

Intra-Procedural Dataflow Analysis

Forward Analyses

Jens Knoop, Markus Schordan, Gergő Barany

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

$a \in \text{AExp}$ arithmetic expressions
 $b \in \text{BExp}$ boolean expressions
 $S \in \text{Stmt}$ statements

$x, y \in \text{Var}$ variables
 $n \in \text{Num}$ numerals
 $\ell \in \text{Lab}$ labels

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

Abstract Syntax

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

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

Computing the Information (2)

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

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

Program of Interest

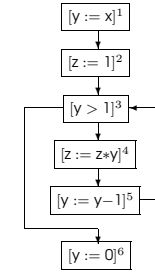
We shall use the notation

- S_* to represent the program being analyzed (the “top level” statement)
- Lab_* to represent the labels ($\text{labels}(S_*)$) appearing in S_*
- Var_* to represent the variables ($\text{FV}(S_*)$) appearing in S_*
- Blocks_* to represent the elementary blocks ($\text{blocks}(S_*)$) occurring in S_*
- AExp_* to represent the set of *non-trivial* arithmetic subexpressions in S_* ; an expression is trivial if it is a single variable or constant
- $\text{AExp}(a)$, $\text{AExp}(b)$ to refer to the set of non-trivial arithmetic subexpressions of a given arithmetic, respectively boolean, expression

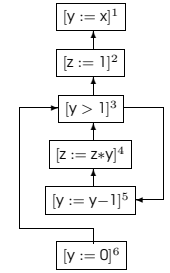
Example Flow Graphs

Example:

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



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



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

Example

Example:

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

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

Simplifying Assumptions

The program of interest S_* is often assumed to satisfy:

- S_* has isolated entries if there are no edges leading into $\text{init}(S_*)$:

$$\forall \ell : (\ell, \text{init}(S_*)) \notin \text{flow}(S_*)$$

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

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

- S_* is label consistent if

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

This holds if S_* is uniquely labelled.

Reaching Definitions Analysis

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

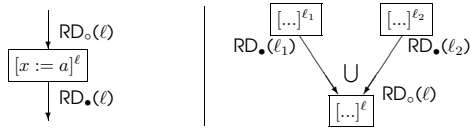
For each program point, which assignments *may* have been made and not overwritten, when program execution reaches this point along some path.

Example:

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

- 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: $RD_o(\ell), RD_e(\ell) : Lab_* \rightarrow \mathcal{P}(Var_* \times Lab_*^2)$

- $RD_o(\ell)$: the definitions that reach **entry** of block ℓ .
- $RD_e(\ell)$: the definitions that reach **exit** of block ℓ .

Analysis properties:

- Direction: forward
- May analysis with combination operator \cup

Analysis of Elementary Blocks

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

Example:

$[x := y]^1; [x := x + 3]^2;$

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

Analysis of the Program



$$\begin{aligned} RD_o(\ell) &= \begin{cases} \{(x, ?) \mid x \in FV(S_*)\} & \text{if } \ell = \text{init}(S_*) \\ \cup \{RD_e(\ell') \mid (\ell', \ell) \in \text{flow}(S_*)\} & \text{otherwise} \end{cases} \\ RD_e(\ell) &= (RD_o(\ell) \setminus \text{kill}_{RD}(B^\ell)) \cup \text{gen}_{RD}(B^\ell) \quad \text{where } B^\ell \in \text{blo} \end{aligned}$$

Example

Example:

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

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

$$\begin{aligned} RD_o(1) &= \{(x, ?), (y, ?), (z, ?)\} & RD_e(1) &= RD_o(1) \setminus S_1 \cup \{(y, 1)\} \\ RD_o(2) &= RD_e(1) & RD_e(2) &= RD_o(2) \setminus S_2 \cup \{(z, 2)\} \\ RD_o(3) &= RD_e(2) \cup RD_e(5) & RD_e(3) &= RD_o(3) \\ RD_o(4) &= RD_e(3) & RD_e(4) &= RD_o(4) \setminus S_2 \cup \{(z, 4)\} \\ RD_o(5) &= RD_e(4) & RD_e(5) &= RD_o(5) \setminus S_1 \cup \{(y, 5)\} \\ RD_o(6) &= RD_e(3) & RD_e(6) &= RD_o(6) \setminus S_1 \cup \{(y, 6)\} \end{aligned}$$

ℓ	$RD_o(\ell)$	$RD_e(\ell)$
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: RD_e .

Data structures

- The current analysis result for block entries: RD_o .
- The worklist W : a list of pairs (ℓ, ℓ') indicating that the current analysis result has changed at the entry to the block ℓ and hence the information must be recomputed for ℓ' .

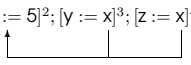
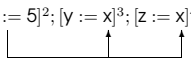
Solving RD Equations - Algorithm

```

W:=nil;
foreach (l, l') in flow(S_*) do W := cons((l, l'), W); od;
foreach l in labels(S_*) do
  if l in init(S_*) then
    RD_o(l) := {(x, ?) | x in FV(S_*)}
  else
    RD_o(l) := {}
  fi
od
while W != nil do
  (l, l') := head(W);
  W := tail(W);
  if (RD_o(l) \ kill_RD(B^l)) union gen_RD(B^l) not subset RD_o(l') then
    RD_o(l') := RD_o(l) union (RD_o(l) \ kill_RD(B^l)) union gen_RD(B^l);
    foreach l'' with (l', l'') in flow(S_*) do
      W := cons((l', l''), W);
    od
  fi
od

```

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

- Definition-Use chains or *du* chains
each assignment of a variable is linked to all uses of it
 $[x := 0]^1; [x := 5]^2; [y := x]^3; [z := x]^4$


UD/DU Chains - Defined via RDs

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

are defined by

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

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

and

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

Available Expressions Analysis

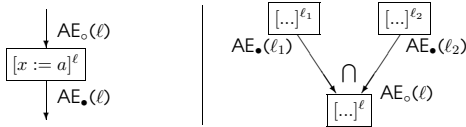
- The aim of the [Available Expressions Analysis](#) is to determine
For each program point, which expressions *must* have already
computed, and not later modified, on all paths to the program
point.

Example:

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

- 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
test in the loop at 3.

Basic Idea



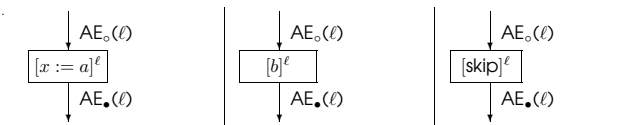
Analysis information: $AE_o(\ell), AE_*(\ell) : \text{Lab}_* \rightarrow \mathcal{P}(\text{AExp}_*)$

- $AE_o(\ell)$: the expressions that have been comp. at **entry** of block ℓ .
- $AE_*(\ell)$: the expressions that have been comp. at **exit** of block ℓ .

Analysis properties:

- Direction: forward
- Must analysis with combination operator \cap

Analysis of Elementary Blocks



$$\text{kill}_{AE}([x := a]^\ell) = \{a' \in \text{AExp}_* \mid x \in FV(a')\}$$

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

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

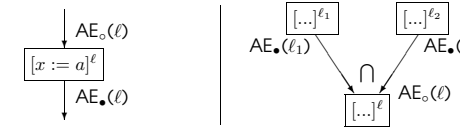
$$\text{gen}_{AE}([x := a]^\ell) = \{a' \in \text{AExp}(a) \mid x \notin FV(a')\}$$

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

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

$$AE_*(\ell) = (AE_o(\ell) \setminus \text{kill}_{AE}(B^\ell)) \cup \text{gen}_{AE}(B^\ell) \quad \text{where } B^\ell \in \text{blocks}(S_*)$$

Analysis of the Program



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

$$AE_*(\ell) = (AE_o(\ell) \setminus \text{kill}_{AE}(B^\ell)) \cup \text{gen}_{AE}(B^\ell) \quad \text{where } B^\ell \in \text{blocks}(S_*)$$

Example

Example:

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

Equations:

$$\begin{aligned} AE_o(1) &= \emptyset & AE_e(1) &= AE_o(1) \setminus \{a * x\} \cup \{a + b\} \\ AE_o(2) &= AE_e(1) & AE_e(2) &= AE_o(2) \setminus \emptyset \cup \{a * x\} \\ AE_o(3) &= AE_e(2) \cap AE_e(5) & AE_e(3) &= AE_o(3) \setminus \emptyset \cup \{a + b\} \\ AE_o(4) &= AE_e(3) & AE_e(4) &= AE_o(4) \setminus \{a + b, a * x, a + 1\} \cup \emptyset \\ AE_o(5) &= AE_e(4) & AE_e(5) &= AE_o(5) \setminus \{a * x\} \cup \{a + b\} \end{aligned}$$

ℓ	$AE_o(\ell)$	$AE_e(\ell)$
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: AE_o .

Data structures

- The current analysis result for block entries: AE_o .
- The worklist W : a list of pairs (ℓ, ℓ') indicating that the current analysis result has changed at the entry to the block ℓ and hence the information must be recomputed for ℓ' .

Solving AE Equations - Algorithm

```

W:=nil;
foreach  $(\ell, \ell') \in \text{flow}(S_*)$  do W := cons(( $\ell, \ell'$ ), W); od;
foreach  $\ell \in \text{labels}(S_*)$  do
  if  $\ell \in \text{init}(S_*)$  then
     $AE_o(\ell) := \emptyset$ 
  else
     $AE_o(\ell) := \text{AExp}_*$ 
  fi
fi
while W  $\neq$  nil do
   $(\ell, \ell') := \text{head}(W)$ ;
  W :=  $\text{tail}(W)$ ;
  if  $(AE_o(\ell) \setminus \text{kill}_{AE}(B^\ell)) \cup \text{gen}_{AE}(B^\ell) \not\subseteq AE_o(\ell')$  then
     $AE_o(\ell') := AE_o(\ell) \cap (AE_o(\ell) \setminus \text{kill}_{AE}(B^\ell)) \cup \text{gen}_{AE}(B^\ell)$ ;
    foreach  $\ell''$  with  $(\ell', \ell'')$  in  $\text{flow}(S_*)$  do
      W := cons(( $\ell', \ell''$ ), W);
    od
  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:

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

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

The Optimization - CSE

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

For each $[...a...]^{\ell}$ in S_*^N with $a \in AE_o(\ell)$ do

- determine the set $\{[y_1 := a]^{\ell_1}, \dots, [y_k := a]^{\ell_k}\}$ of elementary blocks in S_*^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 $\text{flow}(S_*^N)$ from ℓ' to ℓ 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 $\text{flow}(S_*^N)$ from ℓ' to ℓ that does not contain any assignments with expression a on the right side and no variable of a is modified.

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

$$\begin{aligned} \text{reaches}_o(a, \ell) &= \begin{cases} \emptyset & : \text{if } \ell = \text{init}(S_*) \\ \cup \text{reaches}_e(a, \ell') & : \text{otherwise} \end{cases} \\ \text{reaches}_e(a, \ell) &= \begin{cases} \{B^\ell\} & : \text{if } B^\ell \text{ has the form } [x := a]^{\ell} \text{ and } x \notin a \\ \emptyset & : \text{if } B^\ell \text{ has the form } [x := \dots]^{\ell} \text{ and } x \in a \\ \text{reaches}_o(a, \ell) & : \text{otherwise} \end{cases} \end{aligned}$$

Example - CSE

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

ℓ	$AE_o(\ell)$	
1	\emptyset	
2	$\{a+b\}$	reaches(a+b,3) = $\{[x := a + b]^1, [x := a + b]^5\}$
3	$\{a+b\}$	
4	$\{a+b\}$	
5	\emptyset	

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

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

Copy Analysis

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

Example:

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

ℓ	$C_o(\ell)$	$C_\bullet(\ell)$
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 $[x := y]^\ell$ and eliminate them if possible
 If x is used in $B^{\ell'}$ then x can be replaced by y in $B^{\ell'}$ provided that

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

Example 1

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

becomes after CP

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

The Optimization - CP

For each copy statement $[x := y]^\ell$ in S_* do

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

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

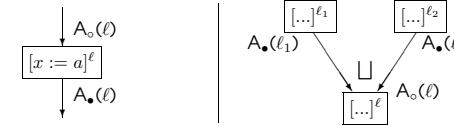
Example 3

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

becomes after CP

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

Summary: Forward Analyses



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

$$A_\bullet(\ell) = (A_o(\ell) \setminus \text{kill}_A(B^\ell)) \cup \text{gen}_A(B^\ell) \quad \text{where } B^\ell \in \text{blocks}$$

where

Analysis	RD	AE
ι_A	$\{(x, ?) \mid x \in FV(S_*)\}$	\emptyset
\bigsqcup_A	\cup	\cap

References

- Material for this 2nd lecture
www.complang.tuwien.ac.at/knoop/oue185187_ws0809.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)