DFA of non-distributive properties

The general pattern of Dataflow Analysis

$$GA_{\odot}(p) = \begin{cases} \iota \text{ if } p \in E \\ \\ \oplus \{\, GA_{\bullet}(q) \mid q \in F \,\} \quad \text{otherwise} \end{cases}$$

$$GA_{\bullet}(p) = f_p\,(\, GA_{\odot}(p) \,)$$

where :

E is the set of initial/final points of the control-flow diagram

ι specifies the initial values

F is the set of successor/predecessor points

$\oplus$ is the combination operator

f is the transfer function associated to node p

Distributive properties

- Monotonicity of a function impiles that

$$f(x \cup y) \supseteq f(x) \cup f(y)$$

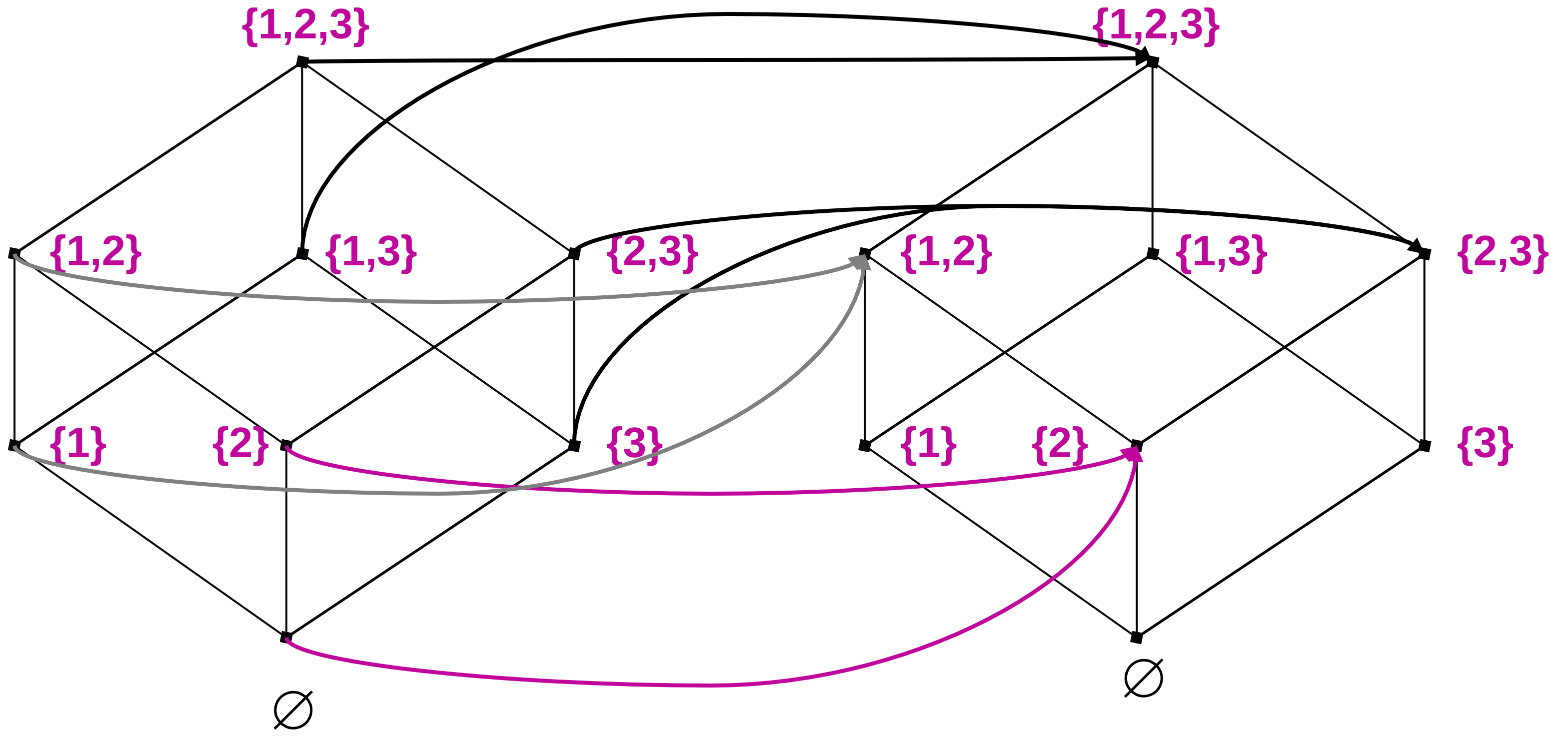
- A function is said distributive a stronger condition hold:

$$f(x \cup y) = f(x) \cup f(y)$$

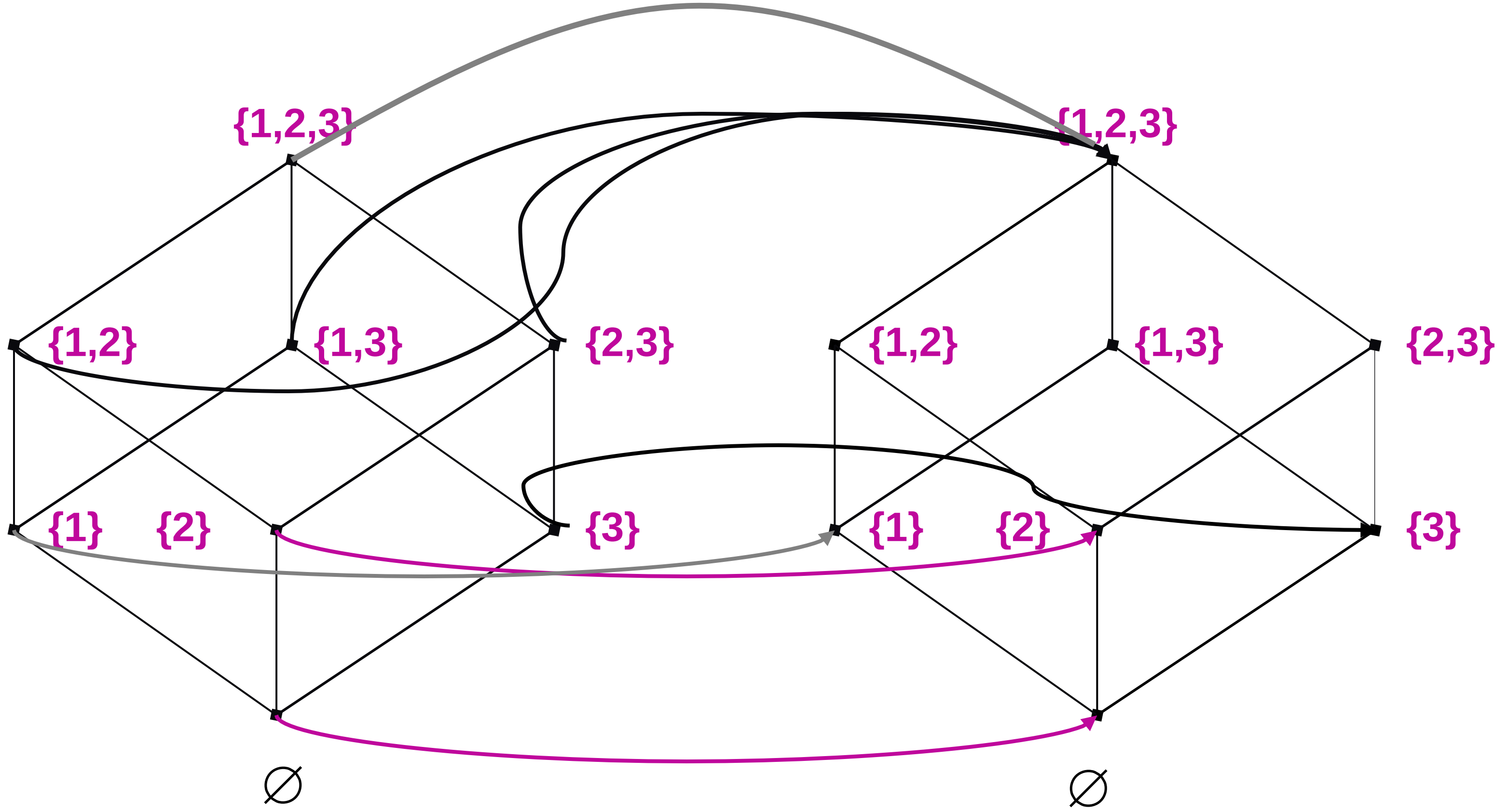
- In general, a dataflow analysis is said distributive if the trasfer functions satisfy

$$f(lub(x,y)) = lub(f(x), f(y))$$

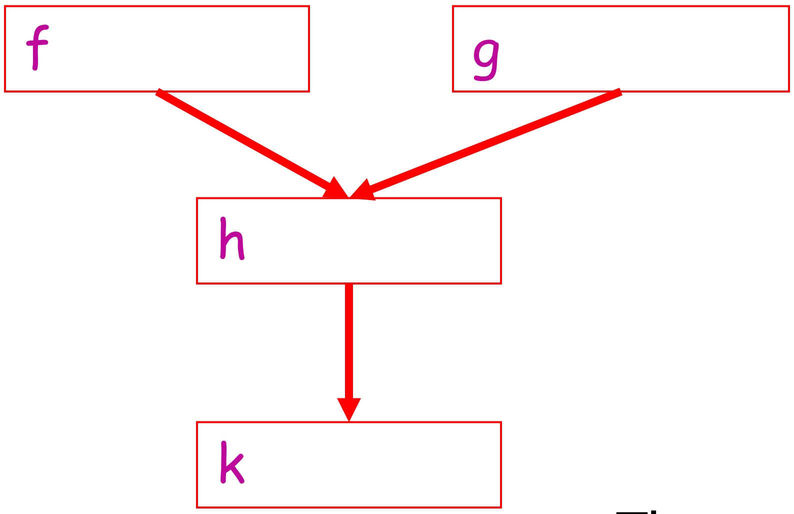
Example: f distributive



Example: f not distributive



Why distributivity is important



k(h(f(0) U g(0))) =

k(h(f(0)) U h(g(0))) =

k(h(f(0))) U k(h(g(0)))

The overall analysis is equal to the lub of the analyses on the different pathes.

DFA of a distributive property

- If the property is distributive, then the minimal solution of the equation system is equivalent to combining the result of the analyses along all the pathes (including infinite pathes).
- In this case the combination operator (least upper bound) does not introduces further loss of accuracy

Which properties are distributive?

- The distributive properties are usually “easy”
- They mainly concern the structure of the program (not the actual values assigned to the variables)
  - E.g., live variables, available expressions, reaching definitions, very busy expressions
  - These properties concern HOW the program pursues the computation, not the actual values of the variables

Non-distributive properties

- They deal with WHAT a program computes
  - E.g.: has the output always the same constant value? Is a variable always assigned a positive number?

- Example: Constant Propagation Analysis

  For each program point, we want to know if a variable is always assigned to exactly the same constant value.

  It is a forward and definite property.

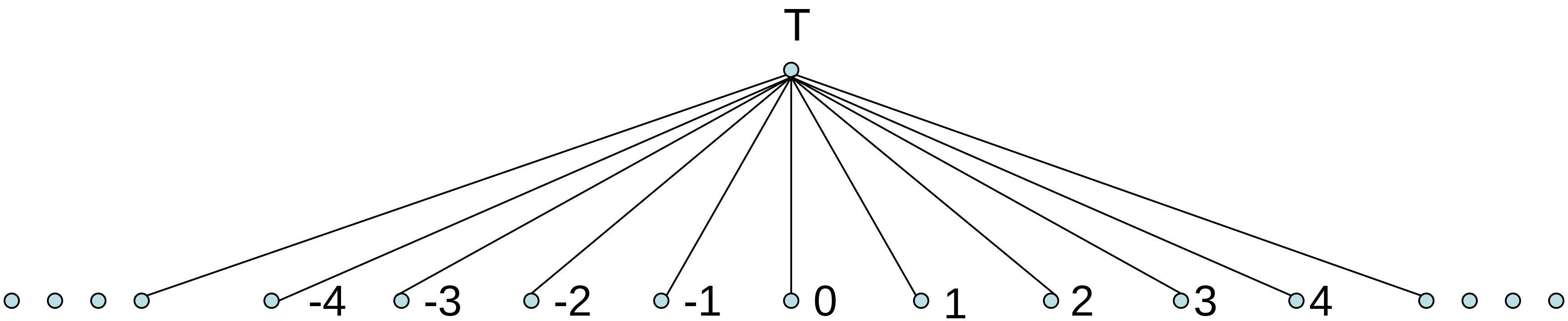
Constant Propagation Analysis

- Consider the set: $(\mathrm{Var} \to \mathbb{Z}^{\top})_{\perp}$
  - Var is the set of variables occurring in the program
  - $\mathbb{Z}^{\top} = \mathbb{Z} \cup \{\top\}$ partially ordered by:

$$\forall\, n \in \mathbb{Z}: \qquad n \leq_{CP} \top$$

$$\forall\, n_1, n_2 \in \mathbb{Z}: \qquad (n_1 \leq_{CP} n_2) \Leftrightarrow (n_1 = n_2)$$

$\mathbb{Z}^{\top}$



- L= $\mathbb{Z} \cup \{\top\}$
- $\forall\, n \in \mathbb{Z}$: $n \le \top$

The lattice (Var $\rightarrow \mathbb{Z}^{\mathsf{T}})_{\perp}$

- In $\mathbb{Z}^{\mathsf{T}}$, the top element T says that a variable is not always assigned to the same constant value (i.e. it may be assigned to different values).

- An element σ: Var $\rightarrow \mathbb{Z}^{\mathsf{T}}$ is a partial function
given a variable x, $\sigma(x)$ tells us if x is a constant or not, and in the positive case (if $\sigma(x)$ is different from T) what is its value.

- The bottom element $\perp$ is added to complete the lattice.

The order in $(\mathrm{Var} \to \mathbb{Z}^{\top})_{\perp}$

- A partial order in $(\mathrm{Var} \to \mathbb{Z}^{\top})_{\perp}$

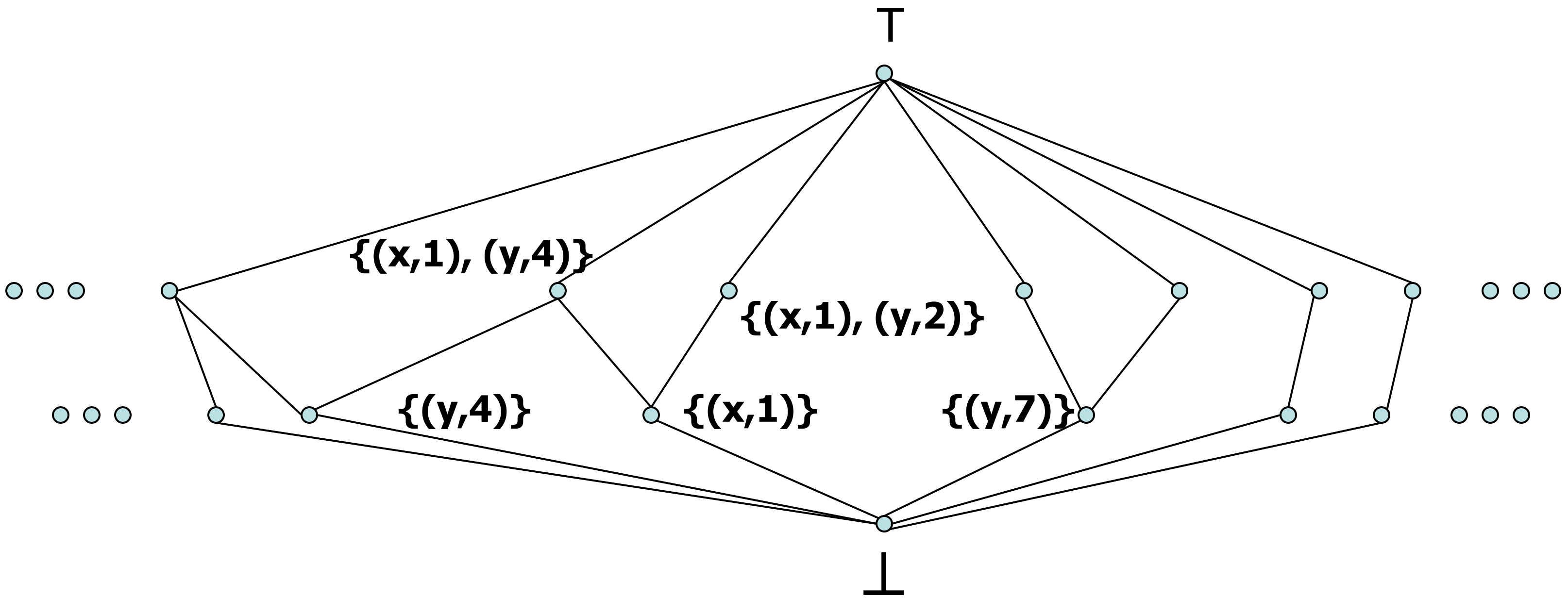
$$\forall\, \sigma \in (\mathrm{Var} \to \mathbb{Z}^{\top})_{\perp} : \qquad \perp \leq \sigma$$
$$\forall\, \sigma_1, \sigma_2 \in (\mathrm{Var} \to \mathbb{Z}^{\top})_{\perp}: (\sigma_1 \leq \sigma_2) \Leftrightarrow (\ \forall x \in \mathrm{dom}(\sigma_1) : \sigma_1(x) \leq_{CP} \sigma_2(x)\ )$$

Means equality when $\sigma_i(x)$ are in Z !

- The least upper bound :

$$\forall\, \sigma \in (\mathrm{Var} \to \mathbb{Z}^{\top})_{\perp} : \mathrm{lub}(\perp, \sigma) = \mathrm{lub}(\sigma, \perp) = \sigma$$
$$\forall\, \sigma_1, \sigma_2 \in (\mathrm{Var} \to \mathbb{Z}^{\top})_{\perp}$$
$$\forall x \in \mathrm{Var} : \mathrm{lub}(\sigma_1, \sigma_2)(x) = \mathrm{lub}(\sigma_1(x), \sigma_2(x))$$

$(\{x,y\} \rightarrow \mathbb{Z}^{\top})_{\perp}$



Expression evaluation

- In order to specify the transfer functions, we have to evaluate an expression given a state σ in $(\text{Var} \to \mathbb{Z}^{\top})_{\perp}$

$$\mathcal{A}: (\text{AExp} \times (\text{Var} \to \mathbb{Z}^{\top})_{\perp}) \to \mathbb{Z}^{\top}_{\perp}$$

$$\mathcal{A}(x,\sigma) = \begin{cases} \perp & \text{if } \sigma = \perp \\ \sigma(x) & \text{otherwise} \end{cases}$$

$$\mathcal{A}(n,\sigma) = \begin{cases} \perp & \text{if } \sigma = \perp \\ n & \text{otherwise} \end{cases}$$

$$\mathcal{A}(a_1 \text{ op } a_2, \sigma) = \mathcal{A}(a_1,\sigma)\ \underline{\text{op}}\ \mathcal{A}(a_2,\sigma)$$

(where $\underline{\text{op}}$ is the corresponding operation of op on $\mathbb{Z}^{\top}_{\perp}$: e.g. 4 $\underline{\text{op}}$ 2 = 6)

Transfer functions

- For Constant Propagation Analysis the set of transfer functions is a subset of

  $\mathcal{F} = \{\, f : (\mathrm{Var} \to \mathbb{Z}^{\top})_{\perp} \to (\mathrm{Var} \to \mathbb{Z}^{\top})_{\perp} \mid f \text{ monotone}\}$

- The trasfer functions f_ℓ are defined by:

  if ℓ is the label of an assignment $[x := a]^\ell$

  $$f_\ell(\sigma) = \begin{cases} \perp & \text{if } \sigma = \perp \\ \sigma[x \to \mathcal{A}(a,\sigma)] & \text{otherwise} \end{cases}$$

  if ℓ is the label of another statement: $f_\ell(\sigma) = \sigma$

Example

- $[x{:=}10]^1;\ [y{:=}x{+}10]^2;\ ([\text{while } x<y]^3\ [y{:=}y{-}1]^4);\ \ [z{:=}x{-}1]^5$

- The minimal solution of the Constant Propagation Analysis of this program is:

- $CP_{\mathtt{entry}}(1) = \emptyset$
  $CP_{\mathtt{exit}}(1) = \{(x \to 10)\}$
  $CP_{\mathtt{entry}}(2) = \{(x \to 10)\}$
  $CP_{\mathtt{exit}}(2) = \{(x \to 10), (y \to 20)\}$
  $CP_{\mathtt{entry}}(3) = CP_{\mathtt{exit}}(3) = CP_{\mathtt{entry}}(4) = CP_{\mathtt{exit}}(4) = \{(x \to 10), (y \to \mathbf{T})\}$
  $CP_{\mathtt{entry}}(5) = \{(x \to 10), (y \to \mathbf{T})\}$
  $CP_{\mathtt{exit}}(5) = \{(x \to 10), (y \to \mathbf{T}), (z \to 9)\}$

Non-distributivity

- In order to show that Constant Propagation Analysis is non distributive, just consider the transfer function f_ℓ corresponding to the statement $[y := x * x]^\ell$

consider two states $\sigma_1(x) = 1$ and $\sigma_2(x) = -1$
in theis case:

$$lub(\sigma_1,\sigma_2)(x) = T$$

and then

$$f_\ell\,(lub(\sigma_1,\sigma_2))(y) = T$$

whereas

$$f_\ell\,(\sigma_1)(y) = 1 = f_\ell\,(\sigma_2)(y)$$

Interprocedural analysis

Interprocedural Optimizations

- Until now, we have only considered optimizations "within a procedure"
- Extending these approaches outside of the procedural space involves similar techniques:
  - Performing interprocedural analysis
    - Control flow
    - Data flow
  - Using that information to perform interprocedural optimizations

What makes this difficult?

```
procedure joe(i,j,k)
   l ← 2 * k
   if (j = 100)
      then m ← 10 * j
      else m ← i
   call ralph(l,m,k)
   o ← m * 2
   q ← 2
   call ralph(o,q,k)
   write q, m, o, l
```

```
procedure main
   call joe( 10, 100, 1000)

procedure ralph(a,b,c)
   b ← a * c / 2000
```

Since j = 100 this always executes the then clause

and always m has the value 1000

What value is printed for q?
Did ralph() change it?

What happens at a procedure call?

Use worst case assumptions about side effects…

leads to imprecise <u>intra</u>procedural information

leads to explosion in <u>intra</u>procedural def-use chains

What makes this difficult?

```
procedure joe(i,j,k)
    l ← 2 * k
    if (j = 100)
        then m ← 10 * j
        else m ← i
```

```
procedure main
    call joe( 10, 100, 1000)

procedure ralph(a,b,c)
    b ← a * c / 2000
```

Since j = 100 this always executes the then clause

and always m has the value 1000

With perfect knowledge, the compiler could replace this with

write 2, 1000, 2000, 2000

and the rest is dead !

```
    write q, m, b, l
```

What value is printed for q?
Did ralph() change it?

What happens at a procedure call?

- Use worst case assumptions about side effects
- Leads to imprecise <u>intra</u>procedural information
- Leads to explosion in <u>intra</u>procedural def-use chains

The general pattern of Dataflow Analysis

$$GA_{\odot}(p) = \begin{cases} \iota \ \text{ if } p \in E \\ \oplus \{ \, GA_{\bullet}(q) \mid q \in F \, \} & \text{otherwise} \end{cases}$$

$$GA_{\bullet}(p) = f_p \, (\, GA_{\odot}(p) \,)$$

where :

E is the set of initial/final points of the control-flow diagram

ι specifies the initial values

F is the set of successor/predecessor points

$\oplus$ is the combination operator

f is the transfer function associated to node p

Procedure calls

- We can label a procedure call by:

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

dove:

a is an input parameter

z is an output parameter

ℓ_c is a label corresponding to the entrance into p

ℓ_r is a label corresponding to the exit out of p

Flow

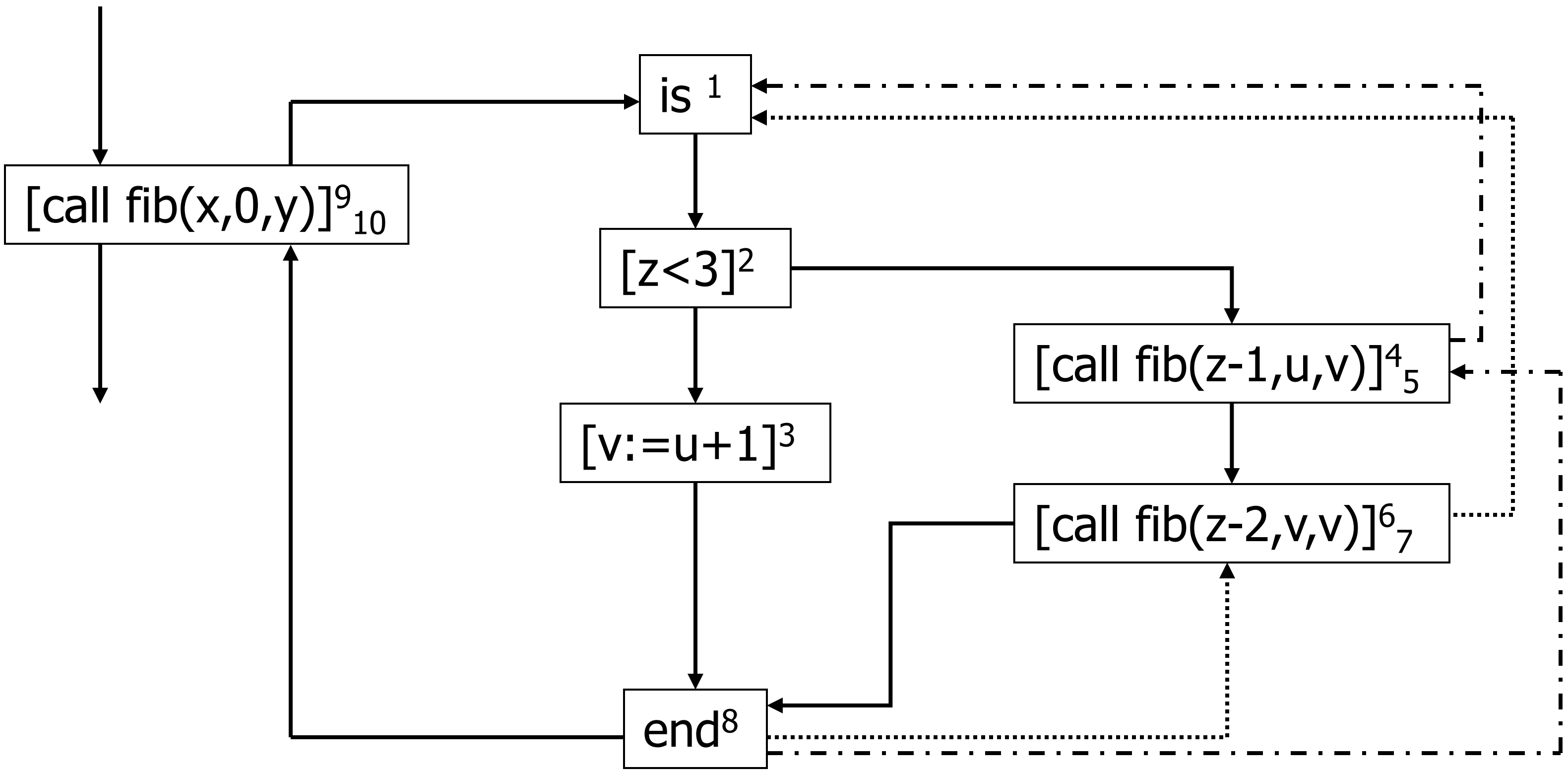
- In the intraprocedural analysis we considered a flow as a set of pairs (p,q) corresponding to an edge in the control flow graph
- We can now consider the call and a procedure declaration $[\text{call } p(a,z)]^{\ell_c}_{\ell_r}$ $\text{proc } p(\text{val } x, \text{res } y) \text{ is}^{\ell_{in}} S \text{ end}^{\ell_{out}};$
- In the interprocedural graph we should then consider also:
  - $(\ell_c; \ell_{in})$ the flow from the call ℓ_c, and the entry label ℓ_{in}
  - $(\ell_{out}; \ell_r)$ the flow from the exit label ℓ_{out} to the calling procedure ℓ_r.

Example

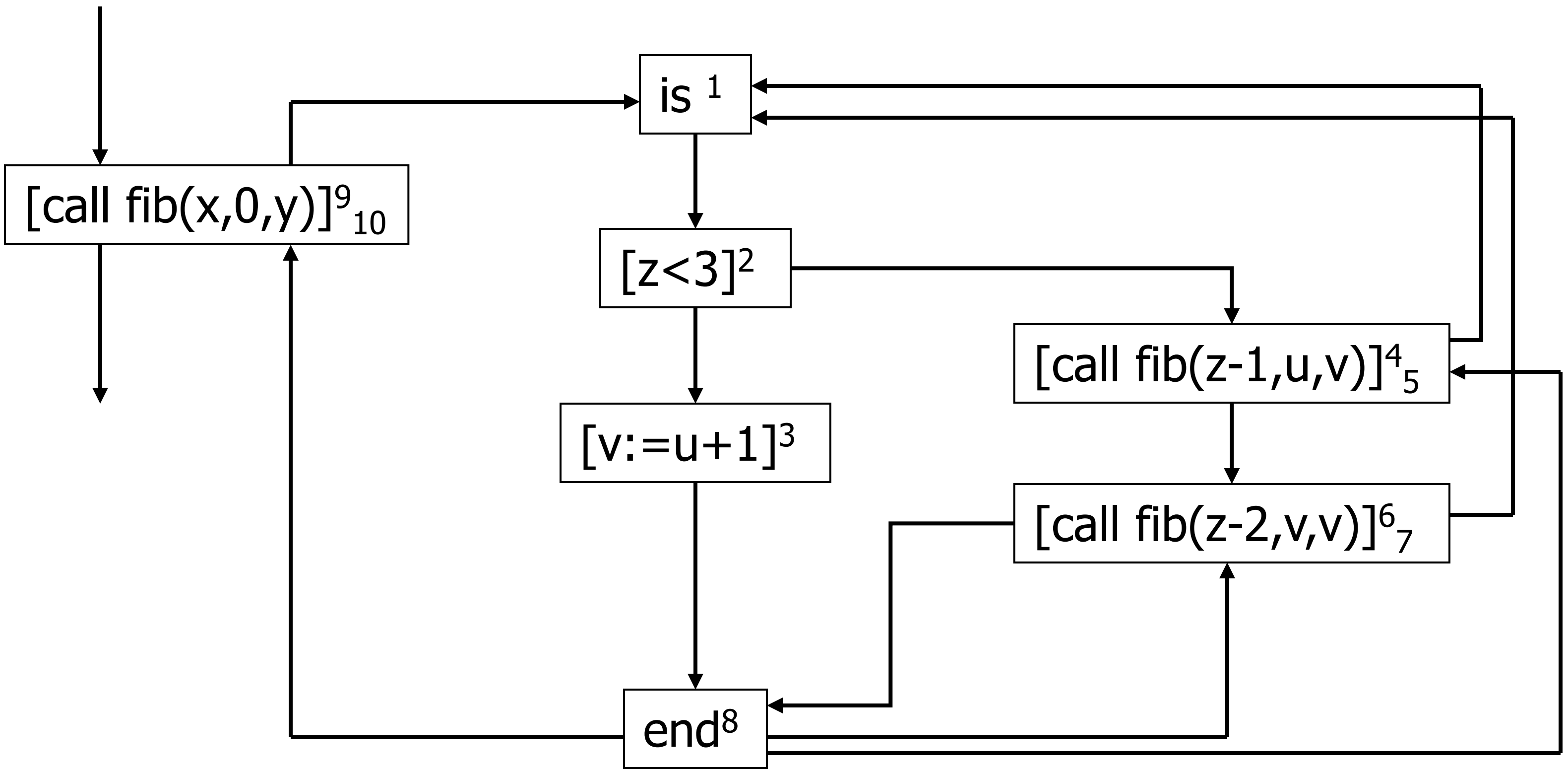
proc p(val x, res y) is$^{\ell_{in}}$ S end$^{\ell_{out}}$;

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

The flow graph



The resulting flattened flow graph



A naif approach

- We may simply extend the dataflow equations using the extended flow

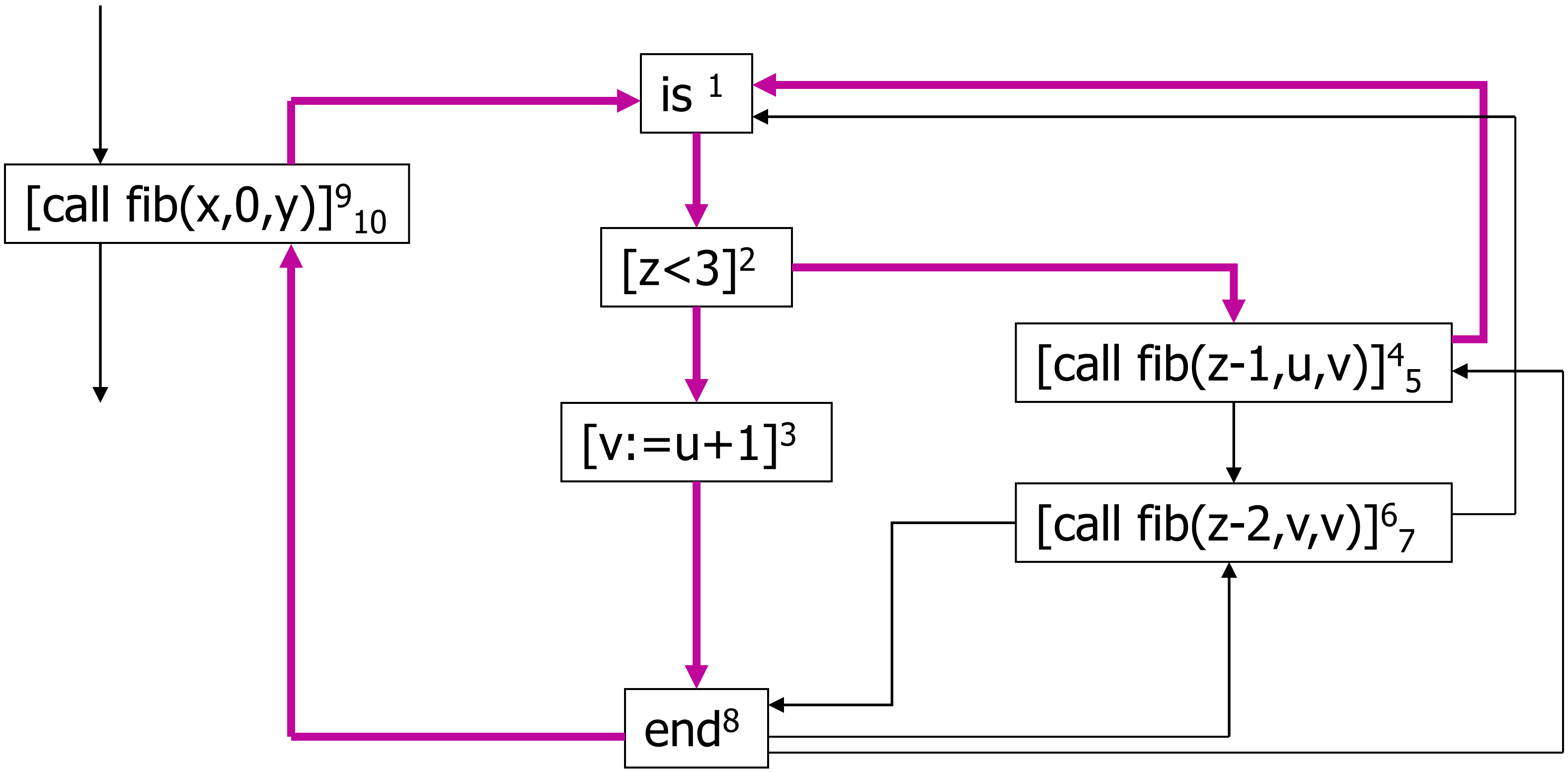
$$GA_{\odot}(\ell) = \begin{cases} \iota \ \text{if } \ell \in E \\ \text{lub}\,\{\, GA_{\bullet}(\ell') \mid (\ell', \ell) \in F \ \text{ or } \ (\ell'; \ell) \in F \,\} \quad \text{otherwise} \end{cases}$$

$$GA_{\bullet}(\ell) = f_{\ell}\,(\, GA_{\odot}(\ell)\,)$$

Correctness and Accuracy issues

- As we consider all possible paths $(\ell', \ell) \in F$ and $(\ell'; \ell) \in F$ the analysis is still correct
- However, the analysis also consider the path [9, 1, 2, 4, 1, 2, 3, 8, 10] that does not correspond to any actual computation of the program.
- This deeply affects the accuracy of the analysis

Spurious paths



- The path [9, 1, 2, 4, 1, 2, 3, 8, 10] never occurs in the actual computations

Inter-flow

We may define a notion of inter-flow:

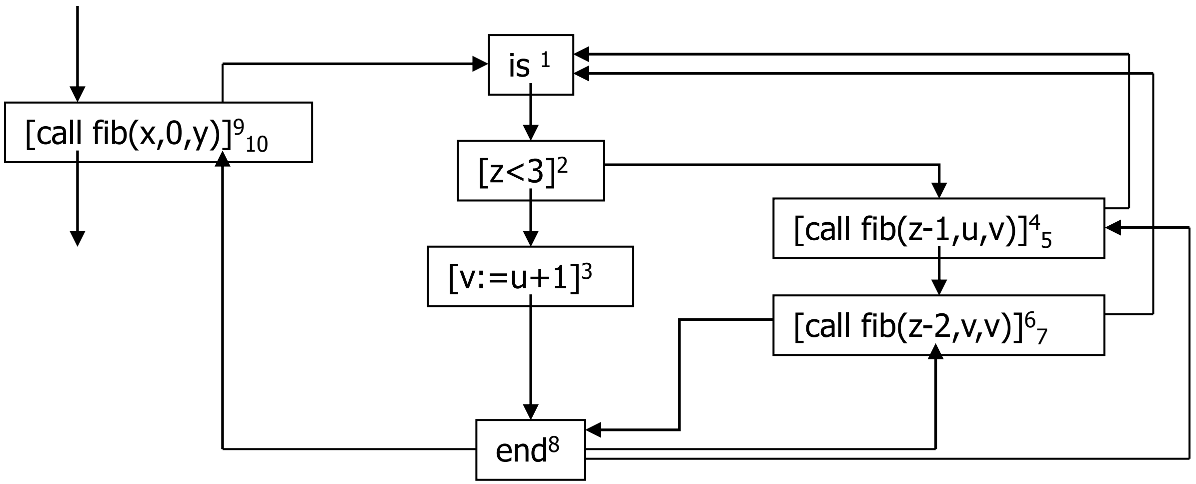
inter-flow = $\{(\ell_c, \ell_{in}, \ell_{out}, \ell_r)$ | the program contains both

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

and proc p(val x, res y) is$^{\ell_{in}}$ S end$^{\ell_{out}}$

}

Flow and inter-flow



- flow= {(1,2), (2,3), (2,4), (3,8), (4;1), (5,6), (6;1), (7,8), (8;5), (8;7), (8;10), (9;1)}
- Inter-flow= {(9,1,8,10), (4,1,8,5), (6,1,8,7)}

Extending the general framework

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

for all labels ℓ that do not appear as a first or last element of an inter-flow tuple

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

for all labels ℓ

Moreover, for each inter-flow tuple $(\ell_c, \ell_{in}, \ell_{out}, \ell_r)$ we introduce the equations:

$EA_{\bullet}(\ell_c) = f_{\ell_c}(\ EA_{\circ}(\ell_c)\ )$

$EA_{\bullet}(\ell_r) = f_{\ell_c,\ell_r}(\ EA_{\circ}(\ell_c), EA_{\circ}(\ell_r)\ )$