Script generated by TTT

Title: Seidl: Programmoptimierung (23.12.2013)

Date: Mon Dec 23 14:15:37 CET 2013

Duration: 89:59 min

Pages: 29

$$X \longmapsto X \sqcup Y \sqcup S$$

$$Y \longmapsto Y$$

Improvement (Cont.):

→ Also, composition can be directly implemented:

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

→ The effects of assignments then are:

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

Improvement (Cont.):

→ Also, composition can be directly implemented:

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

→ The effects of assignments then are:

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

565

$$X \longmapsto X \sqcup Y \sqcup S$$

$$Y \longmapsto Y$$

$$y = \top; \quad x = y$$

$$x \mapsto \top$$

$$y \mapsto \top$$

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:

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

$$X \longmapsto X \sqcup Y \sqcup S$$

$$Y \longmapsto Y$$

$$(X \mapsto X) \sqcup (X \mapsto S) =$$

$$X \mapsto X \sqcup S$$

... in the Example:

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

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

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

(u, f, v) $\quad [v] \sqsupseteq H([f]) \circ [u]$

... in the Example:

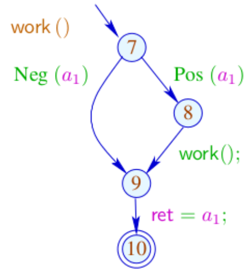
Hf

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

Now we can perform fixpoint iteration :-)

$M \circ \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto 0\}$

$= \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto 0\}$



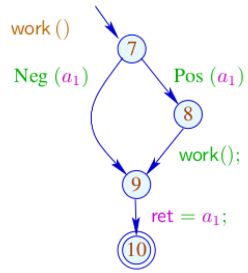
	1
7	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$
9	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$
10	$\{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}$
8	$\{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}$

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

... in the Example:

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

Now we can perform fixpoint iteration :-)



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

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

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:

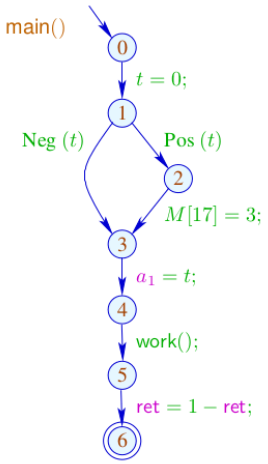
$$\begin{array}{llll}
\mathcal{R}[\mathsf{main}] & \sqsupseteq & \mathsf{enter}^\sharp\ d_0 & \\
\mathcal{R}[f] & \sqsupseteq & \mathsf{enter}^\sharp\ (\mathcal{R}[u]) & k = (u, f\,();, _) \quad \text{call} \\
\mathcal{R}[v] & \sqsupseteq & \mathcal{R}[f] & v \quad \text{entry point of} \quad f \\
\mathcal{R}[v]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

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:

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

$H([\![f]\!]^\#)$

... 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:

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

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

[f, a]

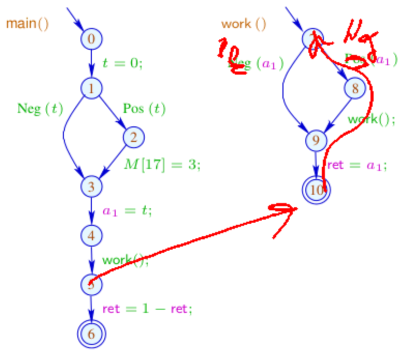
[f, a]

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	⊤	⊤	⊤	⊤
1	⊤	⊤	⊤	⊤
2	⊤	⊤	⊥	
3	⊤	⊤	⊤	⊤
4	⊤	⊤	0	⊤
7	0	⊤	0	⊤
8	0	⊤	⊥	
9	0	⊤	0	⊤
10	0	⊤	0	0
5	⊤	⊤	0	0
main()	⊤	⊤	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!

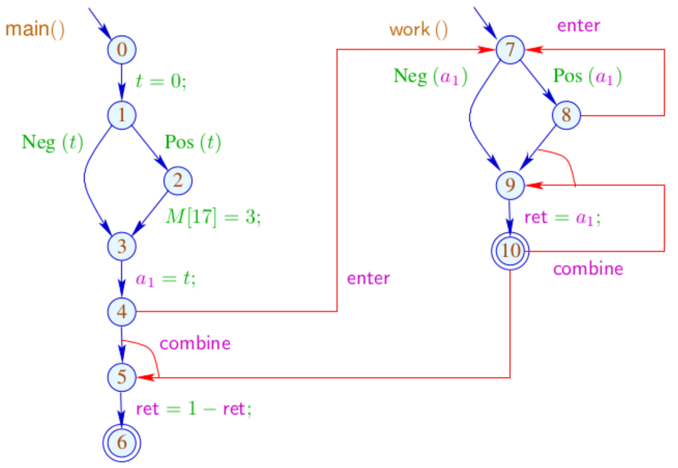
→ In general, this is infinite :-(

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

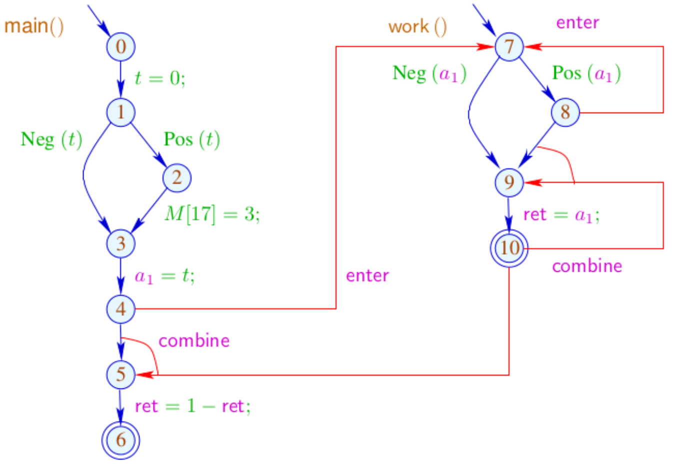
→ Important special case: $d = 0$.

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

... in the Example:



... in the Example:



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

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

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

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

Warning:

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

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

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

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

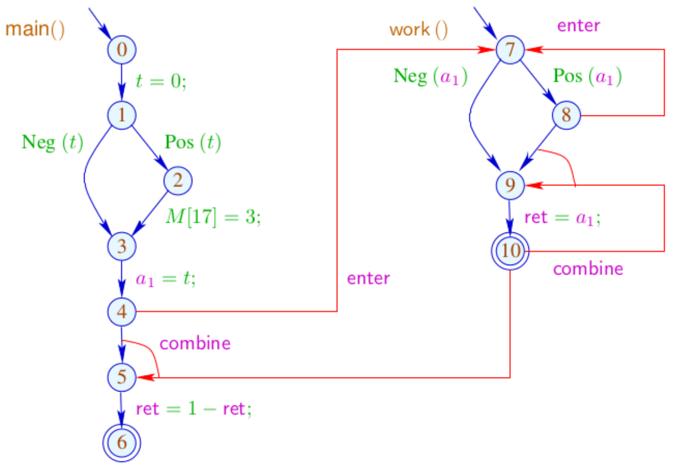
$$\mathcal{R}[7] \sqsupseteq \mathsf{enter}^\sharp\,(\mathcal{R}[8])$$

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

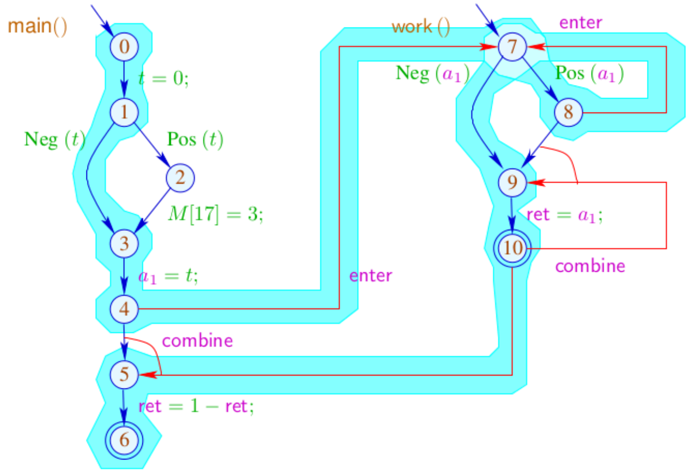
Warning:

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

... in the Example this is:



... in the Example this is:



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

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

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

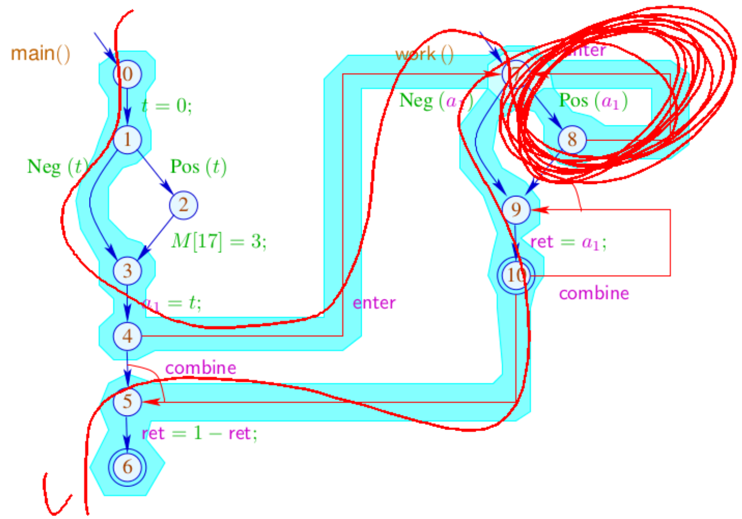
$$\mathcal{R}[7] \sqsupseteq \mathsf{enter}^\sharp\,(\mathcal{R}[8])$$

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

Warning:

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

... 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 :-)