Example:

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

→ Widening returns larger values more quickly.
→ It should be constructed in such a way that termination of iteration is guaranteed :-) 
→ For interval analysis, widening bounds the number of iterations by:

\[
#points \cdot (1 + 2 \cdot \#Vars)
\]
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) :-)

• **Warning:** The construction of suitable widenings is a dark art !!!

Often ⊔ is chosen dynamically during iteration such that

→ the abstract values do not get too complicated;

→ the number of updates remains bounded ...
Our Example:

\[ i = 0; \]

Neg \((i < 42)\)

Pos \((i < 42)\)

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

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

\[ A_1 = A + i; \]

\[ M[A_1] = i; \]

\[ i = i + 1; \]

\[
\begin{array}{c|cc}
 l & \mathopen{1} \\
 \hline
 0 & -\infty & +\infty \\
 1 & 0 & 0 \\
 2 & 0 & 0 \\
 3 & 0 & 0 \\
 4 & 0 & 0 \\
 5 & 0 & 0 \\
 6 & 1 & 1 \\
 7 & \perp & \\
 8 & \perp & \\
\end{array}
\]
Our Example:

0

i = 0;

Neg(i < 42)  Pos(i < 42)

1

Neg(0 ≤ i < 42)  Pos(0 ≤ i < 42)

2

3

A_1 = A + i;

4

M[A_1] = i;

5

i = i + 1;

6

7

8

Neg(0 ≤ i < 42)

Neg(0 ≤ i < 42)

Pos(0 ≤ i < 42)

Pos(i < 42)

Neg(i < 42)

---

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th></th>
<th>2</th>
<th></th>
<th>3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>u</td>
<td>l</td>
<td>u</td>
<td>l</td>
<td>u</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>−∞</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>+∞</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+∞</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+∞</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+∞</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+∞</td>
<td></td>
</tr>
<tr>
<td>dito</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+∞</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>+∞</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>⊥</td>
<td>42</td>
<td>+∞</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>⊥</td>
<td>42</td>
<td>+∞</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
... 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 !!!
In our Example:

\[
\begin{align*}
i & = 0; \\
1 & \quad \text{Pos}(i < 42) \\
8 & \quad \text{Neg}(i < 42) \\
2 & \quad \text{Pos}(0 \leq i < 42) \\
7 & \quad \text{Neg}(0 \leq i < 42) \\
3 & \quad A_1 = A + i; \\
4 & \quad M[A_1] = i; \\
5 & \quad i = i + 1; \\
6 & \\
\end{align*}
\]

\[
\begin{align*}
I_1 & = \{1\} \quad \text{or:} \\
I_2 & = \{2\} \quad \text{or:} \\
I_3 & = \{3\}
\end{align*}
\]
The Analysis with $I = \{1\}$:

\begin{align*}
i &= 0; \\
\text{Neg}(i < 42) &\quad \text{Pos}(i < 42) \\
\text{Neg}(0 \leq i < 42) &\quad \text{Pos}(0 \leq i < 42)
\end{align*}

\[ A_1 = A + i; \]

\[ M[A_1] = i; \]

\[ i = i + 1; \]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l$</td>
<td>$u$</td>
<td>$l$</td>
<td>$u$</td>
</tr>
<tr>
<td>0</td>
<td>$-\infty$</td>
<td>$+\infty$</td>
<td>$-\infty$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>$\perp$</td>
<td>$\perp$</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>$\perp$</td>
<td>42</td>
<td>$+\infty$</td>
</tr>
</tbody>
</table>

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

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& 1 & 2 & 3 & 4 \\
\hline
l & u & l & u & l & u \\
\hline
0 & -\infty & +\infty & -\infty & +\infty & -\infty & +\infty \\
1 & 0 & 0 & 0 & 1 & 0 & 42 \\
2 & 0 & 0 & 0 & +\infty & 0 & +\infty \\
3 & 0 & 0 & 0 & 41 & 0 & 41 \\
4 & 0 & 0 & 0 & 41 & 0 & 41 \\
5 & 0 & 0 & 0 & 41 & 0 & 41 \\
6 & 1 & 1 & 1 & 42 & 1 & 42 \\
7 & \perp & 42 & +\infty & 42 & +\infty & \perp \\
8 & \perp & \perp & 42 & 42 & \perp \\
\hline
\end{array}
\]

\( A_1 = A + i; \)

\( M[A_1] = i; \)

\( i = i + 1; \)
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 $x$ denote any solution of (1), i.e.,

$$x_i \supseteq f_i x, \quad i = 1, \ldots, n$$

Then for monotonic $f_i$,

$$x \supseteq F x \supseteq F^2 x \supseteq \ldots \supseteq F^k x \supseteq \ldots$$

// Narrowing Iteration
Idea 3: Narrowing

Let $x$ denote any solution of (1), i.e.,

$$x_i \supseteq f_i x, \quad i = 1, \ldots, n$$

Then for monotonic $f_i$,

$$x \supseteq Fx \supseteq F^2 x \supseteq \ldots \supseteq F^k x \supseteq \ldots$$

// Narrowing Iteration

Every tuple $F^k x$ is a solution of (1) :-) 

Termination is no problem anymore: we stop whenever we want :-))

// The same also holds for RR-iteration.
Narrowing Iteration in the Example:

\[
\begin{array}{c|c|c}
 l & u \\
 \hline
 0 & -\infty & +\infty \\
 1 & 0 & +\infty \\
 2 & 0 & +\infty \\
 3 & 0 & +\infty \\
 4 & 0 & +\infty \\
 5 & 0 & +\infty \\
 6 & 1 & +\infty \\
 7 & 42 & +\infty \\
 8 & 42 & +\infty \\
\end{array}
\]
Narrowing Iteration in the Example:

\[ i = 0; \]

\[ \text{Neg}(i < 42) \]
\[ \text{Pos}(i < 42) \]
\[ \text{Neg}(0 \leq i < 42) \]
\[ \text{Pos}(0 \leq i < 42) \]

\[ A_1 = A + i; \]
\[ M[A_1] = i; \]
\[ i = i + 1; \]

\[
\begin{array}{|c|c|c|c|}
\hline
 & 0 & & 1 \\
\hline
 l & u & l & u \\
\hline
 0 & -\infty & +\infty & -\infty & +\infty \\
 1 & 0 & +\infty & 0 & +\infty \\
 2 & 0 & +\infty & 0 & 41 \\
 3 & 0 & +\infty & 0 & 41 \\
 4 & 0 & +\infty & 0 & 41 \\
 5 & 0 & +\infty & 0 & 41 \\
 6 & 1 & +\infty & 1 & 42 \\
 7 & 42 & +\infty & \perp & \\
 8 & 42 & +\infty & 42 & +\infty \\
\hline
\end{array}
\]
Narrowing Iteration in the Example:

\[ i = 0; \]

\[
\begin{align*}
&\text{Neg}(i < 42) & &\text{Pos}(i < 42) \\
&\text{Neg}(0 \leq i < 42) & &\text{Pos}(0 \leq i < 42)
\end{align*}
\]

\[ A_1 = A + i; \]

\[ M[A_1] = i; \]

\[ i = i + 1; \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th></th>
<th>1</th>
<th></th>
<th>2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>−∞</td>
<td>u</td>
<td>−∞</td>
<td>u</td>
<td>−∞</td>
<td>u</td>
</tr>
<tr>
<td>0</td>
<td>−∞</td>
<td>+∞</td>
<td>−∞</td>
<td>+∞</td>
<td>−∞</td>
<td>+∞</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>+∞</td>
<td>0</td>
<td>+∞</td>
<td>0</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>+∞</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>+∞</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>+∞</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>+∞</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td>41</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>+∞</td>
<td>1</td>
<td>42</td>
<td>1</td>
<td>42</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>+∞</td>
<td>⊥</td>
<td></td>
<td>⊥</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>42</td>
<td>+∞</td>
<td>42</td>
<td>+∞</td>
<td>42</td>
<td>42</td>
</tr>
</tbody>
</table>
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 \( x = (x_1, \ldots, x_n) \) of the system of constraints:

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

(1)

Then consider the system of equations:

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

(4)

Obviously, we have for monotonic \( f_i \):

\[
H^k x = F^k x \quad :-)
\]

where \( H (x_1, \ldots, x_n) = (y_1, \ldots, y_n), \quad y_i = x_i \cap f_i (x_1, \ldots, x_n) \).

In (4), we replace \( \cap \) durch by the novel operator \( \sqsubseteq \) where:

\[
a_1 \sqsubseteq a_2 \sqsubseteq a_1
\]
... for Interval Analysis:

We preserve finite interval bounds  

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

\[
\begin{align*}
(D_1 \sqcap D_2) x &= (D_1 x) \sqcap (D_2 x) \\
[l_1, u_1] \sqcap [l_2, u_2] &= [l, u]
\end{align*}
\]

where

\[
\begin{align*}
l &= \left\{ \begin{array}{ll}
l_2 & \text{if } l_1 = -\infty \\
l_1 & \text{otherwise}
\end{array} \right. \\
u &= \left\{ \begin{array}{ll}
u_2 & \text{if } u_1 = \infty \\
u_1 & \text{otherwise}
\end{array} \right.
\]

\[\rightarrow \quad \sqcap \quad \text{is not commutative} !!!\]
Accelerated Narrowing in the Example:

\[ i = 0; \]

\begin{align*}
\text{Neg}(i < 42) & \quad \text{Pos}(i < 42) \\
\text{Neg}(0 \leq i < 42) & \quad \text{Pos}(0 \leq i < 42)
\end{align*}

\[ A_1 = A + i; \]

\[ M[A_1] = i; \]

\[ i = i + 1; \]

\[
\begin{array}{c|c|c|c|c|c}
\text{l} & \text{u} & \text{l} & \text{u} & \text{l} & \text{u} \\
\hline
0 & -\infty & +\infty & -\infty & +\infty & -\infty & +\infty \\
1 & 0 & +\infty & 0 & +\infty & 0 & 42 \\
2 & 0 & +\infty & 0 & 41 & 0 & 41 \\
3 & 0 & +\infty & 0 & 41 & 0 & 41 \\
4 & 0 & +\infty & 0 & 41 & 0 & 41 \\
5 & 0 & +\infty & 0 & 41 & 0 & 41 \\
6 & 1 & +\infty & 1 & 42 & 1 & 42 \\
7 & 42 & +\infty & \bot & \bot & \bot & \bot \\
8 & 42 & +\infty & 42 & +\infty & 42 & 42 \\
\end{array}
\]
Discussion:

→ **Warning:** 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 $\sqcap$ only allows for finitely many improvements of values, we may execute narrowing until stabilization.

→ In case of interval analysis these are at most:

$$\#points \cdot (1 + 2 \cdot \#Vars)$$
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
The analyses so far without alias information:

(1) Available Expressions:

- Extend the set $\text{Expr}$ of expressions by occurring loads $M[e]$.

- Extend the Effects of Edges:

  $$\begin{align*}
  [x = e;] \# A &= (A \cup \{e\}) \setminus \text{Expr}_x \\
  [x = M[e];] \# A &= (A \cup \{e, M[e]\}) \setminus \text{Expr}_x \\
  [M[e_1] = e_2;] \# A &= (A \cup \{e_1, e_2\}) \setminus \text{Loads}
  \end{align*}$$
Values of Variables:

- Extend the set $\textit{Expr}$ of expressions by occurring loads $M[e]$.

- Extend the Effects of Edges:

\[
[x = M[e];] \triangleright V e' = \begin{cases} 
\{x\} & \text{if } e' = M[e] \\
\emptyset & \text{if } e' = e \\
V e' \setminus \{x\} & \text{otherwise}
\end{cases}
\]

\[
[M[e_1] = e_2;] \triangleright V e' = \begin{cases} 
\emptyset & \text{if } e' \in \{e_1, e_2\} \\
V e' & \text{otherwise}
\end{cases}
\]
(3) **Constant Propagation:**

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

$$\llbracket x = M[e]; \rrbracket^\# (D, M) = \begin{cases} 
(D \oplus \{ x \mapsto M a \}, M) & \text{if } \llbracket e \rrbracket^\# D = a \sqsubseteq \top \\
(D \oplus \{ x \mapsto \top \}, M) & \text{otherwise} \\
(D, M \oplus \{ a \mapsto \llbracket e_2 \rrbracket^\# D \}) & \text{if } \llbracket e_1 \rrbracket^\# D = a \sqsubseteq \top \\
(D, \bot) & \text{otherwise}
\end{cases}$$

$$\llbracket M[e_1] = e_2; \rrbracket^\# (D, M) = \begin{cases} 
(D, \bot) & \text{if } \llbracket e_1 \rrbracket^\# D = a \sqsubseteq \top \\
(a \in \mathbb{N}) & \text{otherwise}
\end{cases}$$

$$\bot a = \top \quad (a \in \mathbb{N})$$
Problems:

• Addresses are from \( \mathbb{N} \) :-(
  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 \( M \) :-(

\[ \implies \text{constant propagation fails} \ :-( \]
\[ \implies \text{memory accesses/pointers kill precision} \ :-( \]
Simplification:

- We consider pointers to the beginning of blocks $A$ which allow indexed accesses $A[i] \text{ :)}
- 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  
  $y[e_1] = e_2;$  // indexed write access to a block

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

\[
x = \text{new}();
\]
\[
y = \text{new}();
\]
\[
x[0] = y;
\]
\[
y[1] = 7;
\]
The Semantics:

$x$

$y$
The Semantics:
The Semantics:
The Semantics:
The Semantics:
More Complex Example:

```java
r = Null;
while (t ≠ Null) {
    h = t;
    t = t[0];
    h[0] = r;
    r = h;
}
```

Diagram:
```
0 ---------> 1
    |          |
    |          |
Neg(t ≠ Null)  Pos(t ≠ Null)
    |          |
    |          |
  7 ---------> 2
    |          |
    |          |
    h = t;
  3 ---------> 4
    |          |
    |          |
    t = t[0];
  5 ---------> 6
    |          |
    |          |
    h[0] = r;
```
Concrete Semantics:

A store consists of a finite collection of blocks.

After $h$ new-operations we obtain:

\[
\begin{align*}
    Addr_h &= \{ \text{ref } a \mid 0 \leq a < h \} & \text{addresses} \\
    Val_h &= Addr_h \cup \mathbb{Z} & \text{values} \\
    Store_h &= (Addr_h \times \mathbb{N}_0) \to Val_h & \text{store} \\
    State_h &= (Vars \to Val_h) \times Store_h & \text{states}
\end{align*}
\]

For simplicity, we set: $0 = \text{Null}$