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 \( \sqcap \) 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:

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

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

- Extend the Effects of Edges:

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

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

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

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

\[
\overline{\bot a} = \top \quad (a \in \mathbb{N})
\]

\[\mathbb{N}\]
Problems:

- Addresses are from \( \mathbb{N} \) \( \Rightarrow \) constant propagation fails \( \Rightarrow \) memory accesses/pointers kill precision
- There are no infinite strictly ascending chains, but ...
- Exact addresses at compile-time are rarely known \( \Rightarrow \)
- At the same program point, typically different addresses are accessed ...
- Storing at an unknown address destroys all information \( M \) \( \Rightarrow \)
Simplification:

- We consider pointers to the beginning of blocks \( A \) which allow indexed accesses \( A[i] \).
- We ignore well-typedness of the blocks.
- New statements:
  
  \[
  x = \text{new}(); \quad // \quad \text{allocation of a new block}
  
  x = y[e]; \quad // \quad \text{indexed read access to a block}
  
  y[e_1] = e_2; \quad // \quad \text{indexed write access to a block}
  
- Blocks are possibly infinite \(:-)
- For simplicity, all pointers point to the beginning of a block.
Simple Example:

\begin{align*}
x &= \text{new}(); \\
y &= \text{new}(); \\
x[0] &= y; \\
y[1] &= 7;
\end{align*}
The Semantics:

$x$

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

\[ r = \text{Null}; \]

while \((t \neq \text{Null})\) {
    \[ h = t; \]
    \[ t = t[0]; \]
    \[ h[0] = r; \]
    \[ r = h; \]
}\n
\[ r = \text{Null}; \]
Concrete Semantics:

A store consists of a finite collection of blocks.

After \( h \) new-operations we obtain:

\[
\begin{align*}
\text{Addr}_h &= \{ \text{ref } a \mid 0 \leq a < h \} \quad \text{// addresses} \\
\text{Val}_h &= \text{Addr}_h \cup \mathbb{Z} \quad \text{// values} \\
\text{Store}_h &= (\text{Addr}_h \times \mathbb{N}_0) \to \text{Val}_h \quad \text{// store} \\
\text{State}_h &= (\text{Vars} \to \text{Val}_h) \times \text{Store}_h \quad \text{// states}
\end{align*}
\]

For simplicity, we set: \( 0 = \text{Null} \)
Let \((\rho, \mu) \in State_h\). Then we obtain for the new edges:

\[
\begin{align*}
[x = \text{new}()] & (\rho, \mu) = (\rho \oplus \{x \mapsto \text{ref } h\}, \\
\mu \oplus \{(\text{ref } h, i) \mapsto 0 \mid i \in \mathbb{N}_0\}) \\
[x = y[e];] & (\rho, \mu) = (\rho \oplus \{x \mapsto \mu(\rho \cdot y, [e] \rho)\}, \mu) \\
[y[e_1] = e_2;] & (\rho, \mu) = (\rho, \mu \oplus \{(\rho \cdot y, [e_1] \rho) \mapsto [e_2] \rho\})
\end{align*}
\]
Caveat:

This semantics is too detailed in that it computes with absolute Addresses. Accordingly, the two programs:

\[
\begin{align*}
x &= \text{new}(); \\
y &= \text{new}(); \\
y &= \text{new}(); \\
x &= \text{new}();
\end{align*}
\]

are not considered as equivalent !!?

Possible Solution:

Define equivalence only up to permutation of addresses  :-/
Alias Analysis  1. Idea:

- Distinguish finitely many classes of blocks.
- Collect all addresses of a block into one set!
- Use sets of addresses as abstract values!

\[ \Rightarrow \text{Points-to-Analysis} \]

\[
\begin{align*}
\text{Addr}^\# &= \text{Edges} & \text{// creation edges} \\
\text{Val}^\# &= 2^{\text{Addr}^\#} & \text{// abstract values} \\
\text{Store}^\# &= \text{Addr}^\# \rightarrow \text{Val}^\# & \text{// abstract store} \\
\text{State}^\# &= (\text{Vars} \rightarrow \text{Val}^\#) \times \text{Store}^\# & \text{// abstract states}
\end{align*}
\]

// complete lattice !!!
... in the Simple Example:

\[
\begin{align*}
0 &: \quad x = \text{new}(); \\
1 &: \quad y = \text{new}(); \\
2 &: \quad x[0] = y; \\
3 &: \quad y[1] = 7;
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>(x)</th>
<th>(y)</th>
<th>((0, 1))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>1</td>
<td>({(0, 1)})</td>
<td>(\emptyset)</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>2</td>
<td>({(0, 1)})</td>
<td>({(1, 2)})</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>3</td>
<td>({(0, 1)})</td>
<td>({(1, 2)})</td>
<td>({(1, 2)})</td>
</tr>
<tr>
<td>4</td>
<td>({(0, 1)})</td>
<td>({(1, 2)})</td>
<td>({(1, 2)})</td>
</tr>
</tbody>
</table>
The Effects of Edges:

\[
\begin{align*}
\llbracket (\_, \;, \_ ) \rrbracket^\# (D, M) &= (D, M) \\
\llbracket (\_, \text{Pos}(e), \_ ) \rrbracket^\# (D, M) &= (D, M) \\
\llbracket (\_, x = y; \, \_ ) \rrbracket^\# (D, M) &= (D \oplus \{x \mapsto D y\}, M) \\
\llbracket (\_, x = e; \, \_ ) \rrbracket^\# (D, M) &= (D \oplus \{x \mapsto \emptyset\}, M), \quad e \notin \text{Vars} \\
\llbracket (u, x = \text{new}(); \, v) \rrbracket^\# (D, M) &= (D \oplus \{x \mapsto \{(u, v)\}\}, M) \\
\llbracket (\_, x = y[e]; \, \_ ) \rrbracket^\# (D, M) &= (D \oplus \{x \mapsto \bigcup \{M(f) \mid f \in D y\}\}, M) \\
\llbracket (\_, y[e_1] = x; \, \_ ) \rrbracket^\# (D, M) &= (D, M \oplus \{f \mapsto (M f \cup D x) \mid f \in D y\})
\end{align*}
\]
Caveat:

- The value `Null` has been ignored. Dereferencing of `Null` or negative indices are not detected :-(

- **Destructive updates** are only possible for variables, not for blocks in storage!
  
  ➞ no information, if not all block entries are initialized before use :-((

- The effects now depend on the edge itself.
  
  The analysis cannot be proven correct w.r.t. the reference semantics :-(

In order to prove correctness, we first **instrument** the concrete semantics with extra information which records where a block has been created.
We compute possible points-to information.
From that, we can extract may-alias information.
The analysis can be rather expensive — without finding very much :-(
Separate information for each program point can perhaps be abandoned ??
Alias Analysis

2. Idea:

Compute for each variable and address a value which safely approximates the values at every program point simultaneously!

... in the Simple Example:

```
0  x = new();
1  y = new();
2  x[0] = y;
3  y[1] = 7;
4
```

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 1)</td>
<td>{(0, 1)}</td>
<td></td>
</tr>
<tr>
<td>(1, 2)</td>
<td>{(1, 2)}</td>
<td></td>
</tr>
</tbody>
</table>
Each edge \((u, lab, v)\) gives rise to constraints:

<table>
<thead>
<tr>
<th>lab</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x = y;)</td>
<td>(\mathcal{P}[x] \supseteq \mathcal{P}[y])</td>
</tr>
<tr>
<td>(x = \text{new}();)</td>
<td>(\mathcal{P}[x] \supseteq {(u,v)})</td>
</tr>
<tr>
<td>(x = y[e];)</td>
<td>(\mathcal{P}[x] \supseteq \bigcup{\mathcal{P}[f] \mid f \in \mathcal{P}[y]})</td>
</tr>
<tr>
<td>(y[e_1] = x;)</td>
<td>(\mathcal{P}[f] \supseteq (f \in \mathcal{P}[y]) \ ? \mathcal{P}[x] : \emptyset) for all (f \in \text{Addr}^#)</td>
</tr>
</tbody>
</table>

Other edges have no effect  :-)

382