Problem:

→ The solution can be computed with RR-iteration —
  after about 42 rounds :-(
→ On some programs, iteration may never terminate :((

Idea 1: Widening

- Accelerate the iteration — at the prize of imprecision :)
- Allow only a bounded number of modifications of values !!!
  ... in the Example:
- dis-allow updates of interval bounds in $\mathbb{Z}$ ...

$\implies$ a maximal chain:

$[3,17] \sqsubset [3,\infty] \sqsubset [-\infty,\infty]$

(c) The sequence $G^k \sqsubset \Downarrow$, $k \geq 0$, is an ascending chain:

$\Downarrow \subseteq G \subseteq \cdots \subseteq G^k \subseteq \cdots$

(d) If $G^k \sqsubset \Downarrow = y$, then $y$ is a solution of (1).

(e) If $\mathbb{D}$ has infinite strictly ascending chains, then (d) is not yet sufficient ...

but: we could consider the modified system of equations:

$x_i = x_i \sqcup f_i(x_1,\ldots,x_n)$, $i = 1,\ldots,n$ (3)

for a binary operation widening:

$\sqcup : \mathbb{D} \rightarrow \mathbb{D}$ with $v_1 \sqcup v_2 \subseteq v_1 \sqcup v_2$

(RR)-iteration for (3) still will compute a solution of (1) :-)
... for Interval Analysis:

- The complete lattice is: \( D_I = (\text{Vars} \rightarrow \mathbb{L})_\perp \)
- the widening \( \sqcup \) is defined by:
  \[
  \perp \sqcup D = D \sqcup \perp = D \quad \text{and for} \quad D_1 \neq \perp \neq D_2:
  \]
  \[
  (D_1 \sqcup D_2) \cdot x = (D_1 \cdot x) \sqcup (D_2 \cdot x) \quad \text{where}
  \]
  \[
  [l_1, u_1] \sqcup [l_2, u_2] = [l, u] \quad \text{with}
  \]
  \[
  l = \begin{cases} 
  l_1 & \text{if } l_1 \leq l_2 \\
  -\infty & \text{otherwise}
  \end{cases}
  \]
  \[
  u = \begin{cases} 
  u_1 & \text{if } u_1 \geq u_2 \\
  +\infty & \text{otherwise}
  \end{cases}
  \]

\[ \implies \quad \sqcup \quad \text{is not commutative} !!! \]

Example:

\[
[0, 2] \sqcup [1, 2] = [0, 2]
\]
\[
[1, 2] \sqcup [0, 2] = [-\infty, 2]
\]
\[
[1, 5] \sqcup [3, 7] = [1, +\infty]
\]

\[ \rightarrow \quad \text{Widening returns larger values more quickly.} \]
\[ \rightarrow \quad \text{It should be constructed in such a way that termination of iteration is guaranteed} \quad \Rightarrow \]
\[ \rightarrow \quad \text{For interval analysis, widening bounds the number of iterations by:} \quad \#	ext{points} \cdot (1 + 2 \cdot \# \text{Vars}) \]

Conclusion:

- In order to determine a solution of (1) over a complete lattice with infinite ascending chains, we define a suitable widening and then solve (3) \( \Rightarrow \)

- Caveat: The construction of suitable widenings is a dark art !!!
  Often \( \sqcup \) is chosen dynamically during iteration such that
  \[ \rightarrow \quad \text{the abstract values do not get too complicated;} \]
  \[ \rightarrow \quad \text{the number of updates remains bounded ...} \]

<table>
<thead>
<tr>
<th>i</th>
<th>( \mu )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-\infty</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>\perp</td>
</tr>
<tr>
<td>8</td>
<td>\perp</td>
</tr>
</tbody>
</table>

Our Example:
... obviously, the result is disappointing :-(

Idea 2:

In fact, acceleration with \( \sqcup \) need only be applied at sufficiently many places!

A set \( I \) is a loop separator, if every loop contains at least one point from \( I \) :-(

If we apply widening only at program points from such a set \( I \), then RR-iteration still terminates !!!
The Analysis with \( I = \{1\} \):

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>u</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>l</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>u</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>l</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

\( i = i + 1 \);

Neg(\( i < 42 \))
Pos(\( i < 42 \))
Neg(\( 0 \leq i < 42 \))
Pos(\( 0 \leq i < 42 \))

\( A_1 = A + i \)
\( M[A_1] = i \)

The Analysis with \( I = \{2\} \):

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>u</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>l</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>u</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
<td>( \infty )</td>
</tr>
<tr>
<td>l</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\( i = i + 1 \);

Neg(\( i < 42 \))
Pos(\( i < 42 \))
Neg(\( 0 \leq i < 42 \))
Pos(\( 0 \leq i < 42 \))

\( A_1 = A + i \)
\( M[A_1] = i \)

Discussion:

- Both runs of the analysis determine interesting information :-)
- The run with \( I = \{2\} \) proves that always \( i = 42 \) after leaving the loop.
- Only the run with \( I = \{1\} \) finds, however, that the outer check makes the inner check superfluous :-(

How can we find a suitable loop separator \( I \) ???

Idea 3: Narrowing

Let \( \bar{x} \) denote any solution of \( (1) \), i.e.,

\[
F_i \models f_i \bar{x} \quad i = 1, \ldots, n
\]

Then for monotonic \( f_i \),

\[
\bar{x} \sqsupset F \sqsupset F^o \sqsupset \cdots \sqsupset F^k \sqsupset \cdots
\]

// Narrowing Iteration
Idea 3: Narrowing

Let $\mathbf{F}$ denote any solution of $(1)$, i.e.,

$$x_i \supseteq f_i \mathbf{F}, \quad i = 1, \ldots, n$$

Then for monotonic $f_i$,

$$\mathbf{F} \supseteq F \mathbf{F} \supseteq F^2 \mathbf{F} \supseteq \ldots \supseteq F^k \mathbf{F} \supseteq \ldots$$

// Narrowing Iteration

Every tuple $F^k \mathbf{F}$ is a solution of $(1)$:

$\Rightarrow$

Termination is no problem anymore:
we stop whenever we want $\Rightarrow$

// The same also holds for RR-iteration.

\begin{tabular}{|c|c|c|c|c|c|}
\hline
\textbf{i} & \textbf{u} & \textbf{i} & \textbf{u} & \textbf{i} & \textbf{u} \\
\hline
0 & $-\infty$ & 0 & $-\infty$ & 0 & $-\infty$ \\
1 & 0 & 0 & 0 & 0 & 0 \\
2 & 0 & $+\infty$ & 0 & 41 & 0 & 41 \\
3 & 0 & $+\infty$ & 0 & 41 & 0 & 41 \\
4 & 0 & $+\infty$ & 0 & 41 & 0 & 41 \\
5 & 0 & $+\infty$ & 0 & 41 & 0 & 41 \\
6 & 1 & $+\infty$ & 1 & 42 & 1 & 42 \\
7 & 42 & $+\infty$ & 1 & 42 & 1 & 42 \\
8 & 42 & $+\infty$ & 42 & $+\infty$ & 42 & $+\infty$ \\
\hline
\end{tabular}
Discussion:

→ We start with a safe approximation.
→ We find that the inner check is redundant :-)
→ We find that at exit from the loop, always i = 42 :-()
→ It was not necessary to construct an optimal loop separator :-)))

Last Question:
Do we have to accept that narrowing may not terminate ???

4. Idea: Accelerated Narrowing

Assume that we have a solution \( \mathbf{x} = (x_1, \ldots, x_n) \) of the system of constraints:

\[
x_i \sqsupset f_i(x_1, \ldots, x_n), \quad i = 1, \ldots, n
\]

(1)

Then consider the system of equations:

\[
x_i = x_i \sqcap f_i(x_1, \ldots, x_n), \quad i = 1, \ldots, n
\]

(4)

Obviously, we have for monotonic \( f_i: H^k \sqsupset F^k \sqsubset \quad \vdash \)
where \( H(x_1, \ldots, x_n) = (y_1, \ldots, y_n), \quad y_i = x_i \sqsupset f_i(x_1, \ldots, x_n). \)

In (4), we replace \( \sqcap \) by the novel operator \( \sqcap ^{\sqcap} \) where:

\[
a_1 \sqcap a_2 \sqsubset a_1 \sqcap a_2 \sqsubset a_1
\]

... for Interval Analysis:

We preserve finite interval bounds :-)

Therefore, \( \bot \sqcap D = D \sqcap \bot = \bot \) and for \( D_1 \neq \bot \neq D_2: \)

\[
(D_1 \sqcap D_2) x = (D_1 x) \sqcap (D_2 x)
\]

where

\[
[l_1, u_1] \sqcap [l_2, u_2] = [l, u]
\]

with

\[
l = \begin{cases} 
  l_2 & \text{if } l_1 = -\infty \\
  l_1 & \text{otherwise}
\end{cases}
\]

\[
u = \begin{cases} 
  u_2 & \text{if } u_1 = \infty \\
  u_1 & \text{otherwise}
\end{cases}
\]

\( \Rightarrow \) \( \sqcap \) is not commutative !!!

... for Interval Analysis:

We preserve finite interval bounds :-)

Therefore, \( \bot \sqcap D = D \sqcap \bot = \bot \) and for \( D_1 \neq \bot \neq D_2: \)

\[
(D_1 \sqcap D_2) x = (D_1 x) \sqcap (D_2 x)
\]

where

\[
[l_1, u_1] \sqcap [l_2, u_2] = [l, u]
\]

with

\[
l = \begin{cases} 
  l_2 & \text{if } l_1 = -\infty \\
  l_1 & \text{otherwise}
\end{cases}
\]

\[
u = \begin{cases} 
  u_2 & \text{if } u_1 = \infty \\
  u_1 & \text{otherwise}
\end{cases}
\]

\( \Rightarrow \) \( \sqcap \) is not commutative !!!
Discussion:

→ Caveat: Widening also returns for non-monotonic \( f_i \) a solution. Narrowing is only applicable to monotonic \( f_i \) !!

→ In the example, accelerated narrowing already returns the optimal result :-(

→ If the operator \( \cap \) only allows for finitely many improvements of values, we may execute narrowing until stabilization.

→ In case of interval analysis these are at most:

\[
\#\text{points} \cdot (1 + 2 \cdot \#\text{Vars})
\]

1.6 Pointer Analysis

Questions:

→ Are two addresses possibly equal?

→ Are two addresses definitively equal?
1.6 Pointer Analysis

Questions:

→ Are two addresses possibly equal? May Alias
→ Are two addresses definitively equal? Must Alias

⇒⇒⇒ Alias Analysis

(2) Values of Variables:

• Extend the set Expr of expressions by occurring loads $M[e]$.

• Extend the Effects of Edges:

$$[x = M[e]]^T V e' = \begin{cases} \{x\} & \text{if } e' = M[e] \\ \emptyset & \text{if } e' = e \\ V e' \setminus \{x\} & \text{otherwise} \end{cases}$$

$$[M[e_1] = e_2]^T V e' = \begin{cases} \emptyset & \text{if } e' \in \{e_1, e_2\} \\ V e' & \text{otherwise} \end{cases}$$

The analyses so far without alias information:

1. Available Expressions:

• Extend the set Expr of expressions by occurring loads $M[e]$.

• Extend the Effects of Edges:

$$[x = e]_x^T A = (A \cup \{e\}) \setminus Expr_x$$
$$[x = M[e]]^T A = (A \cup \{e, M[e]\}) \setminus Expr_e$$
$$[M[e_1] = e_2]^T A = (A \cup \{e_1, e_2\}) \setminus Loads$$

(3) Constant Propagation:

• Extend the abstract state by an abstract store $M$

• Execute accesses to known memory locations!

$$[x = M[e]]^T (D, M) = \begin{cases} (D \oplus \{x \mapsto M[a]\}, M) & \text{if } a \subseteq T \\ (D \oplus \{x \mapsto \top\}, M) & \text{otherwise} \end{cases}$$

$$[M[e_1] = e_2]^T (D, M) = \begin{cases} (D \oplus \{a \mapsto e_1\}, D) & \text{if } e_1 \subseteq T \\ (D, \top) & \text{otherwise} \end{cases}$$

$$\top a = \top (a \in N)$$
Constant Propagation:

- Extend the abstract state by an abstract store $M$.
- Execute accesses to known memory locations!

$$\begin{align*}
    [x = M[a]]^3(D, M) &= \begin{cases} 
    (D \oplus \{x \mapsto M a\}, M) & \text{if } [a]^3 D = a \sqsubseteq T \\
    (D \oplus \{x \mapsto \top\}, M) & \text{otherwise}
    \end{cases} \\
    [\overline{M}\{c_1\}]^3(D, M) &= \begin{cases} 
    (D, \top) & \text{if } [c_1]^3 D = a \sqsubseteq T \\
    (D, \top) & \text{otherwise where } (a \in \mathbb{N})
    \end{cases}
\end{align*}$$

Problems:

- Addresses are from $\mathbb{N}$ also.
  - There are no infinite strictly ascending chains, but ...
- Exact addresses at compile-time are rarely known :-(
- At the same program point, typically different addresses are accessed ...
- Storing at an unknown address destroys all information $M$ :-(

$\Rightarrow$ constant propagation fails :-(
$\Rightarrow$ memory accesses/pointers kill precision :-(

Simplification:

- We consider pointers to the beginning of blocks $A[i]$ which allow indexed accesses $A[i] ::=)$.
- We ignore well-typedness of the blocks.
- New statements:
  
  - $x = \text{new}();$ // allocation of a new block
  - $x = y[e];$ // indexed read access to a block
  - $y[e1] = e2;$ // indexed write access to a block

- Blocks are possibly infinite :-(
- For simplicity, all pointers point to the beginning of a block.