# *Intra-Procedural Dataflow Analysis*

## *Forward Analyses*

Jens Knoop, Markus Schordan, Jakob Zwirchmayr

Institut für Computersprachen
Technische Universität Wien

# Formalising the Development

- the programming language of interest
  - abstract syntax
  - labelled program fragments
- abstract flow graphs
  - control and data flow between labelled program fragments
- extract equations from the program
  - specify the information to be computed at entry and exit of labeled fragments
- compute the solution to the equations
  - work list algorithms
  - compute entry and exit information at entry and exit of labeled fragments

# *WHILE Language*

Syntactic categories

| | | |
|---|---|---|
| $a$ | $\in$ AExp | arithmetic expressions |
| $b$ | $\in$ BExp | boolean expressions |
| $S$ | $\in$ Stmt | statements |

| | | |
|---|---|---|
| $x, y$ | $\in$ Var | variables |
| $n$ | $\in$ Num | numerals |
| $\ell$ | $\in$ Lab | labels |

| | | |
|---|---|---|
| $op_a$ | $\in \mathsf{Op}_a$ | arithmetic operators |
| $op_b$ | $\in \mathsf{Op}_b$ | boolean operators |
| $op_r$ | $\in \mathsf{Op}_r$ | relational operators |

# *Abstract Syntax*

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

Assignments and tests are (uniquely) labelled to allow analyses to refer to these program fragments – the labels correspond to pointers into the syntax tree. We use abstract syntax and insert parenthesis to disambiguate syntax.

We will often refer to labelled fragments as *elementary blocks*.

# Auxiliary Functions for Flow Graphs

| | |
|---|---|
| labels(S) | set of nodes of flow graphs of $S$ |
| init(S) | initial node of flow graph of $S$; the unique node where execution of program starts |
| final(S) | final nodes of flow graph for $S$; set of nodes where program execution may terminate |
| flow(S) | edges of flow graphs for $S$ (used for forward analyses) |
| flow$^R$(S) | reverse edges of flow graphs for $S$ (used for backward analyses) |
| blocks(S) | set of elementary blocks in a flow graph |

# Computing the Information (1)

| $S$ | labels$(S)$ | init$(S)$ | final$(S)$ |
|---|---|---|---|
| $[x := a]^\ell$ | $\{\ell\}$ | $\ell$ | $\{\ell\}$ |
| $[\mathsf{skip}]^\ell$ | $\{\ell\}$ | $\ell$ | $\{\ell\}$ |
| $S_1; S_2$ | labels$(S_1)$ $\cup$ labels$(S_2)$ | init$(S_1)$ | final$(S_2)$ |
| if $[b]^\ell$ then $(S_1)$ else $(S_2)$ | $\{\ell\}$ $\cup$ labels$(S_1)$ $\cup$ labels$(S_2)$ | $\ell$ | final$(S_1)$ $\cup$ final$(S_2)$ |
| while $[b]^\ell$ do $S$ od | $\{\ell\} \cup$ labels$(S)$ | $\ell$ | $\{\ell\}$ |

# Computing the Information (2)

| $S$ | $\mathsf{flow}(S)$ | $\mathsf{blocks}(S)$ |
|---|---|---|
| $[x := a]^\ell$ | $\emptyset$ | $\{[x := a]^\ell\}$ |
| $[\mathsf{skip}]^\ell$ | $\emptyset$ | $\{[\mathsf{skip}]^\ell\}$ |
| $S_1; S_2$ | $\mathsf{flow}(S_1) \cup \mathsf{flow}(S_2) \cup \{(\ell, \mathsf{init}(S_2)) \mid \ell \in \mathsf{final}(S_1)\}$ | $\mathsf{blocks}(S_1) \cup \mathsf{blocks}(S_2)$ |
| if $[b]^\ell$ then $(S_1)$ else $(S_2)$ | $\mathsf{flow}(S_1) \cup \mathsf{flow}(S_2) \cup \{(\ell, \mathsf{init}(S_1)), (\ell, \mathsf{init}(S_2))\}$ | $\{[b]^\ell\} \cup \mathsf{blocks}(S_1) \cup \mathsf{blocks}(S_2)$ |
| while $[b]^\ell$ do $S$ od | $\{(\ell, \mathsf{init}(S))\} \cup \mathsf{flow}(S) \cup \{(\ell', \ell) \mid \ell' \in \mathsf{final}(S)\}$ | $\{[b]^\ell\} \cup \mathsf{blocks}(S)$ |

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

# *Program of Interest*

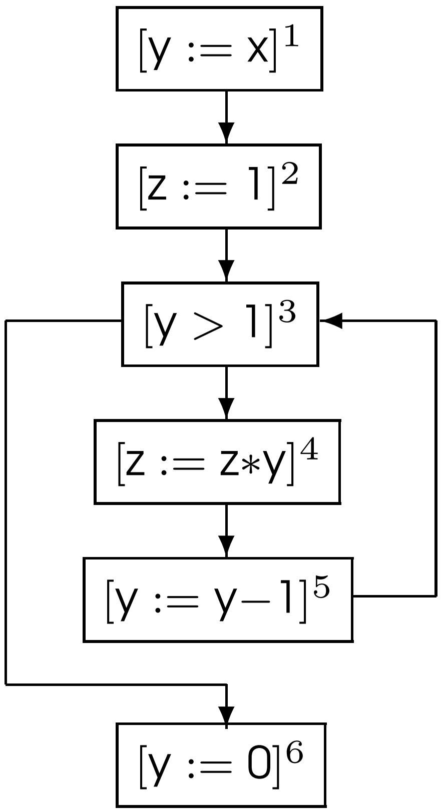
We shall use the notation

- $S_\star$ to represent the program being analyzed (the "top level" statement)
- Lab$_\star$ to represent the labels (labels$(S_\star)$) appearing in $S_\star$
- Var$_\star$ to represent the variables (FV$(S_\star)$) appearing in $S_\star$
- Blocks$_\star$ to represent the elementary blocks (blocks$(S_\star)$) occuring in $S_\star$
- AExp$_\star$ to represent the set of *non-trivial* arithmetic subexpressions in $S_\star$; an expression is trivial if it is a single variable or constant
- AExp$(a)$, AExp$(b)$ to refer to the set of non-trivial arithmetic subexpressions of a given arithmetic, respectively boolean, expression

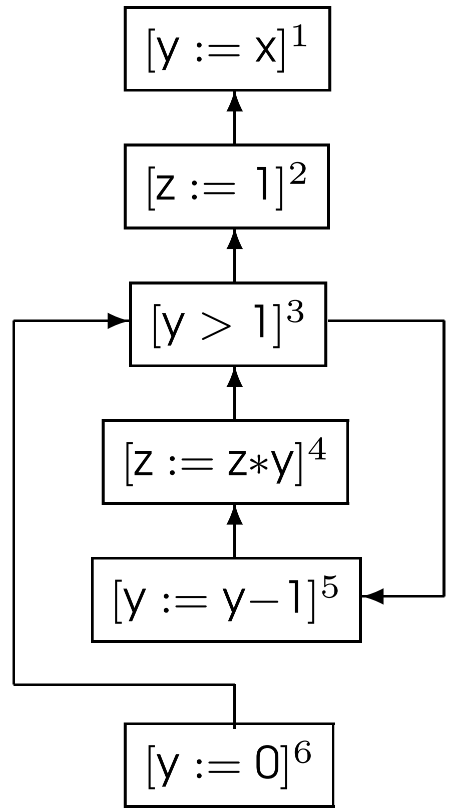
# Example Flow Graphs

Example:

$[\mathsf{y} := \mathsf{x}]^1; [\mathsf{z} := 1]^2; \text{while } [\mathsf{y} > 1]^3 \text{ do } [\mathsf{z} := \mathsf{z} * \mathsf{y}]^4; [\mathsf{y} := \mathsf{y} - 1]^5 \text{ od}; [\mathsf{y} := 0]^6$




$$\text{flow}(S_\star) = \{(1,2),(2,3),(3,4),(4,5),(5,3),(3,6)\}$$




$$\text{flow}^R(S_\star) = \{(6,3),(3,5),(5,4),(4,3),(3,2),(2,1)\}$$

# Example

Example:

$[\mathsf{y} := \mathsf{x}]^1; [\mathsf{z} := 1]^2; \text{while } [\mathsf{y} > 1]^3 \text{ do } [\mathsf{z} := \mathsf{z} * \mathsf{y}]^4; [\mathsf{y} := \mathsf{y} - 1]^5 \text{ od}; [\mathsf{y} := 0]^6$

$$
\begin{array}{rcl}
\mathsf{labels}(S_\star) & = & \{1,2,3,4,5,6\} \\
\mathsf{init}(S_\star) & = & 1 \\
\mathsf{final}(S_\star) & = & \{6\} \\
\mathsf{flow}(S_\star) & = & \{(1,2),(2,3),(3,4),(4,5),(5,3),(3,6)\} \\
\mathsf{flow}^R(S_\star) & = & \{(6,3),(3,5),(5,4),(4,3),(3,2),(2,1)\} \\
\mathsf{blocks}(S_\star) & = & \{[\mathsf{y} := \mathsf{x}]^1, [\mathsf{z} := 1]^2, [\mathsf{y} > 1]^3, \\
 & & \quad [\mathsf{z} := \mathsf{z} * \mathsf{y}]^4, [\mathsf{y} := \mathsf{y} - 1]^5, [\mathsf{y} := 0]^6\}
\end{array}
$$

# Simplifying Assumptions

The program of interest $S_\star$ is often assumed to satisfy:

- $S_\star$ has isolated entries if there are no edges leading into init$(S_\star)$:

$$\forall \ell : (\ell, \mathsf{init}(S_\star)) \notin \mathsf{flow}(S_\star)$$

- $S_\star$ has isolated exits if there are no edges leading out of labels in final$(S_\star)$:

$$\forall \ell \in \mathsf{final}(S_\star), \forall \ell' : (\ell, \ell') \notin \mathsf{flow}(S_\star)$$

- $S_\star$ is label consistent if

$$\forall B_1^{\ell_1}, B_2^{\ell_2} \in \mathsf{blocks}(S_\star) : \ell_1 = \ell_2 \rightarrow B_1 = B_2$$

  This holds if $S_\star$ is uniquely labelled.

# Reaching Definitions Analysis

The aim of the Reaching Definitions Analysis is to determine

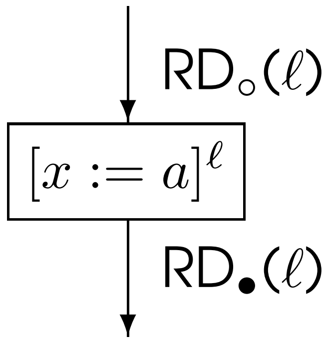
For each program point, which assignments *may* have been made and not overwritten, when program execution reaches this point along some path.

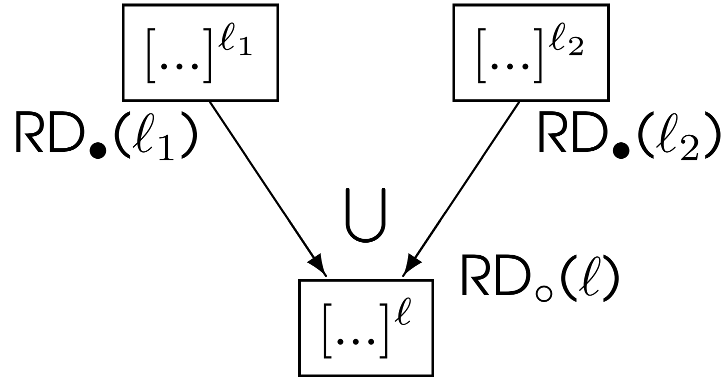
Example:

$[\mathsf{y} := \mathsf{x}]^1; [\mathsf{z} := 1]^2; \text{while } [\mathsf{y} > 1]^3 \text{ do } [\mathsf{z} := \mathsf{z} * \mathsf{y}]^4; [\mathsf{y} := \mathsf{y} - 1]^5 \text{ od}; [\mathsf{y} := 0]^6$

- The assignments labelled 1,2,4,5 reach the entry at 4.
- Only the assignments labelled 1,4,5 reach the entry at 5.

# *Basic Idea*




Analysis information: $RD_\circ(\ell)$,$RD_\bullet(\ell)$ : $Lab_\star \rightarrow \mathcal{P}(Var_\star \times Lab_\star^?)$

- $RD_\circ(\ell)$: the definitions that reach entry of block $\ell$.
- $RD_\bullet(\ell)$: the definitions that reach exit of block $\ell$.

Analysis properties:

- Direction: forward
- May analysis with combination operator $\bigcup$

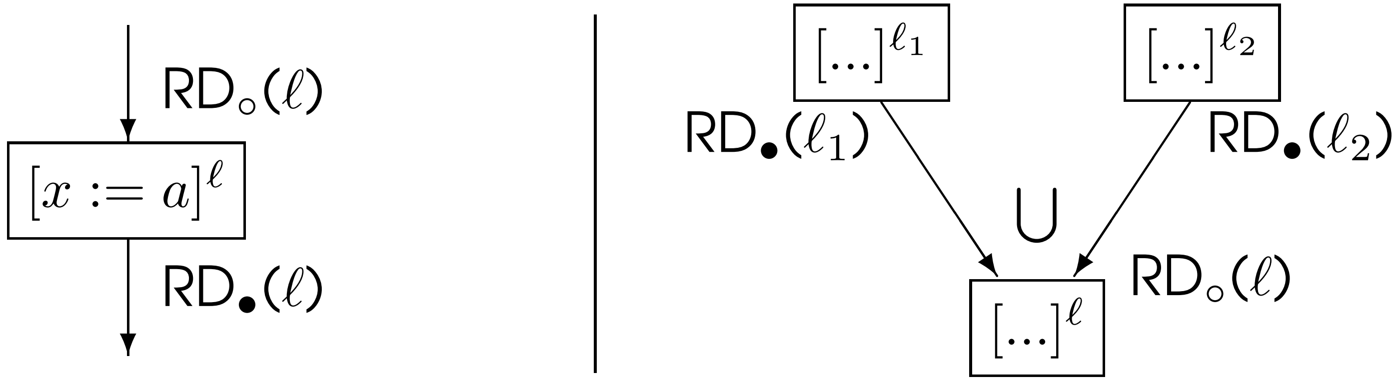
# Analysis of Elementary Blocks

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

Example:
$[\mathsf{x} := \mathsf{y}]^1; [\mathsf{x} := \mathsf{x} + 3]^2;$

- $\mathsf{kill}_{\mathsf{RD}}([\mathsf{x} := \mathsf{y}]^1) = \{(x,?)\} \cup \{(x,1),(x,2)\}$
- $\mathsf{gen}_{\mathsf{RD}}([\mathsf{x} := \mathsf{y}]^1) = \{(x,1)\}$

# Analysis of the Program




$$\mathrm{RD}_\circ(\ell) = \begin{cases} \{(x, ?) \mid x \in FV(S_\star)\} & : \text{ if } \ell = \mathrm{init}(S_\star) \\ \bigcup\{\mathrm{RD}_\bullet(\ell') \mid (\ell', \ell) \in \mathrm{flow}(S_\star)\} & : \text{ otherwise} \end{cases}$$

$$\mathrm{RD}_\bullet(\ell) = (\mathrm{RD}_\circ(\ell) \backslash \mathrm{kill}_{\mathrm{RD}}(B^\ell)) \cup \mathrm{gen}_{\mathrm{RD}}(B^\ell) \quad \text{where } B^\ell \in \mathrm{blocks}(S_\star)$$

# Example

Example:

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

Equations: Let $S_1 = \{(y,?),(y,1),(y,5),(y,6)\}$, $S_2 = \{(z,?),(z,2),(z,4)\}$

$$
\begin{array}{lcl@{\qquad}lcl}
\mathrm{RD}_\circ(1) & = & \{(x,?),(y,?),(z,?)\} & \mathrm{RD}_\bullet(1) & = & \mathrm{RD}_\circ(1) \setminus S_1 \cup \{(y,1)\} \\
\mathrm{RD}_\circ(2) & = & \mathrm{RD}_\bullet(1) & \mathrm{RD}_\bullet(2) & = & \mathrm{RD}_\circ(2) \setminus S_2 \cup \{(z,2)\} \\
\mathrm{RD}_\circ(3) & = & \mathrm{RD}_\bullet(2) \cup \mathrm{RD}_\bullet(5) & \mathrm{RD}_\bullet(3) & = & \mathrm{RD}_\circ(3) \\
\mathrm{RD}_\circ(4) & = & \mathrm{RD}_\bullet(3) & \mathrm{RD}_\bullet(4) & = & \mathrm{RD}_\circ(4) \setminus S_2 \cup \{(z,4)\} \\
\mathrm{RD}_\circ(5) & = & \mathrm{RD}_\bullet(4) & \mathrm{RD}_\bullet(5) & = & \mathrm{RD}_\circ(5) \setminus S_1 \cup \{(y,5)\} \\
\mathrm{RD}_\circ(6) & = & \mathrm{RD}_\bullet(3) & \mathrm{RD}_\bullet(6) & = & \mathrm{RD}_\circ(6) \setminus S_1 \cup \{(y,6)\}
\end{array}
$$

| $\ell$ | $\mathrm{RD}_\circ(\ell)$ | $\mathrm{RD}_\bullet(\ell)$ |
|---|---|---|
| 1 | {(x,?),(y,?),(z,?)} | {(x,?),(y,1),(z,?)} |
| 2 | {(x,?),(y,1),(z,?)} | {(x,?),(z,2),(y,1)} |
| 3 | {(x,?),(z,4),(z,2),(y,5),(y,1)} | {(x,?),(z,4),(z,2),(y,5),(y,1)} |
| 4 | {(x,?),(z,4),(z,2),(y,5),(y,1)} | {(z,4),(x,?),(y,5),(y,1)} |
| 5 | {(z,4),(x,?),(y,5),(y,1)} | {(z,4),(x,?),(y,5)} |
| 6 | {(x,?),(z,4),(z,2),(y,5),(y,1)} | {(z,4),(x,?),(z,2),(y,6)} |

# Solving RD Equations

Input

- a set of reaching definitions equations

Output

- the *least solution* to the equations: $\mathsf{RD}_\circ$

Data structures

- The current analysis result for block entries: $\mathsf{RD}_\circ$
- The worklist W: a list of pairs $(\ell, \ell')$ indicating that the current analysis result has changed at the entry to the block $\ell$ and hence the information must be recomputed for $\ell'$.

# Solving RD Equations - Algorithm

$$
\begin{array}{l}
\texttt{W:=nil;}\\
\texttt{foreach } (\ell,\ell') \in \texttt{flow}(S_\star) \texttt{ do W := cons(}(\ell,\ell')\texttt{,W); od;}\\
\texttt{foreach } \ell \in \texttt{labels}(S_\star) \texttt{ do}\\
\quad \texttt{if } \ell \in \texttt{init}(S_\star) \texttt{ then}\\
\qquad \texttt{RD}_\circ(\ell) := \boxed{\{(x,?) \mid x \in \texttt{FV}(S_\star)\}}\\
\quad \texttt{else}\\
\qquad \texttt{RD}_\circ(\ell) := \boxed{\emptyset}\\
\quad \texttt{fi}\\
\texttt{od}\\
\texttt{while } W \neq nil \texttt{ do}\\
\quad (\ell,\ell') \texttt{ := head(W);}\\
\quad \texttt{W := tail(W);}\\
\quad \texttt{if } (\texttt{RD}_\circ(\ell)\backslash\texttt{kill}_{\texttt{RD}}(B^\ell)) \ \cup\ \texttt{gen}_{\texttt{RD}}(B^\ell) \ \boxed{\not\subseteq}\ \texttt{RD}_\circ(\ell') \texttt{ then}\\
\qquad \texttt{RD}_\circ(\ell') \texttt{ := } \texttt{RD}_\circ(\ell') \ \boxed{\cup}\ (\texttt{RD}_\circ(\ell)\backslash\texttt{kill}_{\texttt{RD}}(B^\ell)) \ \cup\ \texttt{gen}_{\texttt{RD}}(B^\ell)\texttt{;}\\
\qquad \texttt{foreach } \ell'' \texttt{ with } (\ell',\ell'') \texttt{ in } \texttt{flow}(S_\star) \texttt{ do}\\
\qquad\quad \texttt{W := cons(}(\ell',\ell'')\texttt{,W);}\\
\qquad \texttt{od}\\
\quad \texttt{fi}\\
\texttt{od}
\end{array}
$$

# *Use-Definition and Definition-Use Chains*

- Use-Definition chains or *ud* chains

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

  $[\text{x} := 0]^1; [\text{x} := 5]^2; [\text{y} := \text{x}]^3; [\text{z} := \text{x}]^4$

- Definition-Use chains or *du* chains

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

  $[\text{x} := 0]^1; [\text{x} := 5]^2; [\text{y} := \text{x}]^3; [\text{z} := \text{x}]^4$

# UD/DU Chains - Defined via RDs

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

are defined by

$$\mathsf{UD}(x,\ell) = \begin{cases} \{\ell' \mid (x,\ell') \in \mathsf{RD}_\circ(\ell)\} & : \text{ if } x \in \mathsf{used}(B^\ell) \\ \emptyset & : \text{ otherwise} \end{cases}$$

where used($[x := a]^\ell$) = FV($a$), used($[b]^\ell$) = FV($b$), used($[\mathsf{skip}]^\ell$) = $\emptyset$

and

$$\mathsf{DU}(x,\ell) = \{\ell' \mid \ell \in \mathsf{UD}(x,\ell')\}$$

# Available Expressions Analysis

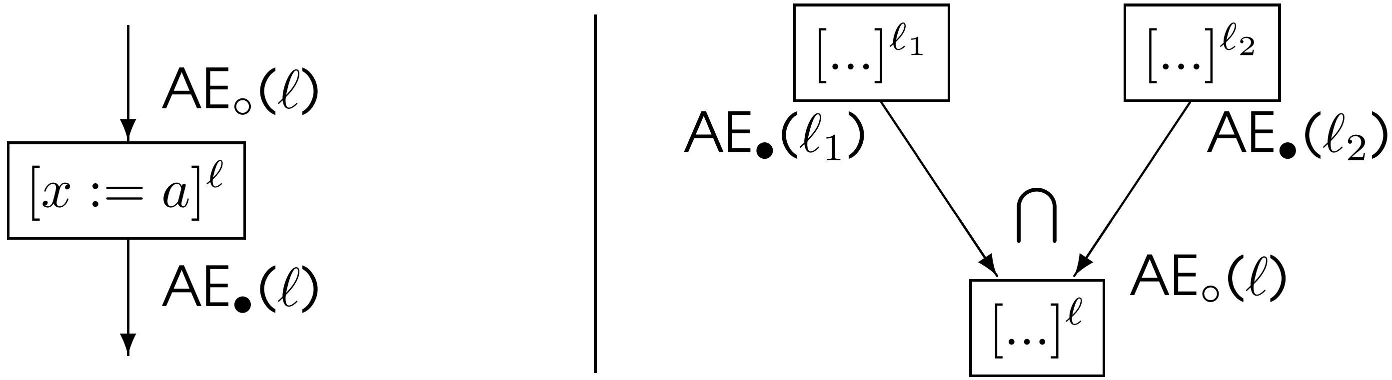
The aim of the Available Expressions Analysis is to determine

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

Example:
$[x := a{+}b]^1; [y := a{*}x]^2;$ while $[y > a{+}b]^3$ do $[a := a + 1]^4; [x := a + b]^5$ od

- No expression is available at the start of the program
- An expression is considered available if no path kills it
- The expression a+b is available every time execution reaches the test in the loop at 3.

# Basic Idea



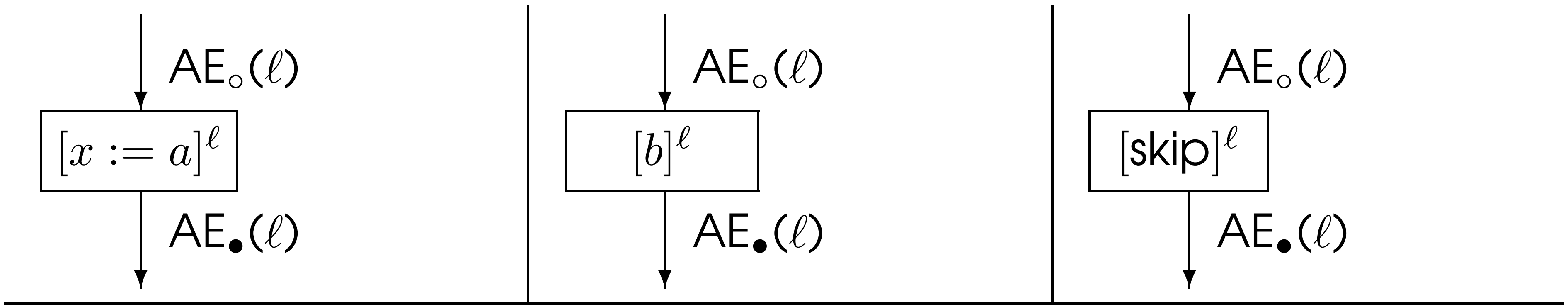

Analysis information: $\mathsf{AE}_\circ(\ell)$,$\mathsf{AE}_\bullet(\ell)$ : $\mathsf{Lab}_\star \rightarrow \mathcal{P}(\mathsf{AExp}_\star)$

- $\mathsf{AE}_\circ(\ell)$: the expressions that have been comp. at entry of block $\ell$.
- $\mathsf{AE}_\bullet(\ell)$: the expressions that have been comp. at exit of block $\ell$.

Analysis properties:

- Direction: forward
- Must analysis with combination operator $\bigcap$

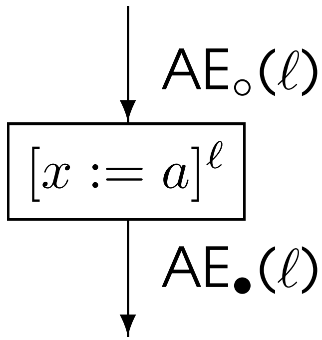
# *Analysis of Elementary Blocks*

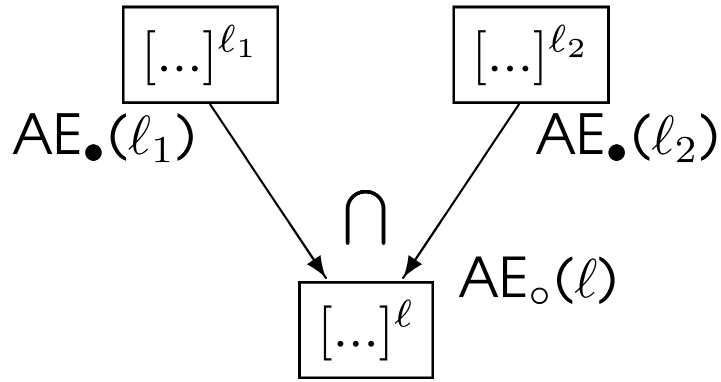



$$
\begin{aligned}
\mathsf{kill}_{\mathsf{AE}}([x := a]^\ell) &= \{a' \in \mathsf{AExp}_\star \mid x \in FV(a')\}\\
\mathsf{kill}_{\mathsf{AE}}([\mathsf{skip}]^\ell) &= \emptyset\\
\mathsf{kill}_{\mathsf{AE}}([b]^\ell) &= \emptyset\\
\mathsf{gen}_{\mathsf{AE}}([x := a]^\ell) &= \{a' \in \mathsf{AExp}(a)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

$$
\mathsf{AE}_\bullet(\ell) = (\mathsf{AE}_\circ(\ell) \setminus \mathsf{kill}_{\mathsf{AE}}(B^\ell)) \cup \mathsf{gen}_{\mathsf{AE}}(B^\ell) \quad \text{where } B^\ell \in \mathsf{blocks}(S_\star)
$$

# Analysis of the Program




$$\mathsf{AE}_\circ(\ell) \quad = \quad \begin{cases} \emptyset & : \quad \text{if } \ell = \mathsf{init}(S_\star) \\ \bigcap\{\mathsf{AE}_\bullet(\ell') | (\ell', \ell) \in \mathsf{flow}(S_\star)\} & : \quad \text{otherwise} \end{cases}$$

$$\mathsf{AE}_\bullet(\ell) \quad = \quad (\mathsf{AE}_\circ(\ell) \backslash \mathsf{kill}_{\mathsf{AE}}(B^\ell)) \ \cup \ \mathsf{gen}_{\mathsf{AE}}(B^\ell) \quad \text{where } B^\ell \in \mathsf{blocks}(S_\star)$$

# Example

Example:

[x := a+b]$^1$; [y := a$*$x]$^2$; while [y $>$ a+b]$^3$ do [a := a + 1]$^4$; [x := a + b]$^5$ od

Equations:

$$
\begin{array}{lcl}
\mathrm{AE}_\circ(1) & = & \emptyset \\
\mathrm{AE}_\circ(2) & = & \mathrm{AE}_\bullet(1) \\
\mathrm{AE}_\circ(3) & = & \mathrm{AE}_\bullet(2) \cap \mathrm{AE}_\bullet(5) \\
\mathrm{AE}_\circ(4) & = & \mathrm{AE}_\bullet(3) \\
\mathrm{AE}_\circ(5) & = & \mathrm{AE}_\bullet(4)
\end{array}
\qquad
\begin{array}{lcl}
\mathrm{AE}_\bullet(1) & = & \mathrm{AE}_\circ(1) \setminus \{a * x\} \cup \{a + b\} \\
\mathrm{AE}_\bullet(2) & = & \mathrm{AE}_\circ(2) \setminus \emptyset \cup \{a * x\} \\
\mathrm{AE}_\bullet(3) & = & \mathrm{AE}_\circ(3) \setminus \emptyset \cup \{a + b\} \\
\mathrm{AE}_\bullet(4) & = & \mathrm{AE}_\circ(4) \setminus \{a + b, a * x, a + 1\} \cup \emptyset \\
\mathrm{AE}_\bullet(5) & = & \mathrm{AE}_\circ(5) \setminus \{a * x\} \cup \{a + b\}
\end{array}
$$

| $\ell$ | $\mathrm{AE}_\circ(\ell)$ | $\mathrm{AE}_\bullet(\ell)$ |
|---|---|---|
| 1 | $\emptyset$ | {a+b} |
| 2 | {a+b} | {a+b,a*x} |
| 3 | {a+b} | {a+b } |
| 4 | {a+b} | $\emptyset$ |
| 5 | $\emptyset$ | {a+b} |

# Solving AE Equations

Input

- a set of available expressions equations

Output

- the *largest solution* to the equations: $\mathrm{AE}_\circ$

Data structures

- The current analysis result for block entries: $\mathrm{AE}_\circ$
- The worklist W: a list of pairs $(\ell, \ell')$ indicating that the current analysis result has changed at the entry to the block $\ell$ and hence the information must be recomputed for $\ell'$.

# Solving AE Equations - Algorithm

```
W:=nil;
foreach (ℓ,ℓ') ∈ flow(S⋆) do W := cons((ℓ,ℓ'),W); od;
foreach ℓ ∈ labels(S⋆) do
   if ℓ ∈ init(S⋆) then
      AE∘(ℓ) := ∅
   else
      AE∘(ℓ) := AExp⋆
   fi
od
while W ≠ nil do
   (ℓ,ℓ') := head(W);
   W := tail(W);
   if (AE∘(ℓ)\kill_AE(B^ℓ)) ∪ gen_AE(B^ℓ) ⊉ AE∘(ℓ') then
      AE∘(ℓ') := AE∘(ℓ') ∩ (AE∘(ℓ)\kill_AE(B^ℓ)) ∪ gen_AE(B^ℓ);
      foreach ℓ'' with (ℓ',ℓ'') in flow(S⋆) do
         W := cons((ℓ',ℓ''),W);
      od
   fi
od
```

# Common Subexpression Elimination (CSE)

The aim is to find computations that are always performed at least twice on a given execution path and to eliminate the second and later occurrences; it uses Available Expressions Analysis to determine the redundant computations.

Example:
$[x := a{+}b]^1; [y := a{*}x]^2;$ while $[y > a{+}b]^3$ do $[a := a + 1]^4; [x := a + b]^5$ od

- Expression a+b is computed at 1 and 5 and recomputation can be eliminated at 3.

# The Optimization - CSE

Let $S_\star^N$ be the normalized form of $S_\star$ such that there is at most one operator on the right hand side of an assignment.

For each $[...a...]^\ell$ in $S_\star^N$ with $a \in \mathsf{AE}_\circ(\ell)$ do

- determine the set $\{[y_1 := a]^{\ell_1}, \ldots, [y_k := a]^{\ell_k}\}$ of elementary blocks in $S_\star^N$ "defining" $a$ that reaches $[...a...]^\ell$
- create a fresh variable $u$ and
  - replace each occurrence of $[y_i := a]^{\ell_i}$ with $[u := a]^{\ell_i}$;$[y_i := u]^{\ell_i'}$ for $1 \leq i \leq k$
  - replace $[...a...]^\ell$ with $[...u...]^\ell$

$[x := a]^{\ell'}$ reaches $[...a...]^\ell$ if there is a path in flow$(S_\star^N)$ from $\ell'$ to $\ell$ that does not contain *any* assignments with expression $a$ on the right hand side and no variable of $a$ is modified.

# Computing the "reaches" Information

$[x := a]^{\ell'}$ reaches $[...a...]^{\ell}$ if there is a path in flow$(S_\star^N)$ from $\ell'$ to $\ell$ that does not contain *any* assignments with expression $a$ on the right hand side and no variable of $a$ is modified.

The set of elementary blocks that reaches $[...a...]^{\ell}$ can be computed as reaches$_\circ(a, \ell)$ where

$$\text{reaches}_\circ(a,\ell) = \begin{cases} \emptyset & : \text{ if } \ell = \text{init}(S_\star) \\ \bigcup \text{reaches}_\bullet(a,\ell') & : \text{ otherwise} \end{cases}$$

$$\text{reaches}_\bullet(a,\ell) = \begin{cases} \{B^\ell\} & : \text{ if } B^\ell \text{ has the form} [x := a]^\ell \text{ and } x \notin \text{FV}(a) \\ \emptyset & : \text{ if } B^\ell \text{ has the form} [x := ...]^\ell \text{ and } x \in \text{FV}(a) \\ \text{reaches}_\circ(a,\ell) & : \text{ otherwise} \end{cases}$$

# Example - CSE

Example:

$[x := a+b]^1; [y := a*x]^2;$ while $[y > a+b]^3$ do $[a := a + 1]^4; [x := a + b]^5$ od

| $\ell$ | $AE_\circ(\ell)$ |
|---|---|
| 1 | $\emptyset$ |
| 2 | $\{a+b\}$ |
| 3 | $\{a+b\}$ |
| 4 | $\{a+b\}$ |
| 5 | $\emptyset$ |

reaches(a+b,3)=$\{[x := a + b]^1, [x := a + b]^5\}$

Result of CSE optimization wrt. reaches(a+b,3)

$[u := a+b]^{1'}; [x := u]^1; [y := a*x]^2;$ while $[y > u]^3$ do $[a := a + 1]^4; [u := a + b]^{5'}; [x := u]^5$ od

# Copy Analysis

The aim of Copy Analysis is to determine for each program point $\ell'$, which copy statements $[x := y]^\ell$ that still are relevant (i.e. neither $x$ nor $y$ have been redefined) when control reaches point $\ell'$.

Example:
$[\mathsf{a} := \mathsf{b}]^1$; if $[\mathsf{x} > \mathsf{b}]^2$ then $([\mathsf{y} := \mathsf{a}]^3)$ else $([\mathsf{b} := \mathsf{b} + 1]^4; [\mathsf{y} := \mathsf{a}]^5)$; $[\mathsf{skip}]^6$

| $\ell$ | $C_\circ(\ell)$ | $C_\bullet(\ell)$ |
|---|---|---|
| 1 | $\emptyset$ | {(a,b)} |
| 2 | {(a,b)} | {(a,b)} |
| 3 | {(a,b)} | {(y,a),(a,b)} |
| 4 | {(a,b)} | $\emptyset$ |
| 5 | $\emptyset$ | {(y,a)} |
| 6 | {(y,a)} | {(y,a)} |

# Copy Propagation (CP)

The aim is to find copy statements $[x := y]^{\ell_j}$ and eliminate them if possible

If $x$ is used in $B^{\ell'}$ then $x$ can be replaced by $y$ in $B^{\ell'}$ provided that

- $[x := y]^{\ell_j}$ is the only kind of definition of $x$ that reaches $B^{\ell'}$ – this information can be obtained from the def-use chain.
- on every path from $\ell_j$ to $\ell'$ (including paths going through $\ell'$ several times but only once through $\ell_j$) there are no redefinitions of $y$; this can be detected by Copy Analysis.

Example 1

[u := a+b]$^{1'}$; [x := u]$^{1}$; [y := a∗x]$^{2}$; while [y > u]$^{3}$ do [a := a + 1]$^{4}$; [u := a + b]$^{5'}$; [x := u]$^{5}$ od

becomes after CP

[u := a+b]$^{1'}$; [y := a∗u]$^{2}$; while [y > u]$^{3}$ do [a := a + 1]$^{4}$; [u := a + b]$^{5'}$; [x := u]$^{5}$ od

# *The Optimization - CP*

For each copy statement $[x := y]^{\ell_j}$ in $S_\star$ do

- determine the set $\{[...x...]^{\ell_1}, ..., [...x...]^{\ell_i}\}$,$1 \leq i \leq k$, of elementary blocks in $S_\star$ that uses $[x := y]^{\ell_j}$ – this can be computed from DU(x,$\ell_j$)
- for each $[...x...]^{\ell_i}$ in this set determine whether $\{(x', y') \in \mathsf{C}_\circ(\ell_i) \mid x' = x\} = \{(x, y)\}$; if so then $[x := y]$ is the only kind of definition of $x$ that reaches $\ell_i$ from all $\ell_j$.
- if this holds for all $i$ ($1 \leq i \leq k$) then
  - remove $[x := y]^{\ell_j}$
  - replace $[...x...]^{\ell_i}$ with $[...y...]^{\ell_i}$ for $1 \leq i \leq k$.

# Examples - CP

## Example 2

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

becomes after CP

$[a := 2]^1; \text{if } [y > u]^2 \text{ then } ([a := a + 1]^3; \quad ;) \text{ else } ([a := a * 2]^5; \quad ;)[y := y*a]^7;$

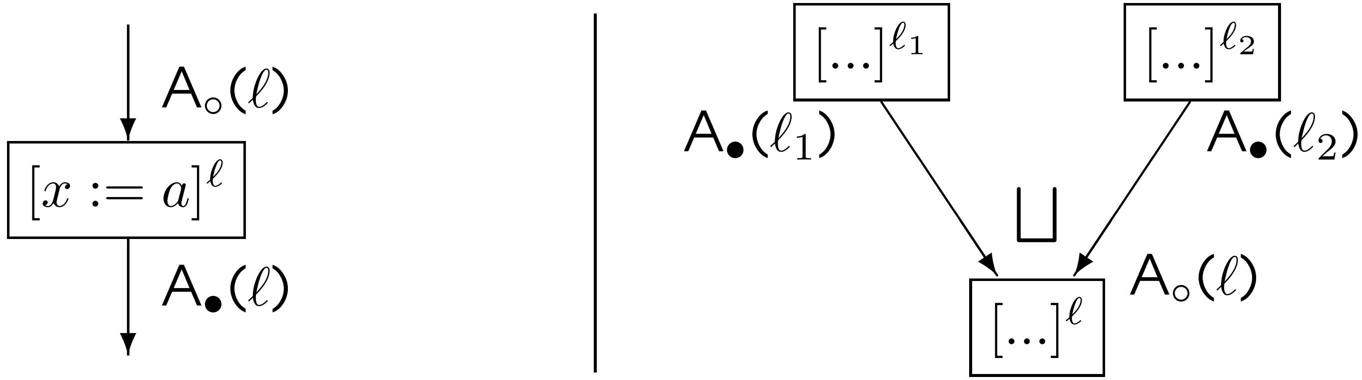
## Example 3

$[a := 10]^1; [b := a]^2; \text{while } [a > 1]^3 \text{ do } [a := a - 1]^4; [b := a]^5; \text{ od } [y := y*b]^6;$

becomes after CP

$[a := 10]^1; \quad ; \text{while } [a > 1]^3 \text{ do } [a := a - 1]^4; \quad ; \text{ od } [y := y*a]^6;$

# Summary: Forward Analyses




$$A_\circ(\ell) = \begin{cases} \iota_A & : \text{ if } \ell = \mathsf{init}(S_\star) \\ \bigsqcup_A \{A_\bullet(\ell') \mid (\ell', \ell) \in \mathsf{flow}(S_\star)\} & : \text{ otherwise} \end{cases}$$

$$A_\bullet(\ell) = (A_\circ(\ell) \backslash \mathsf{kill}_A(B^\ell)) \cup \mathsf{gen}_A(B^\ell) \quad \text{where } B^\ell \in \mathsf{blocks}(S_\star)$$

where

| Analysis | RD | AE |
|---|---|---|
| $\iota_A$ | $\{(x, ?) \mid x \in FV(S_\star)\}$ | $\emptyset$ |
| $\bigsqcup_A$ | $\cup$ | $\cap$ |

# References

- Material for this 2nd lecture

  `www.complang.tuwien.ac.at/knoop/oue185187_ws1112.html`

- Book

  Flemming Nielson, Hanne Riis Nielson, Chris Hankin:

  Principles of Program Analysis.

  Springer, (2nd edition, 452 pages, ISBN 3-540-65410-0), 2005.

  – Chapter 1 (Introduction)
  – Chapter 2 (Data Flow Analysis)