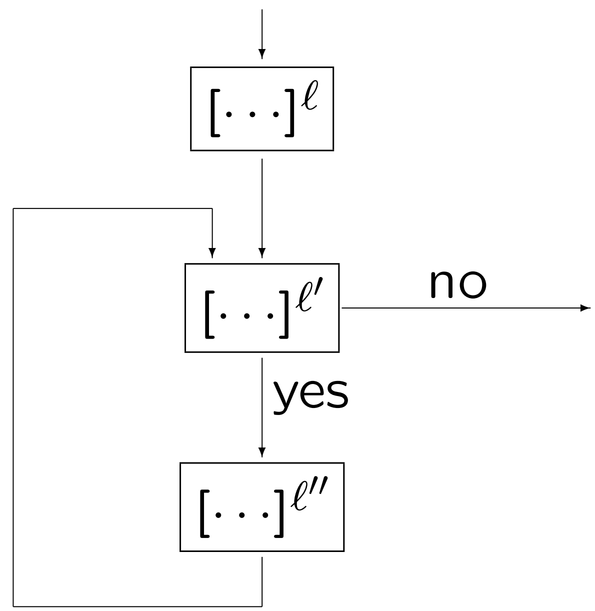
ℓ	$\mathsf{AE}_{entry}(\ell)$	$\mathsf{AE}_{exit}(\ell)$
1	$\emptyset$	$\{\texttt{a+b}\}$
2	$\{\texttt{a+b}\}$	$\{\texttt{a+b}, \texttt{a*b}\}$
3	$\{\texttt{a+b}\}$	$\{\texttt{a+b}\}$
4	$\{\texttt{a+b}\}$	$\emptyset$
5	$\emptyset$	$\{\texttt{a+b}\}$

Why largest solution?

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

Equations:

$$\begin{aligned}
\mathsf{AE}_{entry}(\ell) &= \emptyset \\
\mathsf{AE}_{entry}(\ell') &= \mathsf{AE}_{exit}(\ell)\ \cap\ \mathsf{AE}_{exit}(\ell'') \\
\mathsf{AE}_{entry}(\ell'') &= \mathsf{AE}_{exit}(\ell') \\[4pt]
\mathsf{AE}_{exit}(\ell) &= \mathsf{AE}_{entry}(\ell) \cup \{\texttt{x+y}\} \\
\mathsf{AE}_{exit}(\ell') &= \mathsf{AE}_{entry}(\ell') \\
\mathsf{AE}_{exit}(\ell'') &= \mathsf{AE}_{entry}(\ell'')
\end{aligned}$$



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:

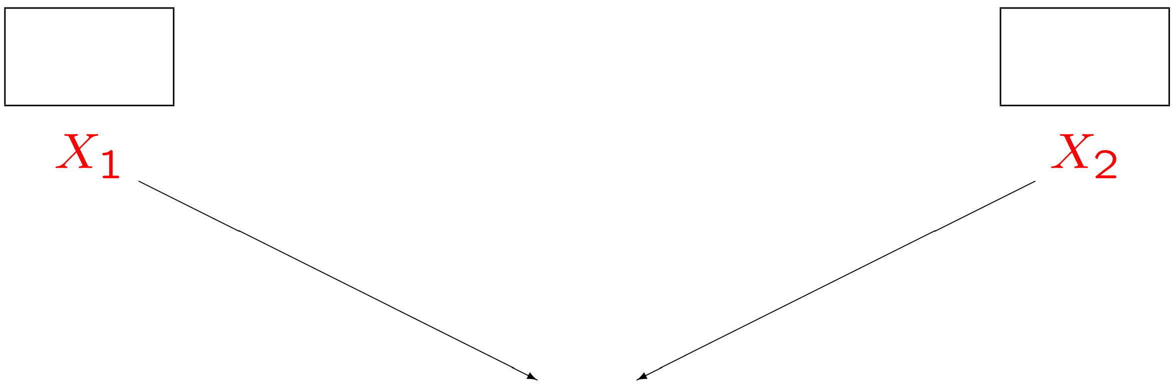
point of interest

$\Downarrow$

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

useful for definition-use chains and use-definition chains

Reaching Definitions Analysis – the basic idea



$$N = X_1 \cup X_2$$

$$\boxed{[x := a]^\ell}$$

$$X = (N \setminus \overbrace{\{(x,?),(x,1),\cdots\}}^{\text{kill}}) \cup \underbrace{\{(x,\ell)\}}_{\text{gen}}$$

Reaching Definitions Analysis

kill and *gen* functions

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

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

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

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

Example:

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

kill and *gen* functions:

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

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

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.):

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

Smallest solution:

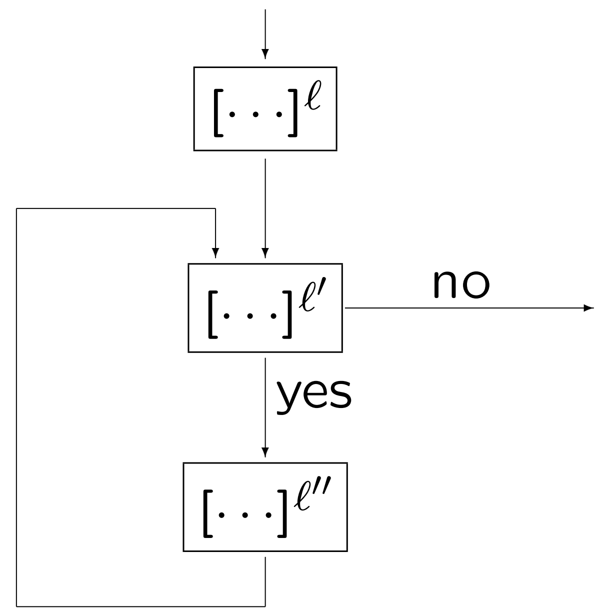
ℓ	$\mathsf{RD}_{entry}(\ell)$	$\mathsf{RD}_{exit}(\ell)$
1	$\{(x,?),(y,?)\}$	$\{(y,?),(x,1)\}$
2	$\{(y,?),(x,1)\}$	$\{(x,1),(y,2)\}$
3	$\{(x,1),(y,2),(y,4),(x,5)\}$	$\{(x,1),(y,2),(y,4),(x,5)\}$
4	$\{(x,1),(y,2),(y,4),(x,5)\}$	$\{(x,1),(y,4),(x,5)\}$
5	$\{(x,1),(y,4),(x,5)\}$	$\{(y,4),(x,5)\}$

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

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

The analysis enables a transformation into

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

Very Busy Expressions Analysis – the basic idea

$$N = (X \setminus \overbrace{\{\text{all expressions with an } x\}}^{\text{kill}})$$

$$\cup \underbrace{\{\text{all subexpressions of } a\}}_{\text{gen}}$$

$$\boxed{x := a}$$

$$X = N_1 \cap N_2$$

N_1 N_2

Very Busy Expressions Analysis

kill and *gen* functions

$$
\begin{array}{rcl}
\mathit{kill}_{\mathsf{VB}}([x := a]^\ell) & = & \{a' \in \mathbf{AExp}_\star \mid x \in \mathit{FV}(a')\} \\
\mathit{kill}_{\mathsf{VB}}([\mathtt{skip}]^\ell) & = & \emptyset \\
\mathit{kill}_{\mathsf{VB}}([b]^\ell) & = & \emptyset \\
\\
\mathit{gen}_{\mathsf{VB}}([x := a]^\ell) & = & \mathbf{AExp}(a) \\
\mathit{gen}_{\mathsf{VB}}([\mathtt{skip}]^\ell) & = & \emptyset \\
\mathit{gen}_{\mathsf{VB}}([b]^\ell) & = & \mathbf{AExp}(b)
\end{array}
$$

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

$$
\begin{array}{rcl}
\mathsf{VB}_{exit}(\ell) & = & \begin{cases} \emptyset & \text{if } \ell \in \mathit{final}(S_\star) \\ \bigcap\{\mathsf{VB}_{entry}(\ell') \mid (\ell', \ell) \in \mathit{flow}^R(S_\star)\} & \text{otherwise} \end{cases} \\
\\
\mathsf{VB}_{entry}(\ell) & = & (\mathsf{VB}_{exit}(\ell) \backslash \mathit{kill}_{\mathsf{VB}}(B^\ell)) \cup \mathit{gen}_{\mathsf{VB}}(B^\ell) \\
 & & \text{where } B^\ell \in \mathit{blocks}(S_\star)
\end{array}
$$

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	$\mathit{kill}_{\mathsf{VB}}(\ell)$	$\mathit{gen}_{\mathsf{VB}}(\ell)$
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:

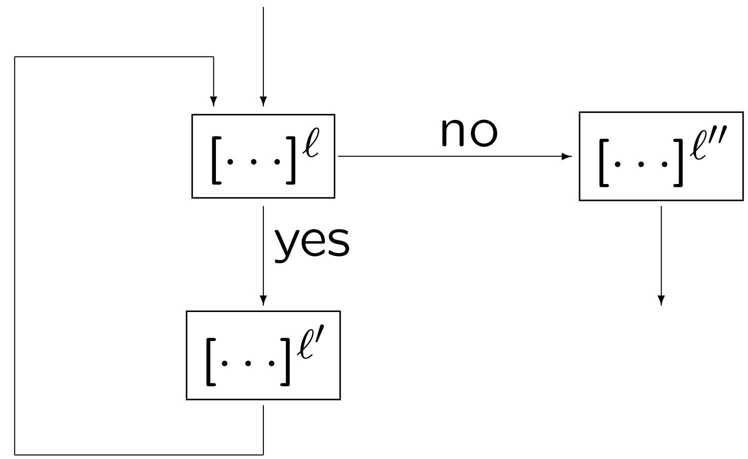
ℓ	$\mathsf{VB}_{entry}(\ell)$	$\mathsf{VB}_{exit}(\ell)$
1	$\{\texttt{a-b}, \texttt{b-a}\}$	$\{\texttt{a-b}, \texttt{b-a}\}$
2	$\{\texttt{a-b}, \texttt{b-a}\}$	$\{\texttt{a-b}\}$
3	$\{\texttt{a-b}\}$	$\emptyset$
4	$\{\texttt{a-b}, \texttt{b-a}\}$	$\{\texttt{a-b}\}$
5	$\{\texttt{a-b}\}$	$\emptyset$

Why largest solution?

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

Equations:

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



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

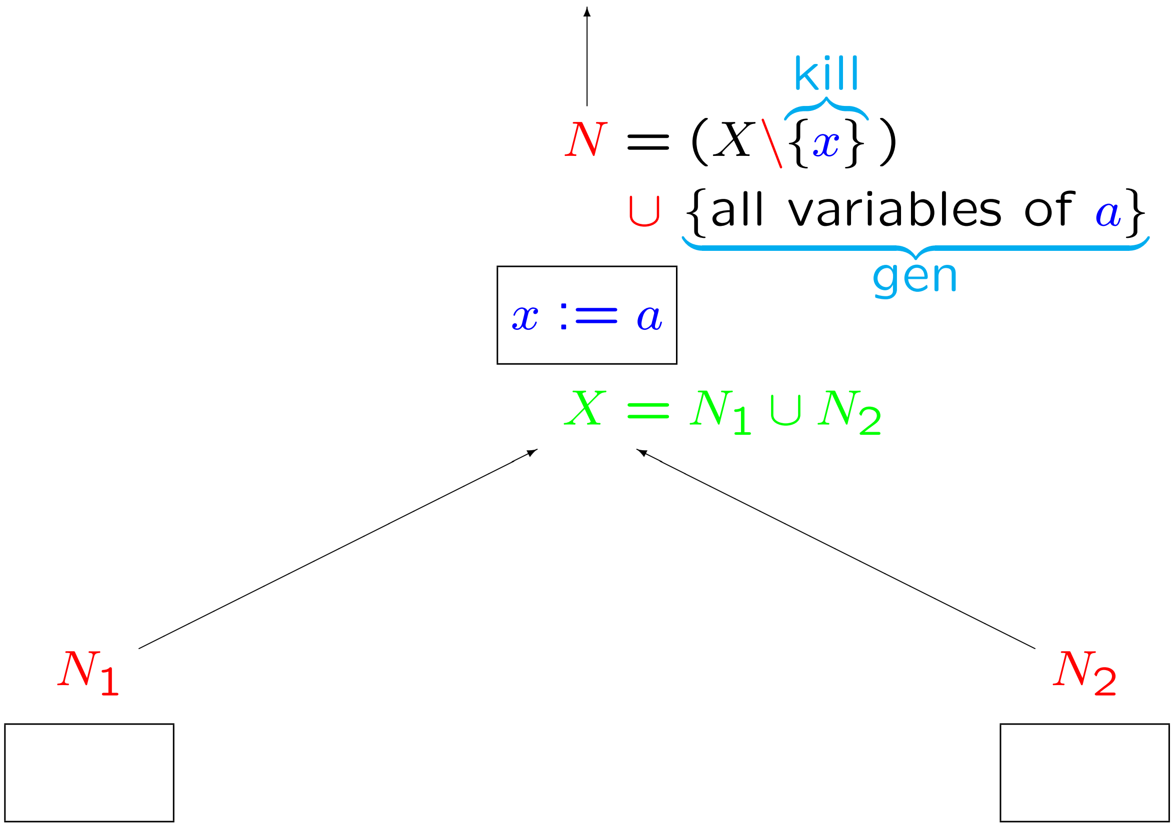
$\Downarrow$

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

The analysis enables a transformation into

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

Live Variables Analysis – the basic idea



Live Variables Analysis

kill and *gen* functions

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

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

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

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

$$\mathsf{LV}_{entry}(\ell) = (\mathsf{LV}_{exit}(\ell) \backslash kill_{\mathsf{LV}}(B^\ell)) \cup gen_{\mathsf{LV}}(B^\ell)$$
$$\text{where } B^\ell \in blocks(S_\star)$$

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	$\mathit{kill}_{\mathsf{LV}}(\ell)$	$\mathit{gen}_{\mathsf{LV}}(\ell)$
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.):

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

Equations:

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

Example (cont.):

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

Smallest solution:

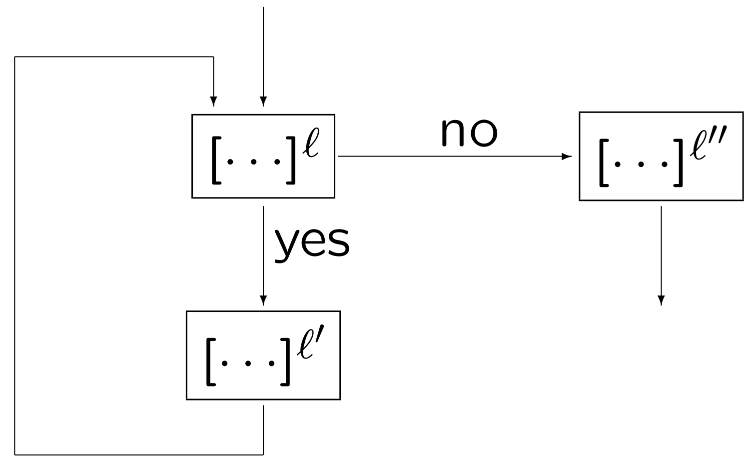
ℓ	$\mathsf{LV}_{entry}(\ell)$	$\mathsf{LV}_{exit}(\ell)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{\texttt{y}\}$
3	$\{\texttt{y}\}$	$\{\texttt{x}, \texttt{y}\}$
4	$\{\texttt{x}, \texttt{y}\}$	$\{\texttt{y}\}$
5	$\{\texttt{y}\}$	$\{\texttt{z}\}$
6	$\{\texttt{y}\}$	$\{\texttt{z}\}$
7	$\{\texttt{z}\}$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

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

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

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

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

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

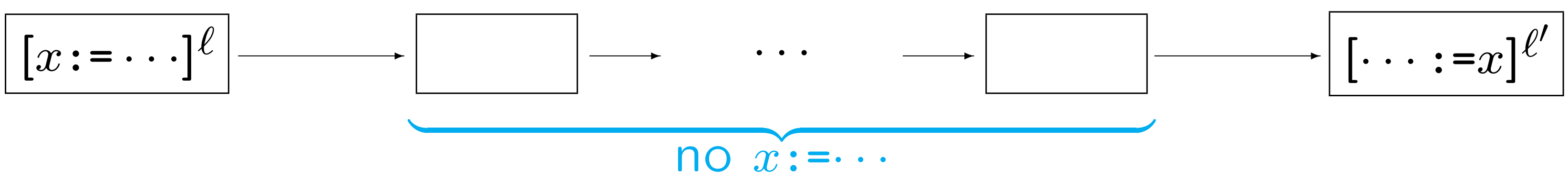
ud chains

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

given by

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

where



- $\mathit{def}(x, \ell)$ means that the block ℓ assigns a value to x
- $\mathit{clear}(x, \ell, \ell')$ 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

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

$$\boxed{[x := \cdots]^{\ell}} \longrightarrow \underbrace{\boxed{} \longrightarrow \;\cdots\; \longrightarrow \boxed{}}_{\text{no } x := \cdots} \longrightarrow \boxed{[\cdots := x]^{\ell'}}$$

One can show that:

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

Example:

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

$\mathit{ud}(x,\ell)$	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	$\emptyset$	$\{6\}$	$\{4,5\}$

$\mathit{du}(x,\ell)$	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

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

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

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

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

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

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

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

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

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

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

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

Example:

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

$$\begin{array}{ll}
\longrightarrow & \langle [\mathtt{z:=1}]^2; \mathtt{while}\ [\mathtt{y>1}]^3\ \mathtt{do}\ ([\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5); [\mathtt{y:=0}]^6, \sigma_{330}\rangle \\
\longrightarrow & \langle \mathtt{while}\ [\mathtt{y>1}]^3\ \mathtt{do}\ ([\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5); [\mathtt{y:=0}]^6, \sigma_{331}\rangle \\
\longrightarrow & \langle [\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5; \\
& \quad \mathtt{while}\ [\mathtt{y>1}]^3\ \mathtt{do}\ ([\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5); [\mathtt{y:=0}]^6, \sigma_{331}\rangle \\
\longrightarrow & \langle [\mathtt{y:=y-1}]^5; \mathtt{while}\ [\mathtt{y>1}]^3\ \mathtt{do}\ ([\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5); [\mathtt{y:=0}]^6, \sigma_{333}\rangle \\
\longrightarrow & \langle \mathtt{while}\ [\mathtt{y>1}]^3\ \mathtt{do}\ ([\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5); [\mathtt{y:=0}]^6, \sigma_{323}\rangle \\
\longrightarrow & \langle [\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5; \\
& \quad \mathtt{while}\ [\mathtt{y>1}]^3\ \mathtt{do}\ ([\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5); [\mathtt{y:=0}]^6, \sigma_{323}\rangle \\
\longrightarrow & \langle [\mathtt{y:=y-1}]^5; \mathtt{while}\ [\mathtt{y>1}]^3\ \mathtt{do}\ ([\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5); [\mathtt{y:=0}]^6, \sigma_{326}\rangle \\
\longrightarrow & \langle \mathtt{while}\ [\mathtt{y>1}]^3\ \mathtt{do}\ ([\mathtt{z:=z*y}]^4; [\mathtt{y:=y-1}]^5); [\mathtt{y:=0}]^6, \sigma_{316}\rangle \\
\longrightarrow & \langle [\mathtt{y:=0}]^6, \sigma_{316}\rangle \\
\longrightarrow & \sigma_{306}
\end{array}$$

Equations and Constraints

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

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

$$\mathsf{LV}_{entry}(\ell) \;\boxed{=}\; (\mathsf{LV}_{exit}(\ell) \backslash \mathit{kill}_{\mathsf{LV}}(B^\ell)) \cup \mathit{gen}_{\mathsf{LV}}(B^\ell)$$
$$\text{where } B^\ell \in \mathit{blocks}(S_\star)$$

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

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

$$\mathsf{LV}_{entry}(\ell) \;\boxed{\supseteq}\; (\mathsf{LV}_{exit}(\ell) \backslash \mathit{kill}_{\mathsf{LV}}(B^\ell)) \cup \mathit{gen}_{\mathsf{LV}}(B^\ell)$$
$$\text{where } B^\ell \in \mathit{blocks}(S_\star)$$

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

$$\begin{tikzcd}
\langle S, \sigma_1 \rangle \arrow[r] \arrow[d, leftrightarrow, "\models \mathsf{LV}^{\subseteq}"] & \langle S', \sigma_1' \rangle \arrow[r] \arrow[d, leftrightarrow, "\models \mathsf{LV}^{\subseteq}"] & \cdots \arrow[r] & \langle S'', \sigma_1'' \rangle \arrow[r] \arrow[d, leftrightarrow, "\models \mathsf{LV}^{\subseteq}"] & \sigma_1''' \\
\mathit{live} & \mathit{live} & \cdots & \mathit{live} &
\end{tikzcd}$$

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.

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

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

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

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

$$
\begin{aligned}
V''' &= \mathit{live}_{exit}(\mathit{init}(S'')) \\
&= \mathit{live}_{exit}(\ell) \\
&\quad \text{for some } \ell \in \mathit{final}(S)
\end{aligned}
$$

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 $\perp$ 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 $\bigsqcup$ 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:

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

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

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

Constraints of the Instance:

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

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

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

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

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

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

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

The analysis enables a transformation into

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

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_ℓ^{CP}

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

$[\mathtt{skip}]^\ell$: $f_\ell^{\mathsf{CP}}(\widehat{\sigma}) = \widehat{\sigma}$

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

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: $\mathit{MFP}_\circ, \mathit{MFP}_\bullet$

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] := $\perp_L$;

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 ($\mathit{MFP}_\circ$ and $\mathit{MFP}_\bullet$)

for all ℓ in F or E do
  $\mathit{MFP}_\circ(\ell)$:= Analysis[ℓ];
  $\mathit{MFP}_\bullet(\ell)$:= 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:

$$\mathit{MOP}_\circ(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in \mathit{path}_\circ(\ell)\}$$

The solution up to and including ℓ:

$$\mathit{MOP}_\bullet(\ell) = \bigsqcup\{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in \mathit{path}_\bullet(\ell)\}$$

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:

$$\mathit{MFP}_\circ \sqsupseteq \mathit{MOP}_\circ \text{ and } \mathit{MFP}_\bullet \sqsupseteq \mathit{MOP}_\bullet$$

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

$$\mathit{MFP}_\circ = \mathit{MOP}_\circ \text{ and } \mathit{MFP}_\bullet = \mathit{MOP}_\bullet$$

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 $\{1,\cdots,9\}$; 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 $i_1 = 1$.

$$
\begin{array}{l}
\texttt{x:=}[\![u_1]\!]\texttt{; y:=}[\![v_1]\!]\texttt{;} \\
\texttt{while } [\cdots] \texttt{ do} \\
\quad \texttt{(if } [\cdots] \texttt{ then x:=x * } 10^{|u_1|} \texttt{ + } [\![u_1]\!]\texttt{; y:=y * } 10^{|v_1|} \texttt{ + } [\![v_1]\!] \texttt{ else} \\
\quad \vdots \\
\quad \texttt{ if } [\cdots] \texttt{ then x:=x * } 10^{|u_n|} \texttt{ + } [\![u_n]\!]\texttt{; y:=y * } 10^{|v_n|} \texttt{ + } [\![v_n]\!] \texttt{ else skip)} \\
[\texttt{z:=abs((x-y)*(x-y))}]^{\ell}
\end{array}
$$

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