Script **generated by TTT**

Title: Seidl: Programmoptimierung (17.12.2015)

Date: Thu Dec 17 08:34:31 CET 2015

Duration: 88:26 min

Pages: 39

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

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

$[v]^\sharp : \mathbb{D} \to \mathbb{D}$ 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 $e \in Vars \cup \mathbb{Z}$ are treated exactly.

Observation

→ The effects of assignments are:

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

→ 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

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

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

height: $(k+1)^n \cdot 3 \cdot n$

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:

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

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

Improvement (Cont.)

→ Also, composition can be directly implemented:

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

→ The effects of assignments then are:

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

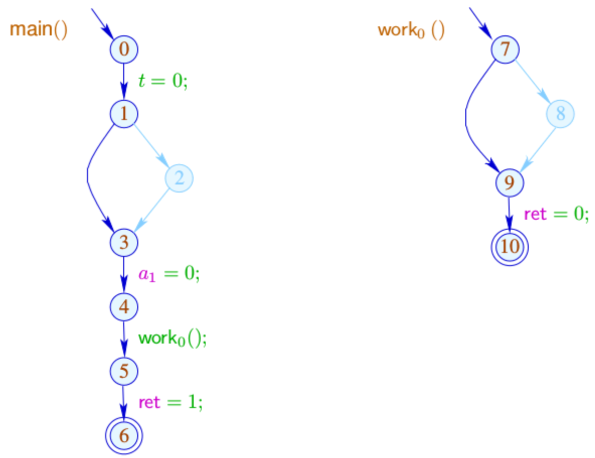
... in the Example:

$$[\![t = 0;]\!]^\sharp \quad = \quad \{a_1 \mapsto a_1, \mathsf{ret} \mapsto \mathsf{ret}, \boxed{t \mapsto 0}\}$$
$$[\![a_1 = t;]\!]^\sharp \quad = \quad \{\boxed{a_1 \mapsto t}, \mathsf{ret} \mapsto \mathsf{ret}, t \mapsto t\}$$

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

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

Example: Constant Propagation



... in the Example:

$$[\![t = 0;]\!]^\sharp \;=\; \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, \boxed{t \mapsto 0}\}$$
$$[\![a_1 = t;]\!]^\sharp \;=\; \{\boxed{a_1 \mapsto t}, \text{ret} \mapsto \text{ret}, t \mapsto t\}$$

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

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

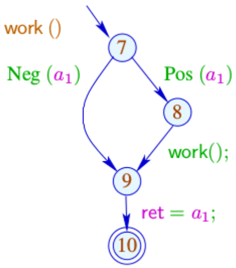
... in the Example:

If $\quad [\text{work}]^\sharp \;=\; \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}$

then $\quad H\,[\text{work}]^\sharp \;=\; \text{Id}_{\{t\}} \oplus \{a_1 \mapsto a_1, \text{ret} \mapsto a_1\}$
$$=\; \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}$$

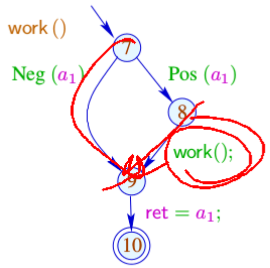
Now we can perform fixpoint iteration ...

$[\![\text{work}]\!]^\sharp \circ \text{enter}^\sharp = \{a_1 \mapsto 0, \text{ret} \mapsto a_1, t \mapsto 0\}$



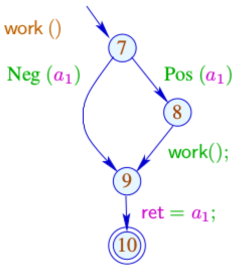
	1
7	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$
9	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$
10	$\{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}$
8	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$

$$[\![(8,\ldots,9)]\!]^\sharp \circ [\![8]\!]^\sharp \;=\; \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\} \;\circ$$
$$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$$
$$=\; \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}$$



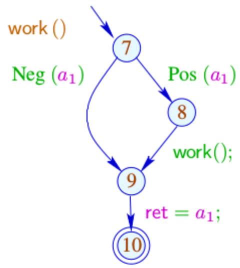
	2
7	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$
9	$\{a_1 \mapsto a_1, \text{ret} \mapsto a_1 \sqcup \text{ret}, t \mapsto t\}$
10	$\{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}$
8	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$

$$[\![(8,\ldots,9)]\!]^\sharp \circ [\![8]\!]^\sharp \;=\; \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\} \;\circ$$
$$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$$
$$=\; \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}$$



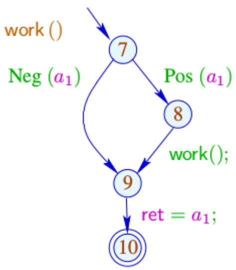
	1
7	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$
9	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$
10	$\{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}$
8	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$

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



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

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



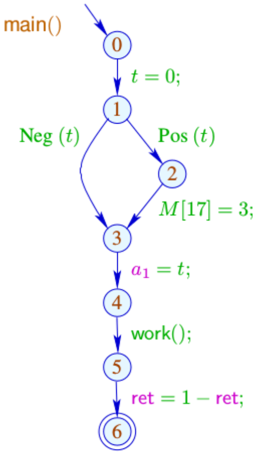
	2
7	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$
9	$\{a_1 \mapsto a_1, \text{ret} \mapsto a_1 \sqcup \text{ret}, t \mapsto t\}$
10	$\{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}$
8	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$

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

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:

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

... in the Example:

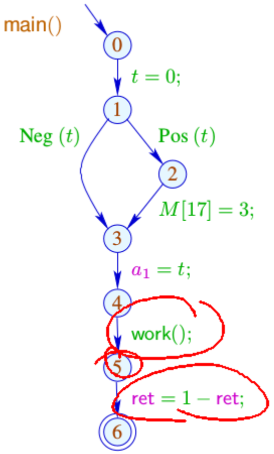


0	$\{a_1 \mapsto \top, \text{ret} \mapsto \top, t \mapsto 0\}$
1	$\{a_1 \mapsto \top, \text{ret} \mapsto \top, t \mapsto 0\}$
2	$\{a_1 \mapsto \top, \text{ret} \mapsto \top, t \mapsto 0\}$
3	$\{a_1 \mapsto \top, \text{ret} \mapsto \top, t \mapsto 0\}$
4	$\{a_1 \mapsto 0, \text{ret} \mapsto \top, t \mapsto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

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

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:

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

// $[v,a]^\sharp$ == 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 $[\text{main}(), a_0]^\sharp$ $\Longrightarrow$ 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.

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:

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

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

570

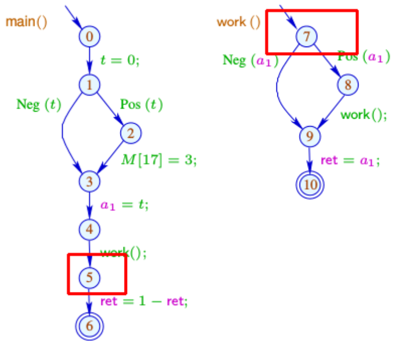
Discussion

- 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 $[\text{main}(), a_0]^\sharp$ $\Longrightarrow$ 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.

571

... 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	$\top$	$\top$	$\bot$	
3	$\top$	$\top$	$\top$	$\top$
4	$\top$	$\top$	0	$\top$
7	0	$\top$	0	$\top$
8	0	$\top$	$\bot$	
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

→ Compute the set of all reachable call stacks!

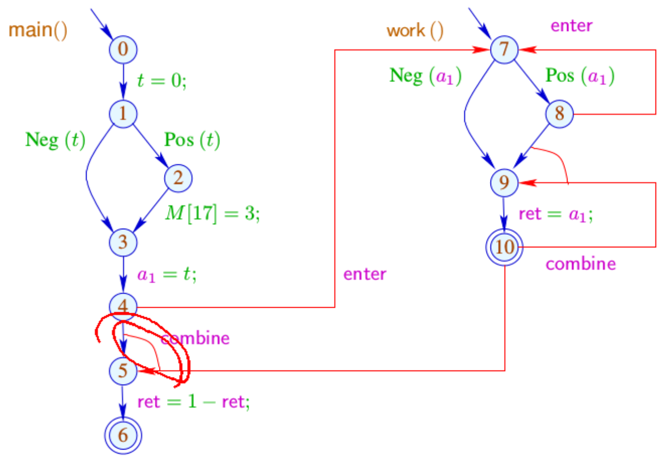
→ In general, this is infinite.

→ Only treat stacks up to a fixed depth d precisely! From longer stacks, we only keep the upper prefix of length d.

→ Important special case: $d = 0$.

⟹ Just track the current stack frame ...

... in the Example:



The conditions for $5, 7, 10$, e.g., are:

$$\mathcal{R}[5] \sqsupseteq \text{combine}^\sharp(\mathcal{R}[4], \mathcal{R}[10])$$
$$\mathcal{R}[7] \sqsupseteq \text{enter}^\sharp(\mathcal{R}[4])$$
$$\mathcal{R}[7] \sqsupseteq \text{enter}^\sharp(\mathcal{R}[8])$$
$$\mathcal{R}[9] \sqsupseteq \text{combine}^\sharp(\mathcal{R}[8], \mathcal{R}[10])$$

Caveat

The resulting super-graph contains obviously impossible paths ...

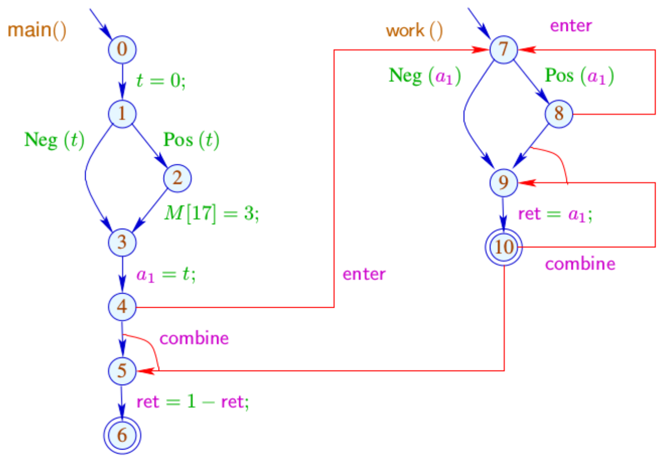
The conditions for $5, 7, 10$, e.g., are:

$$\mathcal{R}[5] \sqsupseteq \text{combine}^\sharp(\mathcal{R}[4], \mathcal{R}[10])$$
$$\mathcal{R}[7] \sqsupseteq \text{enter}^\sharp(\mathcal{R}[4])$$
$$\mathcal{R}[7] \sqsupseteq \text{enter}^\sharp(\mathcal{R}[8])$$
$$\mathcal{R}[9] \sqsupseteq \text{combine}^\sharp(\mathcal{R}[8], \mathcal{R}[10])$$

Caveat

The resulting super-graph contains obviously impossible paths ...

... in the Example:



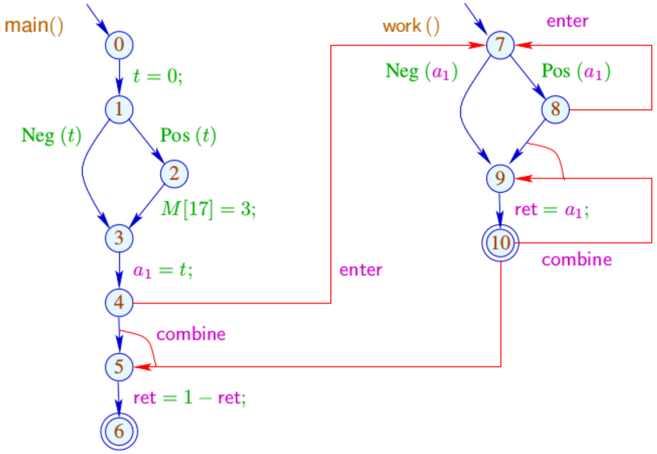
The conditions for $5, 7, 10$, e.g., are:

$$\mathcal{R}[5] \sqsupseteq \text{combine}^\sharp(\mathcal{R}[4], \mathcal{R}[10])$$
$$\mathcal{R}[7] \sqsupseteq \text{enter}^\sharp(\mathcal{R}[4])$$
$$\mathcal{R}[7] \sqsupseteq \text{enter}^\sharp(\mathcal{R}[8])$$
$$\mathcal{R}[9] \sqsupseteq \text{combine}^\sharp(\mathcal{R}[8], \mathcal{R}[10])$$

Caveat

The resulting super-graph contains obviously impossible paths ...

... in the Example:



The conditions for $5, 7, 10$, e.g., are:

$$\mathcal{R}[5] \sqsupseteq \text{combine}^\sharp\,(\mathcal{R}[4], \mathcal{R}[10])$$

$$\mathcal{R}[7] \sqsupseteq \text{enter}^\sharp\,(\mathcal{R}[4])$$

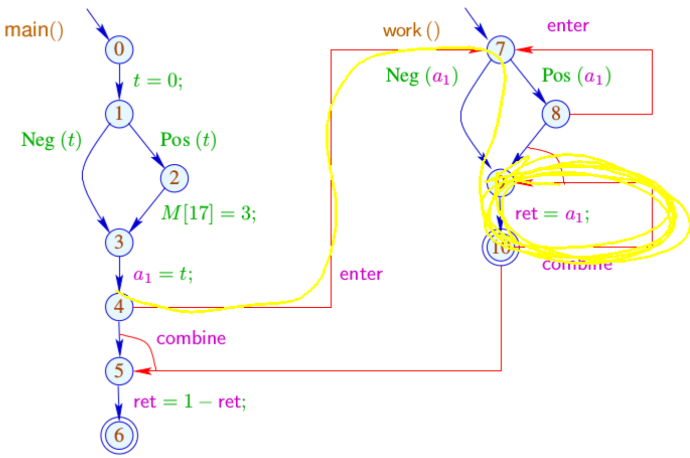
$$\mathcal{R}[7] \sqsupseteq \text{enter}^\sharp\,(\mathcal{R}[8])$$

$$\mathcal{R}[9] \sqsupseteq \text{combine}^\sharp\,(\mathcal{R}[8], \mathcal{R}[10])$$

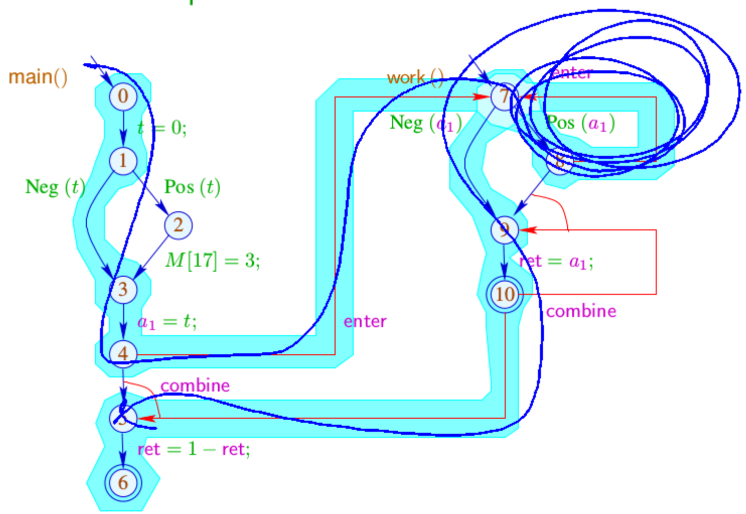
Caveat

The resulting super-graph contains obviously impossible paths ...

... in the Example this is:



... in the Example this is:



Note:

- → In the example, we find the same results:
  more paths render the results less precise.

  In particular, we provide for each procedure the result just for one (possibly very boring) argument.
- → The analysis terminates — whenever $\mathbb{D}$ has no infinite strictly ascending chains.
- → The correctness is easily shown w.r.t. the operational semantics with call stacks.
- → For the correctness of the functional approach, the semantics with computation forests is better suited.

Note:

→ In the example, we find the same results:
more paths render the results less precise.
In particular, we provide for each procedure the result just for one (possibly very boring) argument.
→ The analysis terminates — whenever $\mathbb{D}$ has no infinite strictly ascending chains.
→ The correctness is easily shown w.r.t. the operational semantics with call stacks.
→ For the correctness of the functional approach, the semantics with computation forests is better suited.

580

Note:

→ In the example, we find the same results:
more paths render the results less precise.
In particular, we provide for each procedure the result just for one (possibly very boring) argument.
→ The analysis terminates — whenever $\mathbb{D}$ has no infinite strictly ascending chains.
→ The correctness is easily shown w.r.t. the operational semantics with call stacks.
→ For the correctness of the functional approach, the semantics with computation forests is better suited.

580

3 Exploiting Hardware Features

Question: How can we optimally use:

- ... Registers
- ... Pipelines
- ... Caches
- ... Processors ???