# Optimizing Compilers

## *Inter-Procedural Dataflow Analysis*

Jens Knoop, Markus Schordan, Jakob Zwirchmayr

Institut für Computersprachen
Technische Universität Wien

# Syntax

$$
\begin{array}{lcl}
P_\star & ::= & \text{begin } D_\star\ S_\star \text{ end} \\
D & ::= & D; D \mid \text{proc } p(\text{val } x;\ \text{res } y) \text{ is}^{\ell_n}\ S \text{ end}^{\ell_x} \\
S & ::= & \ldots \mid [\text{call } p(a,z)]^{\ell_c}_{\ell_r}
\end{array}
$$

Labeling scheme

- procedure declarations

  $\ell_n$: for entering the body

  $\ell_x$: for exiting the body

- procedure calls

  $\ell_c$: for the call

  $\ell_r$: for the return

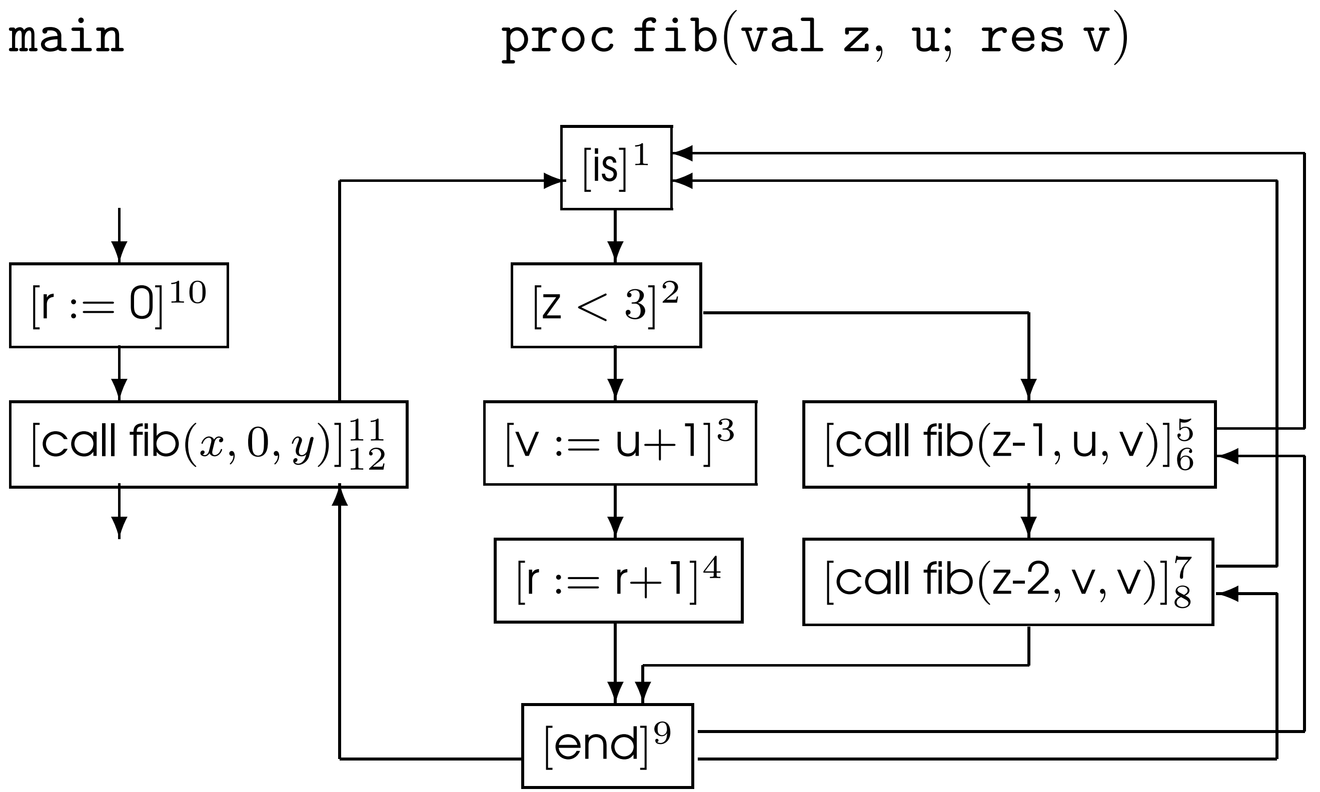
# Analysing Procedures

We consider procedures with call-by-value and call-by-result parameters.

Example:

```
begin
  proc fib(val z,u; res v) is
    if z<3 then
      (v:=u+1; r:=r+1)
    else (
      call fib (z-1,u,v);
      call fib (z-2,v,v)
      )
  end;
r:=0;
call fib(x,0,y)
end
```

# Example Flow Graph



# Flow Graph for Procedures

| | $[\text{call } p(a,z)]^{\ell_c}_{\ell_r}$ | $\text{proc } p(\text{val } x;\ \text{res } y) \text{ is}^{\ell_n}\ S \text{ end}^{\ell_x}$ |
|---|---|---|
| init | $\ell_c$ | $\ell_n$ |
| final | $\{\ell_r\}$ | $\{\ell_x\}$ |
| blocks | $\{[\text{call } p(a,z)]^{\ell_c}_{\ell_r}\}$ | $\{\text{is}^{\ell_n}\} \cup \text{blocks}(S) \cup \{\text{end}^{\ell_x}\}$ |
| labels | $\{\ell_c, \ell_r\}$ | $\{\ell_c, \ell_r\} \cup \text{labels}(S)$ |
| flow | $\{(\ell_c; \ell_n), (\ell_x; \ell_r)\}$ | $\{(\ell_n, \text{init}(S))\} \cup \text{flow}(S) \cup \{\ell, \ell_x) \mid \ell \in \text{final}(S))\}$ |

- $(\ell_c; \ell_n)$ is the flow corresponding to calling a procedure at $\ell_c$ and entering the procedure body at $\ell_n$ and
- $(\ell_x; \ell_r)$ is the flow corresponding to exiting a procedure body at $\ell_x$ and returning to the call at $\ell_r$.

# Naive Formulation

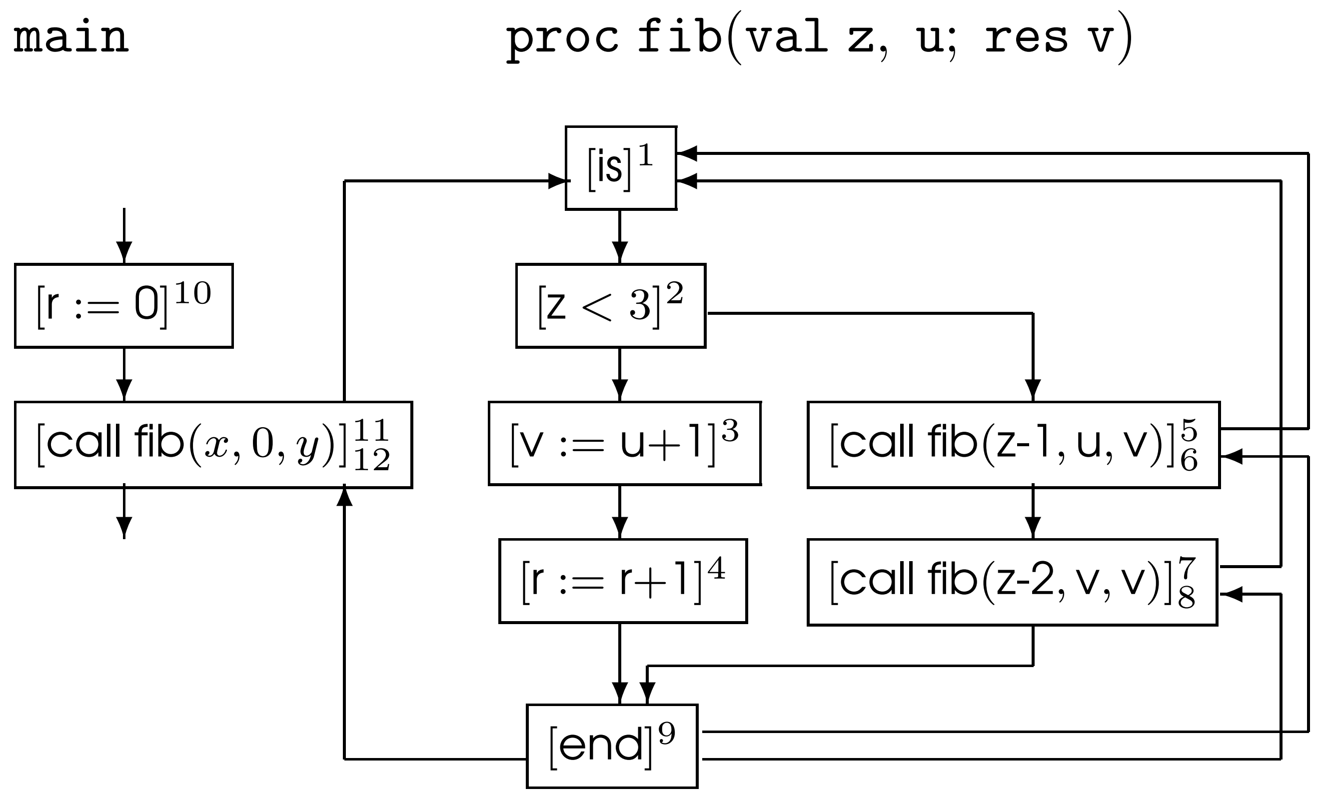
Treat the three kinds of flow, $(\ell_1, \ell_2)$, $(\ell_c; \ell_n)$, $(\ell_x; \ell_r)$ in the same way.

Equation system:

$$\begin{aligned} A_\circ(\ell) &= \bigsqcup\{A_\bullet(\ell') \mid (\ell', \ell) \in F \lor (\ell'; \ell) \in F\} \sqcup \iota_E^\ell \\ A_\bullet(\ell) &= f_\ell^A(A_\circ(\ell)) \end{aligned}$$

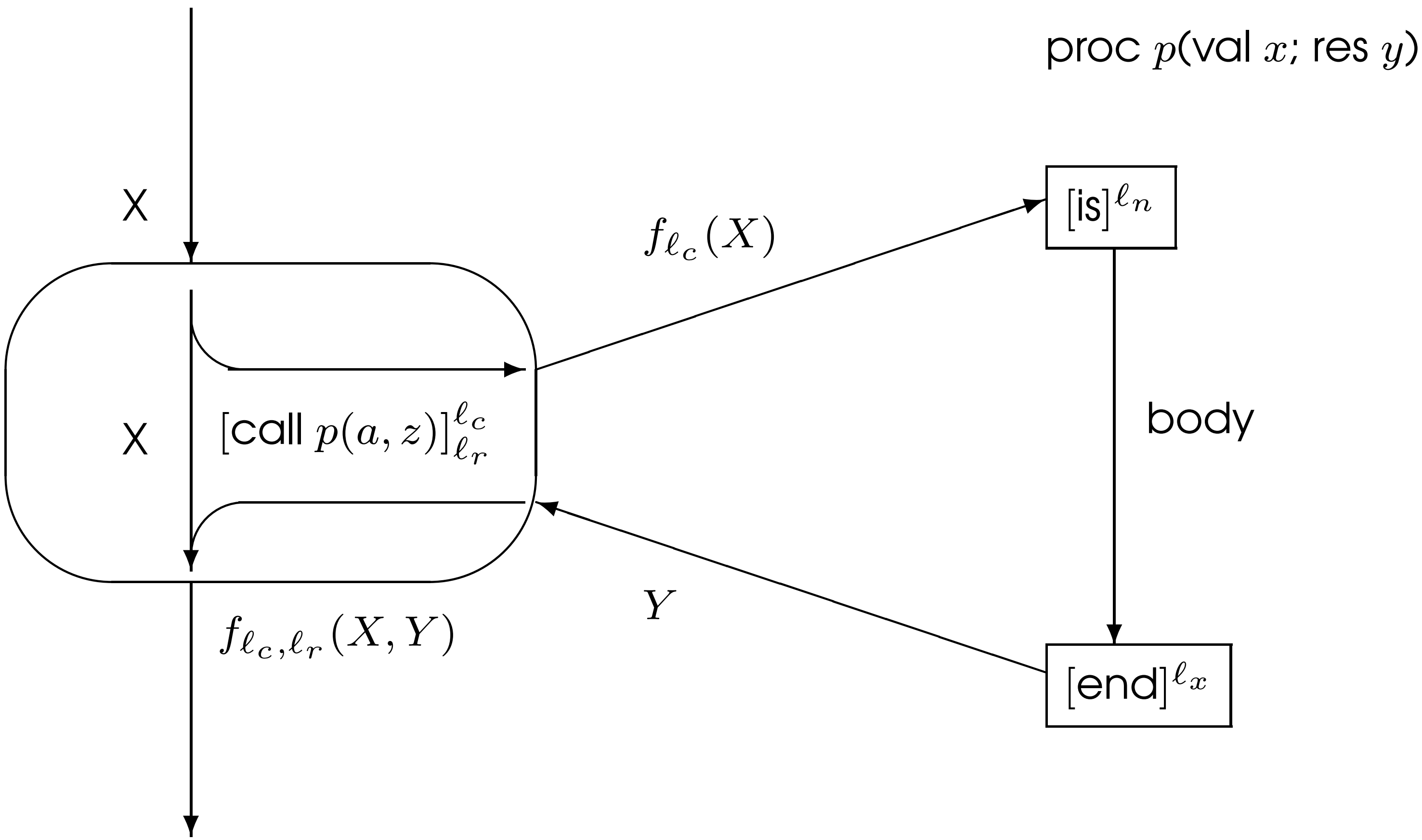
- both procedure calls $(\ell_c; \ell_n)$ and procedure returns $(\ell_x; \ell_r)$ are treated like "goto's".
- there is no mechanism for ensuring that information flowing along $(\ell_c; \ell_n)$ flows back along $(\ell_x; \ell_r)$ to the *same* call
- intuitively, the equation system considers a much too large set of "paths" through the program and hence will be grossly imprecise (although formally on the safe side)

# Matching Procedure Entries and Exits




We want to overcome the shortcoming of the naive formulation by restricting attention to paths that have the proper nesting of procedure calls and exits.

# General Formulation: Calls and Returns



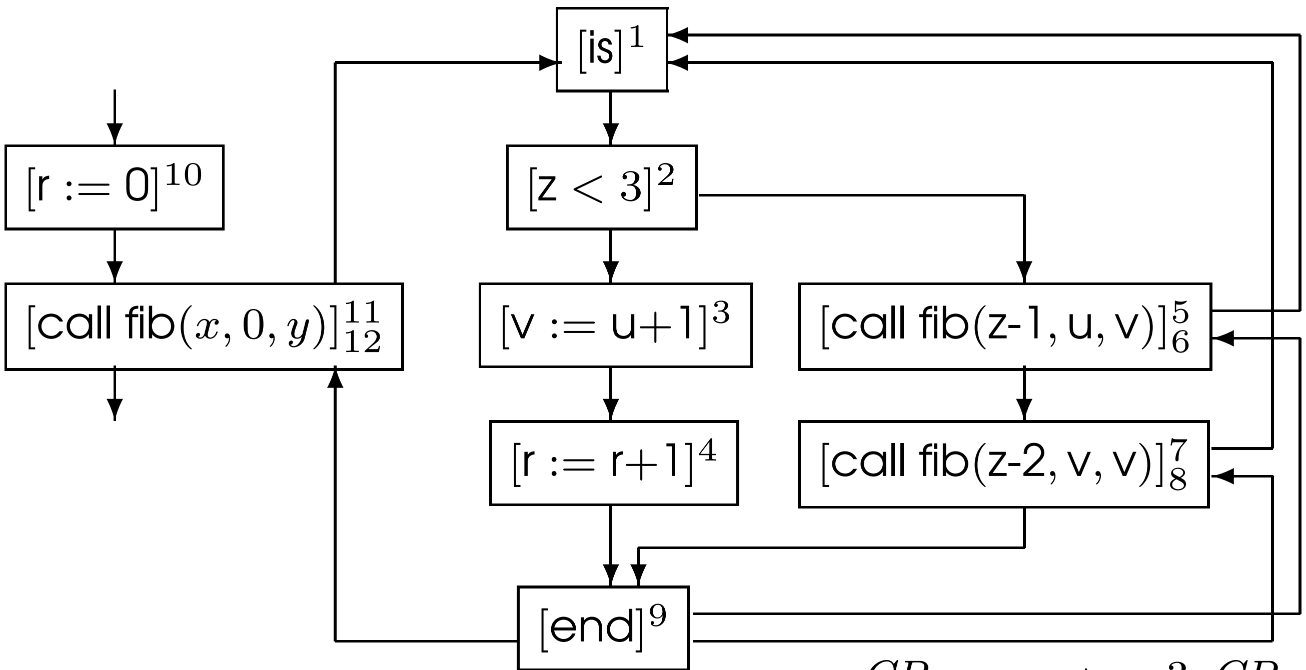
# “Meet” over Valid Paths (MVP)

A complete path from $\ell_1$ to $\ell_2$ in $P_\star$ has proper nesting of procedure entries and exits; and a procedure returns to the point where it was called:

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

**Definition:** $(\ell_c, \ell_n, \ell_r, \ell_x) \in$ interflow$_\star$ if $P_\star$ contains [call $p(a, z)]_{\ell_r}^{\ell_c}$ as well as proc $p($val $x;$ res $y)$ is$^{\ell_n}$ $S$ end$^{\ell_x}$

# Example




$$
\begin{aligned}
CP_{10,12} &\to 10,\ CP_{11,12} \\
CP_{11,12} &\to 11,\ CP_{1,9},\ CP_{12,12} \\
CP_{1,9} &\to 1,\ CP_{2,9} \\
CP_{2,9} &\to 2,\ CP_{3,9} \\
CP_{2,9} &\to 2,\ CP_{5,9}
\end{aligned}
$$

$$
\begin{aligned}
CP_{3,9} &\to 3,\ CP_{4,9} \\
CP_{4,9} &\to 4,\ CP_{9,9} \\
CP_{5,9} &\to 5,\ CP_{1,9},\ CP_{6,9} \\
CP_{6,9} &\to 6,\ CP_{7,9} \\
CP_{7,9} &\to 7,\ CP_{1,9},\ CP_{8,9} \\
CP_{8,9} &\to 8,\ CP_{9,9}
\end{aligned}
$$

$$
\begin{aligned}
CP_{12,12} &\to 12 \\
CP_{9,9} &\to 9
\end{aligned}
$$

Some valid paths: (10,11,1,2,3,4,9,12) and (10,11,1,2,5,1,2,3,4,9,6,7,1,2,3,4,9,8,9,12)

A non-valid path: (10,11,1,2,5,1,2,3,4,9,12)

# Valid Paths

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

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

# *MVP Solution*

$$MVP_\circ(\ell) = \bigsqcup \{ f_{\vec{\ell}}(\iota) | \vec{\ell} \in vpath_\circ(\ell) \}$$

$$MVP_\bullet(\ell) = \bigsqcup \{ f_{\vec{\ell}}(\iota) | \vec{\ell} \in vpath_\bullet(\ell) \}$$

where

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

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

The MVP solution may be undecidable for lattices satisfying the Ascending Chain Condition, just as was the case for the MOP solution.

# Making Context Explicit

- The MVP solution may be undecidable for lattices of finite height (as was the case for the MOP solution)
- We have to reconsider the MFP solution and avoid taking too many invalid paths into account
- Encode information about the paths taken into data flow properties themselves
- Introduce context information

# MFP Counterpart

Context sensitive analysis: add context information

- call strings:
  - an abstraction of the sequences of procedure calls that have been performed so far
  - example: the program point where the call was initiated
- assumption sets:
  - an abstraction of the states in which previous calls have been performed
  - example: an abstraction of the actual parameters of the call

Context insensitive analysis: take no context information into account.

# Call Strings as Context

- Encode the path taken
- Only record flows of the form $(\ell_c, \ell_n)$ corresponding to a procedure call
- we take as context $\Delta = \text{Lab}^*$ where the most recent label $\ell_c$ of a procedure call is at the right end
- Elements of $\Delta$ are called call strings
- The sequence of labels $\ell_c^1, \ell_c^2, \ldots, \ell_c^n$ is the call string leading to the current call which happened at $\ell_c^1$; the previous calls where at $\ell_c^2 \ldots \ell_c^n$. If $n = 0$ then no calls have been performed so far.

For the example program the following call strings are of interest:

$\Lambda, [11], [11,5], [11,7], [11,5,5], [11,5,7], [11,7,5], [11,7,7], ...$

# Abstracting Call Strings

**Problem:** call strings can be arbitrarily long (recursive calls)

**Solution:** truncate the call strings to have length of at most $k$ for some fixed number $k$

- $\Delta =$ Lab$^{\leq k}$
- $k = 0$: context insensitive analysis
  - $\Lambda$ (the call string is the empty string)
- $k = 1$: remember the last procedure call
  - $\Lambda, [11], [5], [7]$
- $k = 2$: remember the last two procedure calls
  - $\Lambda, [11], [11, 5], [11, 7], [5, 5], [5, 7], [7, 5], [7, 7]$

# References

- Material for this 5th lecture

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

- Book

  Flemming Nielson, Hanne Riis Nielson, Chris Hankin:

  Principles of Program Analysis.

  Springer, (450 pages, ISBN 3-540-65410-0), 1999.

  – Chapter 2 (Data Flow Analysis)