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

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

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

Observation:

→ The effects of assignments are:

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

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

→ The occurring effects can be taken from

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

→ The complete lattice is huge, but finite !!!

Improvement:

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

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

$$M \quad = \quad \{x \mapsto (b_x \sqcup \bigsqcup_{y\in I_x} y) \mid x \in \mathit{Vars}\} \qquad \text{where:}$$

$$M\ D \quad = \quad \{x \mapsto (b_x \sqcup \bigsqcup_{y\in I_x} D\ y) \mid x \in \mathit{Vars}\} \qquad \text{für} \quad D \neq \perp$$

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

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

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

Improvement (Cont.):

→ Also, composition can be directly implemented:

$$
\begin{array}{rcll}
(M_1 \circ M_2)\ x & = & b' \sqcup \bigsqcup_{y \in I'} y & \quad \text{with} \\
b' & = & b \sqcup \bigsqcup_{z \in I} b_z & \\
I' & = & \bigcup_{z \in I} I_z & \quad \text{where} \\
M_1\ x & = & b \sqcup \bigsqcup_{y \in I} y & \\
M_2\ z & = & b_z \sqcup \bigsqcup_{y \in I_z} y &
\end{array}
$$

→ The effects of assignments then are:

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

... in the Example:

$$
\begin{array}{lll}
[\![t = 0;]\!]^\sharp & = & \{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, \boxed{t \mapsto 0}\} \\
[\![a_1 = t;]\!]^\sharp & = & \{\boxed{a_1 \mapsto t}, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}
\end{array}
$$

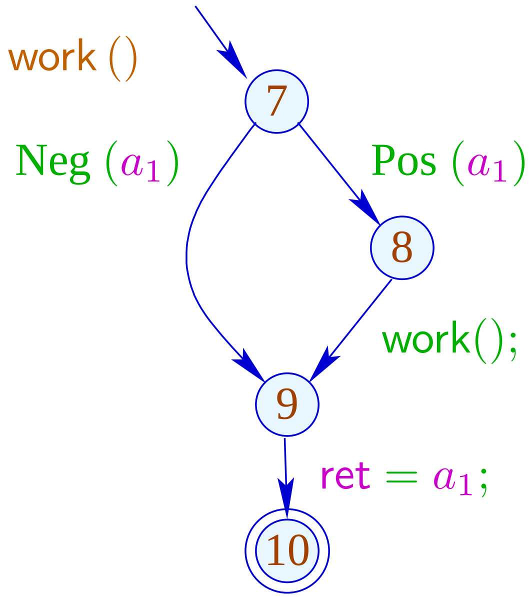
In order to implement the analysis, we additionally must construct the effect of a call $k = (\_, f\,();, \_)$ from the effect of a procedure f :

$$
\begin{array}{lll}
[\![k]\!]^\sharp & = & H\,([\![f]\!]^\sharp) \qquad\qquad \text{where:} \\
H\,(M) & = & \mathsf{Id}|_{Locals} \oplus (M \circ \mathsf{enter}^\sharp)|_{Globals} \\
\mathsf{enter}^\sharp\; x & = & \begin{cases} x & \text{if} \quad x \in Globals \\ 0 & \text{otherwise} \end{cases}
\end{array}
$$

... in the Example:

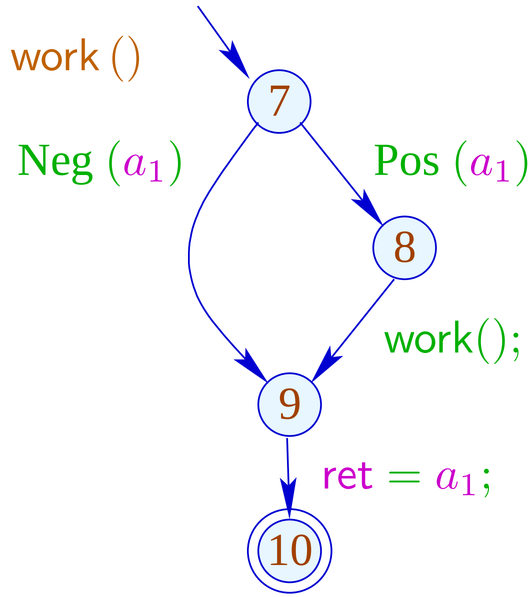
$$
\begin{array}{lrcl}
\text{If} & [\![\mathsf{work}]\!]^\sharp & = & \{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\} \\
\text{then} & H\,[\![\mathsf{work}]\!]^\sharp & = & \mathsf{Id}_{\{t\}} \oplus \{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1\} \\
 & & = & \{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\}
\end{array}
$$

Now we can perform fixpoint iteration :-)



	1
7	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$
9	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$
10	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\}$
8	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$

$$
\begin{aligned}
[\![(8, \ldots, 9)]\!]^\sharp \circ [\![8]\!]^\sharp &= \{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\} \circ \\
&\quad \{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\} \\
&= \{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\}
\end{aligned}
$$



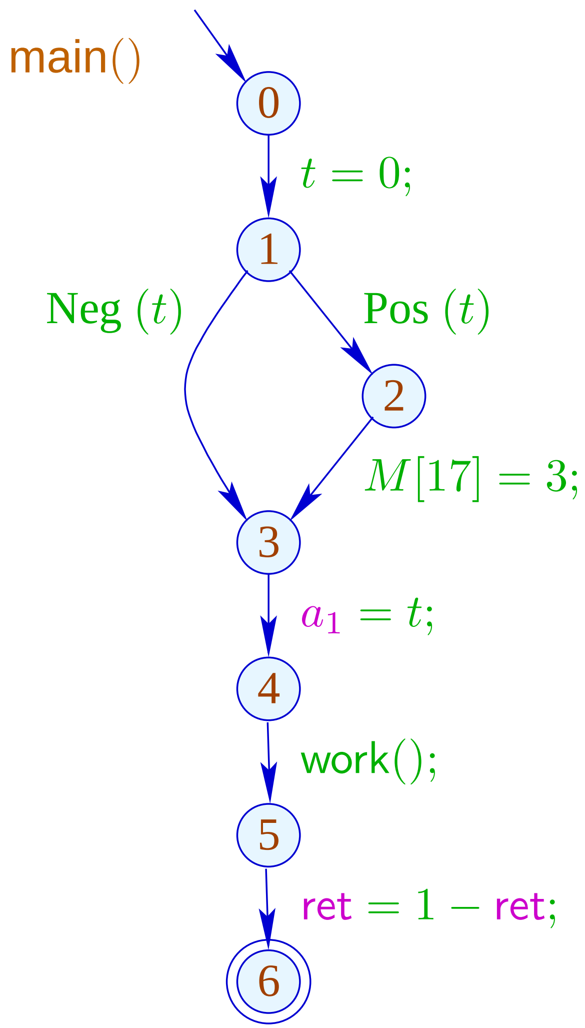
	2
7	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$
9	$\boxed{\{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1 \sqcup \mathsf{ret}, t \mapsto t\}}$
10	$\boxed{\{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\}}$
8	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$

$$\begin{aligned}
[\![(8,\ldots,9)]\!]^\sharp \circ [\![8]\!]^\sharp \quad &= \quad \{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\} \;\circ \\
&\qquad \{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\} \\
&= \quad \{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\}
\end{aligned}$$

If we know the effects of procedure calls, we can put up a constraint system for determining the abstract state when reaching a program point:

$$
\begin{array}{llll}
\mathcal{R}[\textsf{main}] & \sqsupseteq & \textsf{enter}^\sharp\; d_0 & \\
\mathcal{R}[f] & \sqsupseteq & \textsf{enter}^\sharp\; (\mathcal{R}[u]) & k = (u, f\,();, \_) \quad \text{call} \\
\mathcal{R}[v] & \sqsupseteq & \mathcal{R}[f] & v \quad \text{entry point of} \quad f \\
\mathcal{R}[v] & \sqsupseteq & [\![k]\!]^\sharp\; (\mathcal{R}[u]) & k = (u, \_, v) \quad \text{edge}
\end{array}
$$

... in the Example:



0	$\{a_1 \mapsto \top, \mathsf{ret} \mapsto \top, t \mapsto 0\}$
1	$\{a_1 \mapsto \top, \mathsf{ret} \mapsto \top, t \mapsto 0\}$
2	$\{a_1 \mapsto \top, \mathsf{ret} \mapsto \top, t \mapsto 0\}$
3	$\{a_1 \mapsto \top, \mathsf{ret} \mapsto \top, t \mapsto 0\}$
4	$\{a_1 \mapsto 0, \mathsf{ret} \mapsto \top, t \mapsto 0\}$
5	$\{a_1 \mapsto 0, \mathsf{ret} \mapsto 0, t \mapsto 0\}$
6	$\{a_1 \mapsto 0, \mathsf{ret} \mapsto \top, t \mapsto 0\}$

Discussion:

- At least copy-constants can be determined interprocedurally.
- For that, we had to ignore conditions and complex assignments :-(
- In the second phase, however, we could have been more precise :-)
- The extra abstractions were necessary for two reasons:

  (1) The set of occurring transformers $\mathbb{M} \subseteq \mathbb{D} \to \mathbb{D}$ must be finite;

  (2) The functions $M \in \mathbb{M}$ must be efficiently implementable :-)

- The second condition can, sometimes, be abandoned ...

Observation: Sharir/Pnueli, Cousot

→ Often, procedures are only called for few distinct abstract arguments.

→ Each procedure need only to be analyzed for these :-)

→ Put up a constraint system:

$$
\begin{array}{lll}
[\![v, a]\!]^\sharp & \sqsupseteq & a \qquad\qquad\qquad\qquad v \quad \text{entry point} \\
[\![v, a]\!]^\sharp & \sqsupseteq & \mathsf{combine}^\sharp\,([\![u, a]\!], [\![f, \mathsf{enter}^\sharp\,[\![u, a]\!]^\sharp]\!]^\sharp) \\
 & & \qquad\qquad\qquad (u, f\,();, v) \quad \text{call} \\
[\![v, a]\!]^\sharp & \sqsupseteq & [\![lab]\!]^\sharp\,[\![u, a]\!]^\sharp \quad k = (u, lab, v) \quad \text{edge} \\
[\![f, a]\!]^\sharp & \sqsupseteq & [\![stop_f, a]\!]^\sharp \qquad stop_f \quad \text{end point of} \quad f
\end{array}
$$

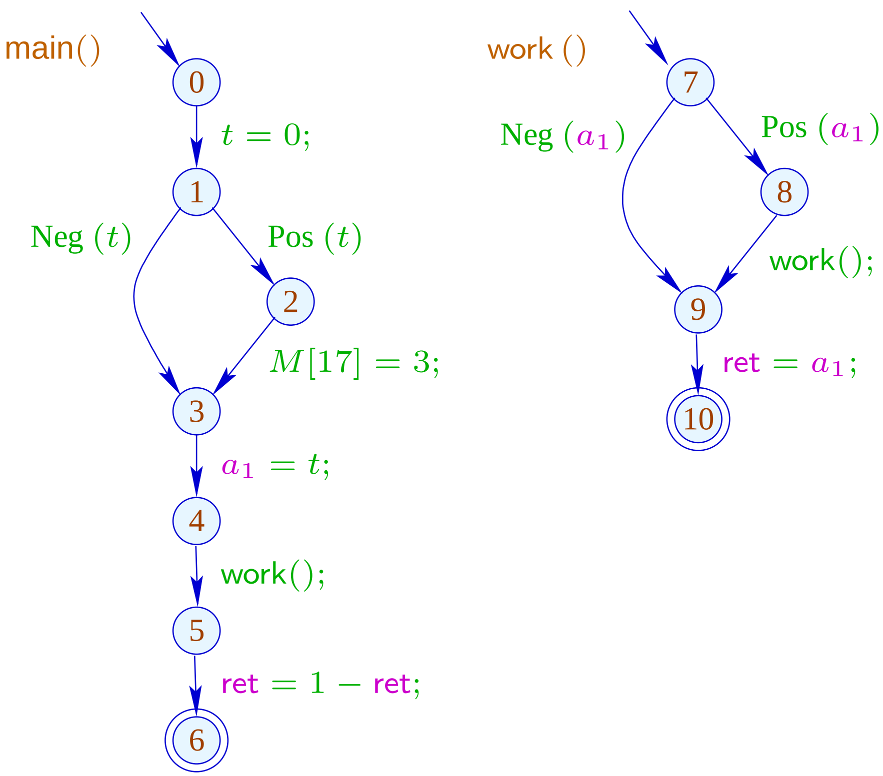
// $[\![v, a]\!]^\sharp$ == value for the argument a .

Discussion:

- This constraint system may be huge :-(
- We do not want to solve it completely!!!
- It is sufficient to compute the correct values for all calls which occur, i.e., which are necessary to determine the value $[\![\mathsf{main}(), a_0]\!]^\sharp$ $\Longrightarrow$ We apply our local fixpoint algorithm :-))
- The fixpoint algo provides us also with the set of actual parameters $a \in \mathbb{D}$ for which procedures are (possibly) called and all abstract values at their program points for each of these calls :-)

... in the Example:

Let us try a full constant propagation ...



	a_1	ret	a_1	ret
0	$\top$	$\top$	$\top$	$\top$
1	$\top$	$\top$	$\top$	$\top$
2	$\top$	$\top$	$\perp$	
3	$\top$	$\top$	$\top$	$\top$
4	$\top$	$\top$	0	$\top$
7	0	$\top$	0	$\top$
8	0	$\top$	$\perp$	
9	0	$\top$	0	$\top$
10	0	$\top$	0	0
5	$\top$	$\top$	0	0
main()	$\top$	$\top$	0	1

Discussion:

- In the Example, the analysis terminates quickly :-)
- If $\mathbb{D}$ has finite height, the analysis terminates if each procedure is only analyzed for finitely many arguments :-))
- Analogous analysis algorithms have proved very effective for the analysis of Prolog :-)
- Together with a points-to analysis and propagation of negative constant information, this algorithm is the heart of a very successful race analyzer for C with Posix threads :-)

(2) The Call-String Approach:

Idea:

$\rightarrow$ Compute the set of all reachable call stacks!

$\rightarrow$ In general, this is infinite :-(

$\rightarrow$ Only treat stacks up to a fixed depth d precisely! From longer stacks, we only keep the upper prefix of length d :-)

$\rightarrow$ Important special case: $d = 0$.

$\Longrightarrow$ Just track the current stack frame ...