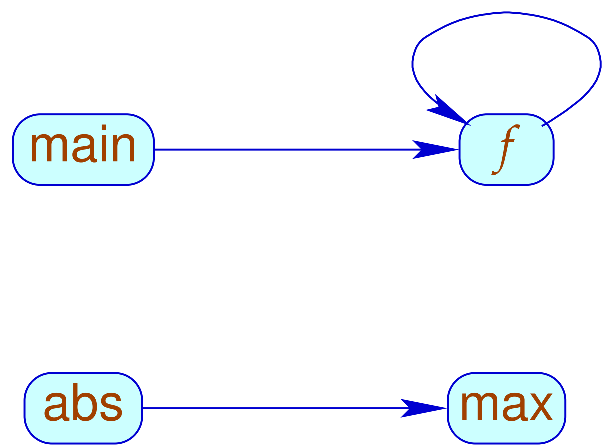
Detection of Recursion:

We construct the call-graph of the program.

In the Examples:



Call-Graph:

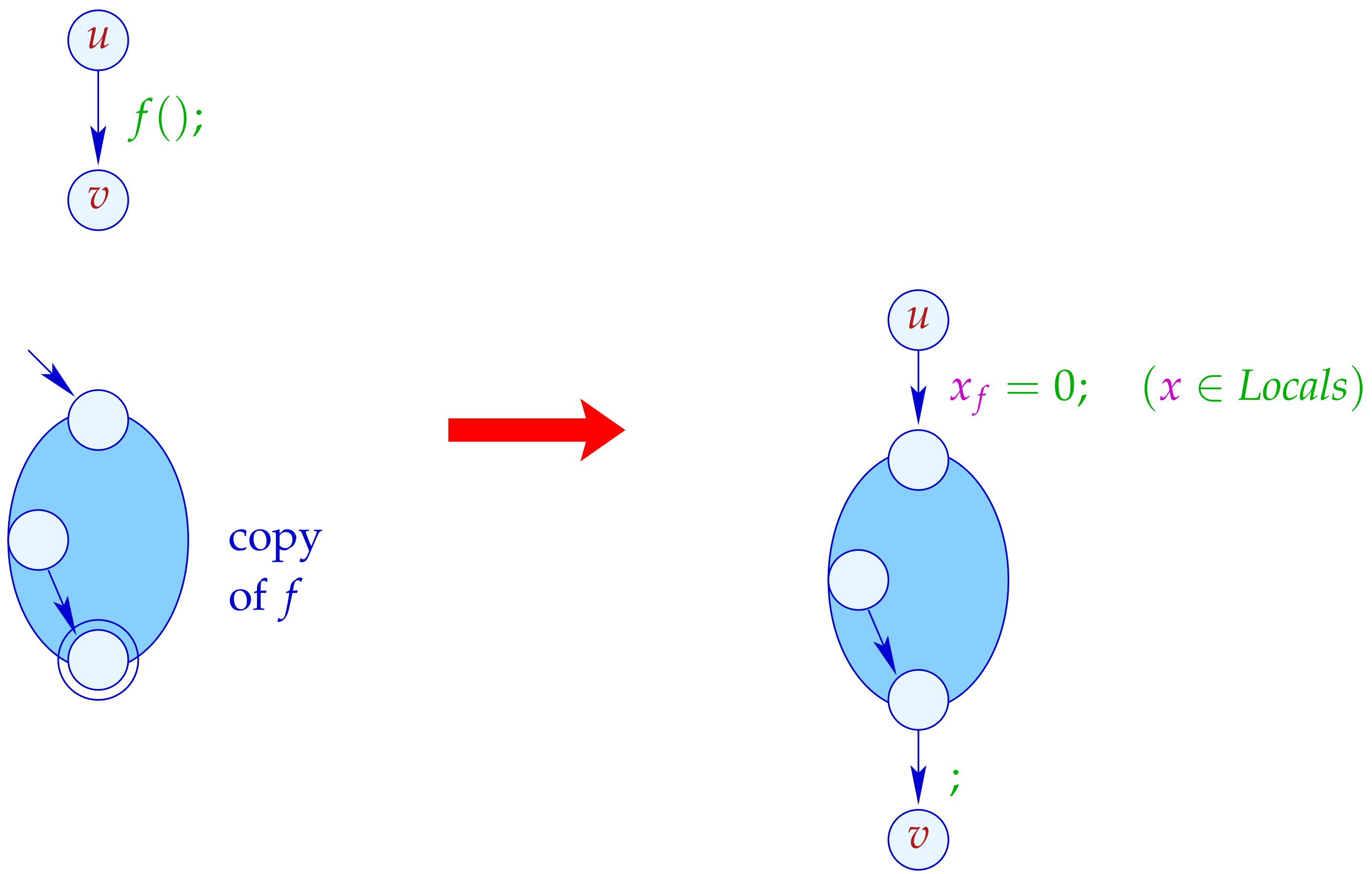
- The nodes are the procedures.
- An edge connexts g with h, whenever the body of g contains a call of h.

Strategies for Inlining:

- Just copy nur leaf-procedures, i.e., procedures without further calls :-)
- Copy all non-recursive procedures!

... here, we consider just leaf-procedures ;-)

Transformation 9:



Note:

- The Nop-edge can be eliminated if the *stop*-node of f has no out-going edges ...
- The x_f are the copies of the locals of the procedure f.
- According to our semantics of procedure calls, these must be initialized with 0 :-)

2. Idea: Elimination of Tail Recursion

$$
\begin{array}{l}
f\,()\ \{\quad \mathsf{int}\ \ b; \\
\qquad \mathsf{if}\ \ (a_2 \leq 1)\ \ \{\ \ \mathsf{ret} = a_1;\ \mathsf{goto}\ \_exit;\ \ \} \\
\qquad b = a_1 \cdot a_2; \\
\qquad a_2 = a_2 - 1; \\
\qquad a_1 = b; \\
\qquad f\,(); \\
\_exit\ \ : \\
\}
\end{array}
$$

After the procedure call, nothing in the body remains to be done.

$\Longrightarrow$ We may directly jump to the beginning :-)

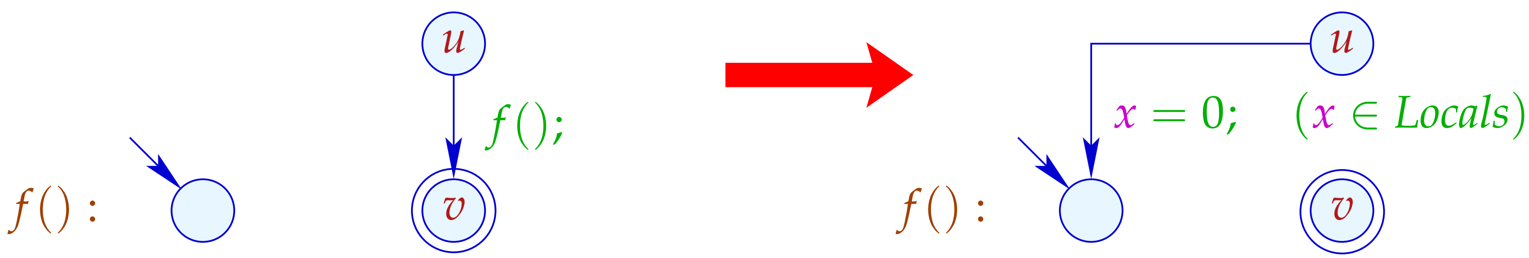
... after having reset the locals to 0.

... this yields in the Example:

$$
\begin{array}{ll}
f\,()\;\{ & \mathsf{int}\;\; b;\\
\_f: & \mathsf{if}\;\;(a_2 \leq 1)\;\;\{\;\mathsf{ret} = a_1;\; \mathsf{goto}\;\_exit;\;\;\}\\
 & b = a_1 \cdot a_2;\\
 & a_2 = a_2 - 1;\\
 & a_1 = b;\\
 & b = 0;\;\; \mathsf{goto}\;\;\_f;\\
\_exit: & \\
\} &
\end{array}
$$

// It works, since we have ruled out references to variables!

Transformation 11:



Warning:

- $\longrightarrow$ This optimization is crucial for programming languages without iteration constructs !!!
- $\longrightarrow$ Duplication of code is not necessary :-)
- $\longrightarrow$ No variable renaming is necessary :-)
- $\longrightarrow$ The optimization may also be profitable for non-recursive tail calls :-)
- $\longrightarrow$ The corresponding code may contain jumps from the body of one procedure into the body of another ???

Background 4: Interprocedural Analysis

So far, we can analyze each procedure separately.

→ The costs are moderate :-)

→ The methods also work in presence of separate compilation :-)

→ At procedure calls, we must assume the worst case :-(

→ Constant propagation only works for local constants :-((

Question:

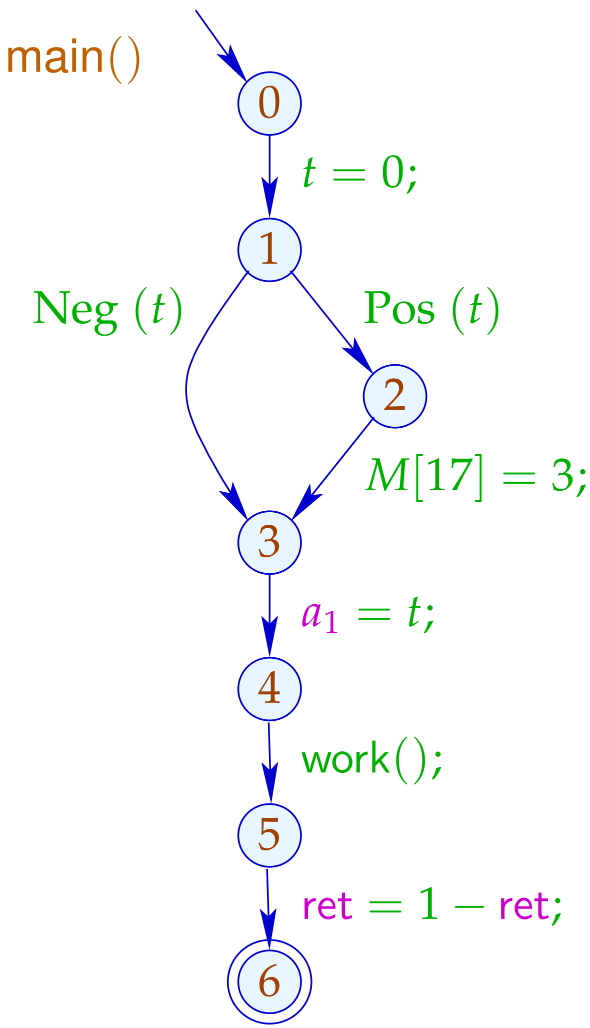
How can recursive programs be analyzed ???

Example: Constant Propagation

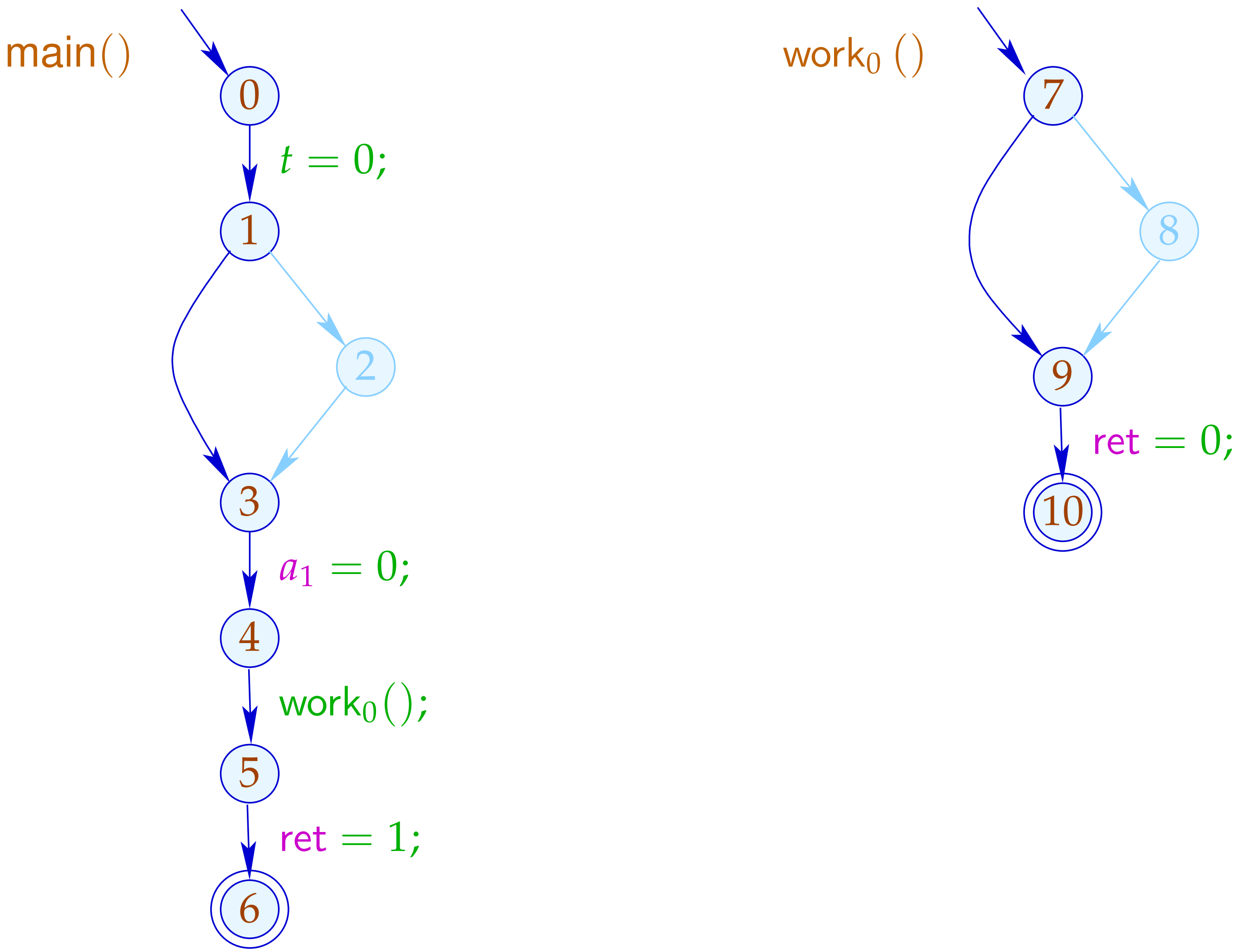
```
main() { int t;
        t = 0;
        if (t) M[17] = 3;
        a1 = t;
        work ();
        ret = 1 - ret;
}
```

```
work() {
        if (a1) work();
        ret = a1;
}
```

Example: Constant Propagation



Example: Constant Propagation



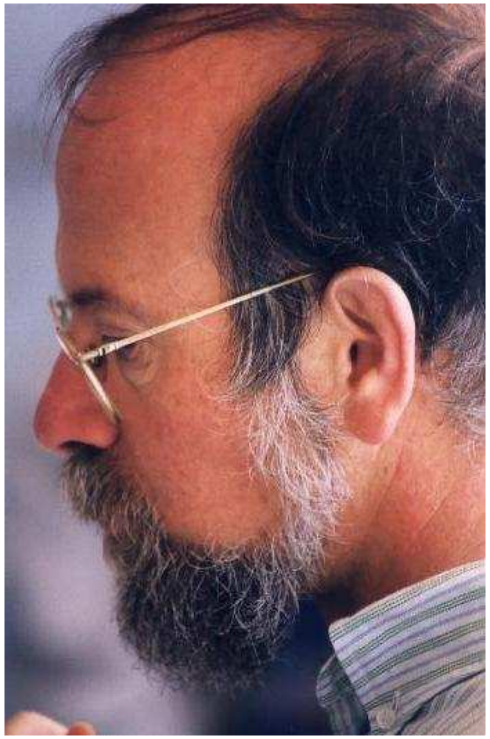
(1) Functional Approach:

Let $\mathbb{D}$ denote a complete lattice of (abstract) states.

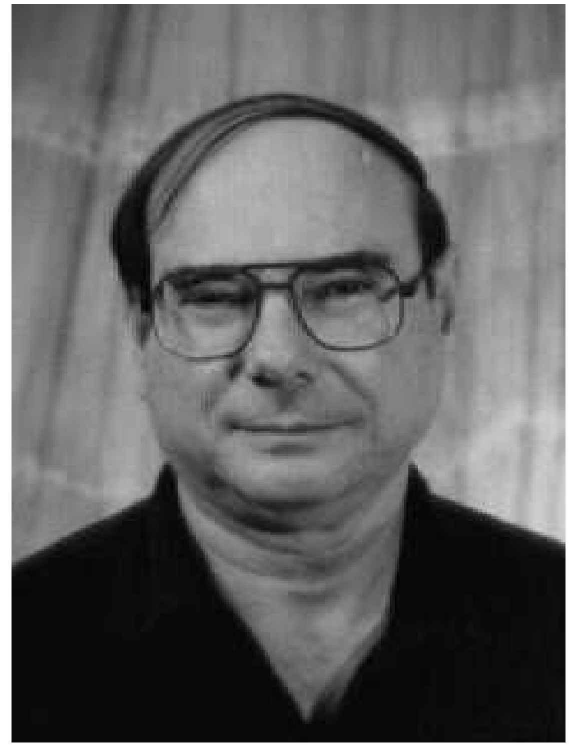
Idea:

Represent the effect of $f()$ by a function:

$$[\![f]\!]^\sharp \;:\; \mathbb{D} \to \mathbb{D}$$



Micha Sharir, Tel Aviv University



Amir Pnueli, Weizmann Institute

In order to determine the effect of a call edge $k = (u, f\,();, v)$ we require abstract functions:

$$\begin{array}{lll} \mathsf{enter}^\sharp & : & \mathbb{D} \longrightarrow \mathbb{D} \\ \mathsf{combine}^\sharp & : & \mathbb{D}^2 \longrightarrow \mathbb{D} \end{array}$$

Then we define:

$$[\![k]\!]^\sharp\, D \quad = \quad \mathsf{combine}^\sharp\, (D, [\![f]\!]^\sharp\, (\mathsf{enter}^\sharp\, D))$$

... for Constant Propagation:

$$\mathbb{D} = (\mathit{Vars} \to \mathbb{Z}^\top)_\perp$$

$$\mathsf{enter}^\sharp\ D = \begin{cases} \perp & \text{if} \quad D = \perp \\ D|_{\mathit{Globals}} \oplus \{x \mapsto 0 \mid x \in \mathit{Locals}\} & \text{otherwise} \end{cases}$$

$$\mathsf{combine}^\sharp\ (D_1, D_2) = \begin{cases} \perp & \text{if} \quad D_1 = \perp \vee D_2 = \perp \\ D_1|_{\mathit{Locals}} \oplus D_2|_{\mathit{Globals}} & \text{otherwise} \end{cases}$$

The effects $[\![f]\!]^\sharp$ then can be determined by a system of constraints over the complete lattice $\mathbb{D} \to \mathbb{D}$:

$$
\begin{array}{llll}
[\![v]\!]^\sharp & \sqsupseteq & \mathsf{Id} & v \quad \text{entry point} \\
[\![v]\!]^\sharp & \sqsupseteq & [\![k]\!]^\sharp \circ [\![u]\!]^\sharp & k = (u, \_, v) \quad \text{edge} \\
[\![f]\!]^\sharp & \sqsupseteq & [\![stop_f]\!]^\sharp & stop_f \quad \text{end point of} \quad f
\end{array}
$$

$[\![v]\!]^\sharp \;:\; \mathbb{D} \to \mathbb{D}$ describes the effect of all prefixes of computation forests w of a procedure which lead from the entry point to v

:-)

Problems:

- How can we represent functions $f : \mathbb{D} \to \mathbb{D}$???
- If $\#\mathbb{D} = \infty$, then $\mathbb{D} \to \mathbb{D}$ has infinite strictly increasing chains :-(

Simplification: Copy-Constants

$\longrightarrow$ Conditions are interpreted as $;$:-)

$\longrightarrow$ Only assignments $x = e;$ with $e \in \mathit{Vars} \cup \mathbb{Z}$ are treated exactly :-)

Observation:

$\longrightarrow$ The effects of assignments are:

$$[\![x = e;]\!]^\sharp\, D \;=\; \begin{cases} D \oplus \{x \mapsto c\} & \text{if} \quad e = c \in \mathbb{Z} \\ D \oplus \{x \mapsto (D\ y)\} & \text{if} \quad e = y \in \mathit{Vars} \\ D \oplus \{x \mapsto \top\} & \text{otherwise} \end{cases}$$

$\longrightarrow$ Let $\mathbb{V}$ denote the (finite !!!) set of constant right-hand sides. Then variables may only take values from $\mathbb{V}^\top$:-))

$\longrightarrow$ The occurring effects can be taken from

$$\mathbb{D}_f \to \mathbb{D}_f \qquad \text{with} \qquad \mathbb{D}_f = (\mathit{Vars} \to \mathbb{V}^\top)_\perp$$

$\longrightarrow$ The complete lattice is huge, but finite !!!

Improvement:

$\longrightarrow$ Not all functions from $\mathbb{D}_f \to \mathbb{D}_f$ will occur :-)

$\longrightarrow$ All occurring functions $\lambda D.\perp \;\neq\; M$ are of the form:

$$\begin{array}{lll} M & = & \{x \mapsto (b_x \sqcup \bigsqcup_{y \in I_x} y) \mid x \in Vars\} \qquad \text{where:} \\ M\; D & = & \{x \mapsto (b_x \sqcup \bigsqcup_{y \in I_x} D\; y) \mid x \in Vars\} \qquad \text{für} \quad D \neq \perp \end{array}$$

$\longrightarrow$ Let $\mathbb{M}$ denote the set of all these functions. Then for $M_1, M_2 \in \mathbb{M}$ $\quad (M_1 \neq \lambda\, D.\perp \neq M_2)$:

$$(M_1 \sqcup M_2)\; x \quad = \quad (M_1\; x) \sqcup (M_2\; x)$$

$\longrightarrow$ For $k = \#Vars$, $\mathbb{M}$ has height $\mathcal{O}(k^2)$:-)