Problems:

- How can we represent functions \( f : \mathbb{D} \rightarrow \mathbb{D} \) ???
- If \( \#\mathbb{D} = \infty \), then \( \mathbb{D} \rightarrow \mathbb{D} \) has infinite strictly increasing chains  :-(

Simplification: Copy-Constants

\[ \rightarrow \text{Conditions are interpreted as } \ ; \ ; \ :-) \]
\[ \rightarrow \text{Only assignments } x = e; \text{ with } e \in Vars \cup \mathbb{Z} \text{ are treated exactly } :-) \]
Observation:

→ The effects of assignments are:

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

→ Let \( \forall \) denote the (finite !!!) set of constant right-hand sides. Then variables may only take values from \( \forall^\top :-)) \)

→ The occurring effects can be taken from

\[
\mathbb{D}_f \rightarrow \mathbb{D}_f \quad \text{with} \quad \mathbb{D}_f = (\text{Vars} \rightarrow \forall^\top)_{\bot}
\]

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

→ Not all functions from $\mathbb{D}_f \to \mathbb{D}_f$ will occur $\therefore$

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

$$M = \{ x \mapsto (b_x \sqcup \bigcup_{y \in I_x} y) \mid x \in Vars \}$$

where:

$$M D = \{ x \mapsto (b_x \sqcup \bigcup_{y \in I_x} D y) \mid x \in Vars \}$$

für $D \neq \bot$

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

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

→ For $k = \# Vars$, $\mathcal{M}$ has height $O(k^2) \therefore$
Improvement  (Cont.):

→ Also, composition can be directly implemented:

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

with

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

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

where

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

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

→ The effects of assignments then are:

\[[x = e;] \# = \begin{cases} 
\text{Id}_{\text{Vars}} \oplus \{x \mapsto c\} & \text{if } e = c \in \mathbb{Z} \\
\text{Id}_{\text{Vars}} \oplus \{x \mapsto y\} & \text{if } e = y \in \text{Vars} \\
\text{Id}_{\text{Vars}} \oplus \{x \mapsto \top\} & \text{otherwise} 
\end{cases}\]
... in the Example:

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

\[
[a_1 = t;]^{\#} = \{a_1 \mapsto t, \text{ret} \mapsto \text{ret}, t \mapsto t\}
\]

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

\[
[k]^{\#} = H([f]^{\#}) \quad \text{where:}
\]

\[
H(M) = \text{Id}|_{\text{Locals} \oplus (M \circ \text{enter}^{\#})}|_{\text{Globals}}
\]

\[
\text{enter}^{\#} x = \begin{cases} 
  x & \text{if } x \in \text{Globals} \\
  0 & \text{otherwise}
\end{cases}
\]
... in the Example:

If $\text{[work]}^\# = \{ a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t \}$
then $H \text{[work]}^\# = \text{ld}_\{t\} \oplus \{ a_1 \mapsto a_1, \text{ret} \mapsto a_1 \}$

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

Now we can perform fixpoint iteration :-)

563
work();

Neg \(a_1\)  

Pos \(a_1\)

ret = \(a_1\);

\[
\begin{array}{|c|c|}
\hline
\text{1} & \\
\hline
7 & \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\} \\
9 & \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\} \\
10 & \{a_1 \mapsto a_1, \text{ret} \mapsto a_1, t \mapsto t\} \\
8 & \{a_1 \mapsto a_1, \text{ret} \mapsto \text{ret}, t \mapsto t\} \\
\hline
\end{array}
\]

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

Neg\(a_1\)  Pos\(a_1\)  work();

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

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

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

```plaintext
main()

0
  t = 0;

1
  M[17] = 3;

2
  a_1 = t;

3
  work();

4
  ret = 1 - ret;

5

6

Neg(t)                 Pos(t)

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 $M \subseteq D \rightarrow D$ must be finite;
  2. The functions $M \in M$ 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:

\[
{[v, a]}^\# \sqsubseteq a
\]

\[\text{entry point}\]

\[
{[v, a]}^\# \sqsubseteq \text{combine}^\# ([u, a], [f, \text{enter}^\# [u, a]^\#])
\]

\[(u, f(); v) \text{ call}\]

\[
{[v, a]}^\# \sqsubseteq [lab]^\# [u, a]^\# \quad k = (u, lab, v) \quad \text{edge}\]

\[
[f, a]^\# \sqsubseteq [stop_f, a]^\# \quad stop_f \quad \text{end point of } f
\]

// \[{[v, a]}^\# \rightleftharpoons v \text{ value for the argument } a.\]
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 $\llbracket \text{main()}, a_0 \rrbracket^\# \implies$ We apply our local fixpoint algorithm :-))
- The fixpoint algo provides us also with the set of actual parameters $a \in \mathcal{D}$ for which procedures are (possibly) called and all abstract values at their program points for each of these calls :-)

570
... in the Example:

Let us try a **full** constant propagation ...

```
main()

0
  \ t = 0;

1
  Pos (t)

2
  M[17] = 3;

3
  a_1 = t;

4
  work();

5
  ret = 1 - ret;

6

7
  Neg (a_1)

8
  Pos (a_1)

9
  work();

10
  ret = a_1;

```

<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></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
main()     T   T
```

**571**
Discussion:

- In the Example, the analysis terminates quickly :-(
- If $\mathcal{D}$ 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 ...