The effects \([f]^2\) then can be determined by a system of constraints over the complete lattice \(\mathbb{D} \rightarrow \mathbb{D}\):

\[
\begin{align*}
[v]^2 & \sqsubseteq \text{id} & v & \text{entry point} \\
[v]^2 & \sqsubseteq [k]^2 \circ [u]^1 & k = (u, \_ , v) & \text{edge} \\
[f]^1 & \supseteq [\text{stop}_f]^1 & \text{stop}_f & \text{end point of } f
\end{align*}
\]

\([v]^2 : \mathbb{D} \rightarrow \mathbb{D}\) describes the effect of all prefixes of computation forests \(w\) of a procedure which lead from the entry point to \(v\).

---

**Observation**

\[
\begin{align*}
\to & \quad \text{The effects of assignments are:} \\
\to & \quad [x = e]^2 D = \begin{cases} \\
D \oplus \{x \mapsto c\} & \text{if } e = c \in \mathbb{Z} \\
D \oplus \{x \mapsto (D y)\} & \text{if } e = y \in \text{Vars} \\
D \oplus \{x \mapsto \bot\} & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\to & \quad \text{Let } \mathbb{V} \text{ denote the (finite !!!!) set of constant right-hand sides. Then variables may only take values from } \mathbb{V}^\uparrow. \\
\to & \quad \text{The occurring effects can be taken from} \\
\to & \quad D_f \rightarrow D_f \quad \text{with} \\
\to & \quad D_f = (\text{Vars} \rightarrow \mathbb{V}^\uparrow)_\bot
\end{align*}
\]

\[
\begin{align*}
\to & \quad \text{The complete lattice is huge, but finite !!!!}
\end{align*}
\]

\[
\text{Seidl: Programmoptimierung (17.12.2015)}
\]

Title: Seidl: Programmoptimierung (17.12.2015)

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

Duration: 88:26 min

Pages: 39
Improvement

→ Not all functions from \( \mathbf{D_f} \to \mathbf{D_f} \) will occur.

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

\[
M = \{ x \mapsto (b_a \cup \bigsqcup_{y \in \text{id}_f} y) \mid x \in \text{Vars} \}
\]

\[
M D = \{ x \mapsto (b_a \cup \bigsqcup_{y \in \text{id}_f} D y) \mid x \in \text{Vars} \}
\]

für \( D \neq \bot \)

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

\[
(M_1 \cup M_2) x = (M_1 x) \cup (M_2 x)
\]

→ For \( k = \# \text{Vars} \), \( \mathbf{M} \) has height \( \mathcal{O}(k^3) \).

... in the Example:

\[
[t = 0]^2 = \{ a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto 0 \}
\]

\[
[a_1 = t]^2 = \{ (a_1 \mapsto t), \text{ret} \mapsto \text{ret}, t \mapsto t \}
\]

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

\[
[k]^4 = H ([f]^2)
\]

where:

\[
H (M) = \text{id}_{\text{Locals}} \otimes (M \circ \text{enter}^2)|_{\text{globals}}
\]

\[
\text{enter}^4 x = \begin{cases} x & \text{if } x \in \text{Globals} \\ 0 & \text{otherwise} \end{cases}
\]

Improvement (Cont.)

→ Also, composition can be directly implemented:

\[
(M_1 \circ M_2) x = b' \cup \bigsqcup_{y \in \text{id}_f} y
\]

with

\[
b' = b \cup \bigsqcup_{x \in f} b_x
\]

\[
\text{id}_f = \bigsqcup_{x \in f} \text{id}_x
\]

\[
M_1 x = b \cup \bigsqcup_{y \in \text{id}_f} y
\]

\[
M_2 z = b \cup \bigsqcup_{y \in \text{id}_f} y
\]

→ The effects of assignments then are:

\[
[x = e]^2 = \begin{cases} \text{id}_{\text{Vars}} \oplus \{ x \mapsto e \} & \text{if } e = c \in \mathbb{Z} \\ \text{id}_{\text{Vars}} \oplus \{ x \mapsto y \} & \text{if } e = y \in \text{Vars} \\ \text{id}_{\text{Vars}} \oplus \{ x \mapsto \mathcal{T} \} & \text{otherwise} \end{cases}
\]

Example: Constant Propagation
... in the Example:

\[
[t = 0] = \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\}
\]

\[
[a_1 = t] = \{(a_1 \mapsto t, \text{ret} \mapsto \text{ret}, t \mapsto t)\}
\]

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

\[
[k]^2 = H([f]^2)
\]

where:

\[
H(M) = \text{id}_{\text{locals}} \oplus (M \circ \text{enter}^2)_{\text{globals}}
\]

\[
\text{enter}^t x = \begin{cases} x & \text{if } x \in \text{Globals} \\ 0 & \text{otherwise} \end{cases}
\]

Now we can perform fixpoint iteration ...

\[
\begin{align*}
[(8, ..., 9)]^2 \circ [8]^2 &= \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\} \circ \\
&\quad \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\} \circ \\
&\quad \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\} \\
&= \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\}
\end{align*}
\]
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:

\[
\mathcal{R}[\text{main}] \not\sqsubseteq \text{enter}^t d_0
\]

\[
\mathcal{R}[f] \not\sqsubseteq \text{enter}^t (\mathcal{R}[u]) \quad k = (u, f();_v) \quad \text{call}
\]

\[
\mathcal{R}[u] \not\sqsubseteq \mathcal{R}[f] \quad v \quad \text{entry point of } f
\]

\[
\mathcal{R}[v] \not\sqsubseteq k^t (\mathcal{R}[u]) \quad k = (u, \ldots, v) \quad \text{edge}
\]
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 $\mathcal{M} \subseteq \mathcal{D} \to \mathcal{D}$ must be finite;
  2. The functions $M \in \mathcal{M}$ must be efficiently implementable.
- The second condition can, sometimes, be abandoned ...
Observation

→ Often, procedures are only called for few distinct abstract arguments.
→ Each procedure need only to be analyzed for these.
→ Put up a constraint system:

$$\begin{align*}
[v, a] & \subseteq a & v \text{ entry point} \\
[v, a] & \subseteq \text{combine}([u, a]^t, [f, \text{enter}^t, [v, a]^t]) & (u, f(), v) \text{ call} \\
[v, a] & \subseteq [\text{lab}]^t[u, a]^t & k = (u, \text{lab}, v) \text{ edge} \\
[f, a] & \subseteq [\text{stop_f}, a]^t & \text{stop_f end point of } f \\
// [v, a]^t & \quad \text{value for the argument } a \\
\end{align*}$$

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 $[\text{main}, a]^2$ We apply our local fixpoint algorithm!
- The fixpoint algo provides us also with the set of actual parameters $a \in D$ for which procedures are (possibly) called and all abstract values at their program points for each of these calls.

Observation

→ Often, procedures are only called for few distinct abstract arguments.
→ Each procedure need only to be analyzed for these.
→ Put up a constraint system:

$$\begin{align*}
[v, a] & \subseteq a & v \text{ entry point} \\
[v, a] & \subseteq \text{combine}([u, a]^t, [f, \text{enter}^t, [v, a]^t]) & (u, f(), v) \text{ call} \\
[v, a] & \subseteq [\text{lab}]^t[u, a]^t & k = (u, \text{lab}, v) \text{ edge} \\
[f, a] & \subseteq [\text{stop_f}, a]^t & \text{stop_f end point of } f \\
// [v, a]^t & \quad \text{value for the argument } a \\
\end{align*}$$

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 $[\text{main}, a]^2$ We apply our local fixpoint algorithm!
- The fixpoint algo provides us also with the set of actual parameters $a \in D$ 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 ...

<table>
<thead>
<tr>
<th></th>
<th>$a_1$</th>
<th>ret</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>T</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.

(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:

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

Caveat

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

... in the Example:

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

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

Caveat

The resulting super-graph contains obviously impossible paths ...
... in the Example:

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

\[ \mathcal{R}[5] \subseteq \text{combine}^3(\mathcal{R}[4], \mathcal{R}[10]) \]
\[ \mathcal{R}[7] \subseteq \text{enter}^4(\mathcal{R}[4]) \]
\[ \mathcal{R}[7] \subseteq \text{enter}^4(\mathcal{R}[8]) \]
\[ \mathcal{R}[9] \subseteq \text{combine}^3(\mathcal{R}[8], \mathcal{R}[10]) \]

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 $D$ 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.

3 Exploiting Hardware Features

Question: How can we optimally use:

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