Improvement (Cont.):

→ Also, composition can be directly implemented:

\[(M_1 \circ M_2) \ x = b' \cup \bigcup_{y \in I'} y\]

with

\[b' = b \cup \bigcup_{z \in I} b_z\]

\[I' = \bigcup_{z \in I} I_z\]

where

\[M_1 \ x = b \cup \bigcup_{y \in I} y\]

\[M_2 \ z = b_z \cup \bigcup_{y \in I_z} y\]

→ The effects of assignments then are:

\[\llbracket x = e; \rrbracket^\sharp = \begin{cases} \ld_{\text{Vars}} \oplus \{x \mapsto c\} & \text{if } e = c \in \mathbb{Z} \\ \ld_{\text{Vars}} \oplus \{x \mapsto y\} & \text{if } e = y \in \text{Vars} \\ \ld_{\text{Vars}} \oplus \{x \mapsto \top\} & \text{otherwise} \end{cases}\]
... in the Example:

\[
\begin{align*}
[t = 0;]^\# &= \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto 0\} \\
[a_1 = t;]^\# &= \{a_1 \mapsto t, \text{ret} \mapsto \text{ret}, t \mapsto t\}
\end{align*}
\]

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

\[
\begin{align*}
[k]^\# &= H ([f]^\#) \\
H (M) &= \text{Id}_{\text{Locals} \oplus \{x \mapsto (M \circ \text{enter}^\#)\}}_{\text{Globals}} \\
\text{enter}^\# x &= \begin{cases} 
  x & \text{if } x \in \text{Globals} \\
  0 & \text{otherwise}
\end{cases}
\end{align*}
\]
... in the Example:

If \([\text{work}]^\sharp = \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}\)

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\}\)

Now we can perform fixpoint iteration  :-)}
\[ [(8, \ldots, 9)]^\# \circ [8]^\# = \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\} \circ \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\} = \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\} \]
\[
\begin{align*}
\text{work}() \\
\text{Neg} (a_1) & \rightarrow 7 \\
\text{Pos} (a_1) & \rightarrow 8 \\
\text{work}(); & \rightarrow 9 \\
\text{ret} = a_1; & \rightarrow 10
\end{align*}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Node} & \text{Guard Set} \\
\hline
7 & \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\} \\
9 & \{a_1 \mapsto a_1, \text{ret} \mapsto a_1 \sqcup \text{ret}, t \mapsto t\} \\
10 & \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\} \\
8 & \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\} \\
\hline
\end{array}
\]

\[
\left[ (8, \ldots, 9) \right]^\# \circ [8]^\# = \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\} \circ \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\} = \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}
\]
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:

\[\begin{align*}
\mathcal{R}[\text{main}] & \sqsupseteq \text{enter}^d \ d_0 \\
\mathcal{R}[f] & \sqsupseteq \text{enter}^d (\mathcal{R}[u]) \quad k = (u, f();,\_) \quad \text{call} \\
\mathcal{R}[v] & \sqsubseteq \mathcal{R}[f] \quad v \quad \text{entry point of} \quad f \\
\mathcal{R}[v] & \sqsupseteq [k]^d (\mathcal{R}[u]) \quad k = (u,,v) \quad \text{edge}
\end{align*}\]
... in the Example:

main()

0
  \( t = 0; \)

1
  Neg (t)
  Pos (t)

2
  \( M[17] = 3; \)

3
  \( a_1 = t; \)

4
  work();

5
  ret = 1 − ret;

6

<table>
<thead>
<tr>
<th></th>
<th>( a_1 \mapsto \top, \text{ret} \mapsto \top, t \mapsto 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( a_1 \mapsto \top, \text{ret} \mapsto \top, t \mapsto 0 )</td>
</tr>
<tr>
<td>1</td>
<td>( a_1 \mapsto \top, \text{ret} \mapsto \top, t \mapsto 0 )</td>
</tr>
<tr>
<td>2</td>
<td>( a_1 \mapsto \top, \text{ret} \mapsto \top, t \mapsto 0 )</td>
</tr>
<tr>
<td>3</td>
<td>( a_1 \mapsto \top, \text{ret} \mapsto \top, t \mapsto 0 )</td>
</tr>
<tr>
<td>4</td>
<td>( a_1 \mapsto 0, \text{ret} \mapsto \top, t \mapsto 0 )</td>
</tr>
<tr>
<td>5</td>
<td>( a_1 \mapsto 0, \text{ret} \mapsto 0, t \mapsto 0 )</td>
</tr>
<tr>
<td>6</td>
<td>( a_1 \mapsto 0, \text{ret} \mapsto \top, t \mapsto 0 )</td>
</tr>
</tbody>
</table>
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 \( M \subseteq \mathcal{D} \to \mathcal{D} \) must be finite;
  (2) The functions \( M \in \mathcal{M} \) 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:

\[
\begin{align*}
[v, a] & \subseteq a & v & \text{entry point} \\
[v, a] & \subseteq \text{combine} ([u, a], [f, \text{enter}([u, a])]) & (u, f();, v) & \text{call} \\
[v, a] & \subseteq [lab][u, a] & k = (u, lab, v) & \text{edge} \\
[f, a] & \subseteq [stop_f, a] & \text{stop_f} & \text{end point of} \\ // & [v, a] & \text{value for the argument} & a .
\end{align*}
\]
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 \([\text{main()}, a_0]#\) \(\Rightarrow\) 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 ...

```
main()
0
  t = 0;
1
  Neg(t)
  Pos(t)
  M[17] = 3;
2
  a1 = t;
3
  work();
4
  ret = 1 - ret;
5
work()
7
  Neg(a1)
  Pos(a1)
8
  work();
9
  ret = a1;
10
```

<table>
<thead>
<tr>
<th></th>
<th>a1</th>
<th>ret</th>
<th>a1</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
<td>⊥</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>T</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>T</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>T</td>
<td>⊥</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>T</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>T</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>T</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>main()</td>
<td>T</td>
<td>T</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Discussion:

- In the Example, the analysis terminates quickly :-)
- If \( D \) 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 :-}

571