Script generated by TTT

Title: Seidl: Programmoptimierung (10.12.2012)

Date: Mon Dec 10 15:05:26 CET 2012

Duration: 88:13 min

Pages: 44

2. Idea: Elimination of Tail Recursion

$f\,()\ \{\ \ \texttt{int}\ \ b;$

$\quad \texttt{if}\ \ (a_2 \leq 1)\ \ \{\ \ \texttt{ret} = a_1;\ \ \texttt{goto}\ \_exit;\ \ \}$

$\quad b = a_1 \cdot a_2;$

$\quad a_2 = a_2 - 1;$

$\quad a_1 = b;$

$\quad f\,();$

$\_exit\ :$

$\}$

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:

$f\,()\ \{\ \ \texttt{int}\ \ b = 0;$

$\_f\ :\quad \texttt{if}\ \ (a_2 \leq 1)\ \ \{\ \ \texttt{ret} = a_1;\ \ \texttt{goto}\ \_exit;\ \ \}$

$\quad b = a_1 \cdot a_2;$

$\quad a_2 = a_2 - 1;$

$\quad a_1 = b;$

$\quad \cancel{b = 0;}\ \ \texttt{goto}\ \ \_f;$

$\_exit\ :$

$\}$

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

Transformation 11:

Warning:

- This optimization is crucial for programming languages without iteration constructs !!!
- Duplication of code is not necessary :-)
- No variable renaming is necessary :-)
- The optimization may also be profitable for non-recursive tail calls :-)
- 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:

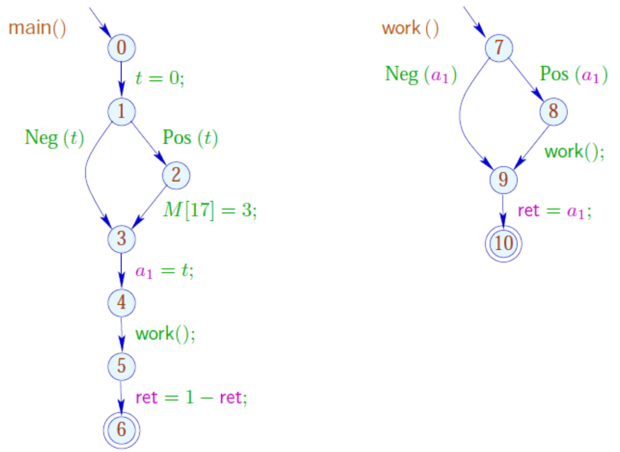
How can recursive programs be analyzed ???

Example: Constant Propagation

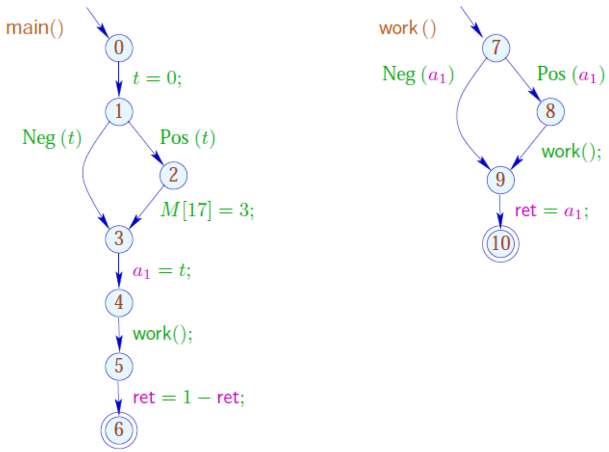
```
main() { int t;
    t = 0;
    if (t) M[17] = 3;
    a1 = t;
    work ();
    ret = 1 − ret;
}

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

0

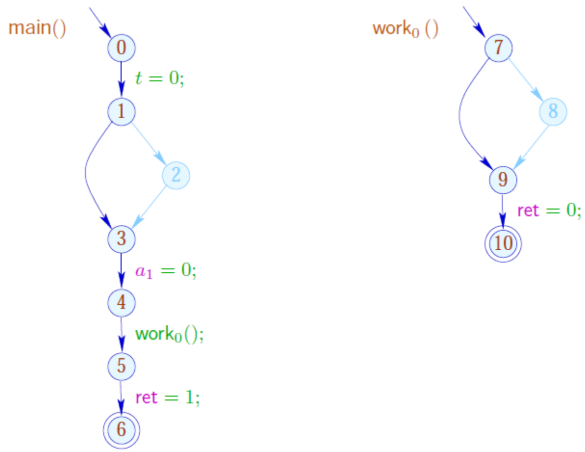
Example: Constant Propagation



Example: Constant Propagation



Example: Constant Propagation



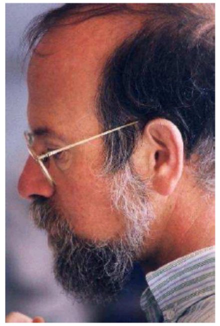
(1) Functional Approach:

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

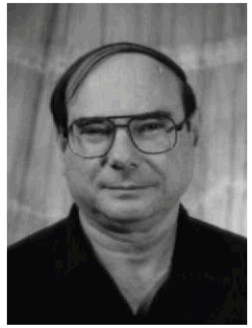
Idea:

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:

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

Then we define:

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

(1) Functional Approach:

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

Idea:

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

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

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

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

Then we define:

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

... for Constant Propagation:

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

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

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

↳ $D_1 \oplus D_2|_{Globals}$

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

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

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

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

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

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

→ Conditions are interpreted as ; :-)

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

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

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

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

→ Conditions are interpreted as ; :-)

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

$[\![x = y + 1]\!]^\# D = D \oplus \{x \mapsto \top\}$

$[\![x = y]\!]^\# D = D \oplus \{x \mapsto D\, y\}$

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

→ Conditions are interpreted as ; :-)

→ 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 !!!

Observation:

$$[\![x = y+1]\!]^\# D = D \oplus \{x \mapsto \top\}$$

→ 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 $\mathbb{V}^\top$:-))

$$[\![x = y]\!]^\# D = D \oplus \{x \mapsto D\, y\}$$

→ The occurring effects can be taken from

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

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

Observation:

$$\#\mathbb{D}_f = (c+1)^k + 1$$

→ 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

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

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

$$\#\mathbb{D}_f^{\#\mathbb{D}_f}$$

Improvement:

$$x \mapsto 5 \sqcup y_1 \sqcup y_2$$

$$x \mapsto 5 \sqcup x$$

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

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

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

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

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

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

$$[\![x = y+1]\!]^\# D = D \oplus \{x \mapsto \top\}$$

$$[\![x = y]\!]^\# D = D \oplus \{x \mapsto D\, y\}$$

Improvement:

$M: x \mapsto (b, I_i)$

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

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

$b \in \mathbb{V}^\top$

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

$I \subseteq Vars$

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

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

$M_1 \circ M_2$

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

Improvement (Cont.):

→ Also, composition can be directly implemented:

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

→ The effects of assignments then are:

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

Improvement:

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

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

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

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

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

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

$\mathsf{Id} = \{x \mapsto (\bot, \{x\}) \mid x \in Vars\}$

Improvement (Cont.):

I

→ Also, composition can be directly implemented:

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

→ The effects of assignments then are:

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

... in the Example:

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

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

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

... in the Example:

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

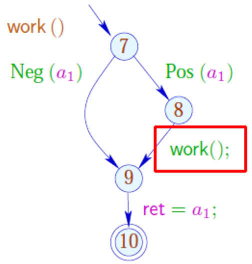
In order to implement the analysis, we additionally must construct the effect of a call $k = (\_, f\,();, \_)$ from the effect of a procedure f :

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

... in the Example:

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

Now we can perform fixpoint iteration :-)



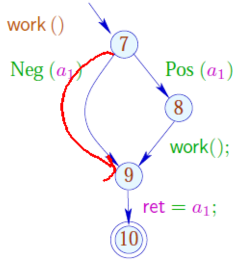
	1
7	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$
9	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$
10	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\}$
8	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$

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

... in the Example:

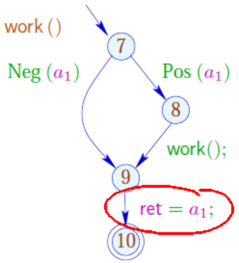
$$\begin{array}{lrcl} \text{If} & [\![\mathsf{work}]\!]^\sharp & = & \{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\} \\ \text{then} & H\,[\![\mathsf{work}]\!]^\sharp & = & \mathsf{Id}_{\{t\}} \oplus \{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1\} \\ & & = & \{a_1 \mapsto a_1, \mathsf{ret} \mapsto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

Now we can perform fixpoint iteration :-)



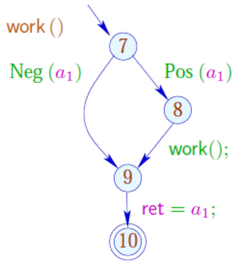
	1
7	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$
9	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$
10	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\}$
8	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$

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



	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	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$

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



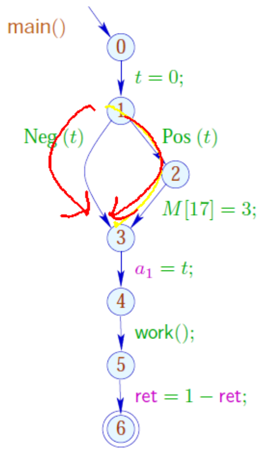
	2
7	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$
9	$\boxed{\{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1 \sqcup \mathsf{ret}, t \mapsto t\}}$
10	$\boxed{\{a_1 \mapsto a_1, \mathsf{ret} \mapsto a_1, t \mapsto t\}}$
8	$\{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto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

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

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:

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

... 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:

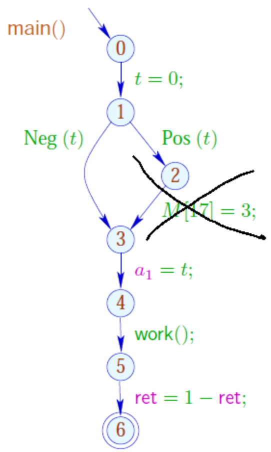
- At least copy-constants can be determined interprocedurally.
- For that, we had to ignore conditions and complex assignments :-(
- In the second phase, however, we could have been more precise :-)
- The extra abstractions were necessary for two reasons:

  (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 ...

... 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 ...