Let $(\rho, \mu) \in State_h$. Then we obtain for the new edges:

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

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  :-)

374
Alias Analysis 1. Idea:

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

\implies \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}^\# \to \text{Val}^\# & \text{abstract store} \\
\text{State}^\# &= (\text{Vars} \to \text{Val}^\#) \times \text{Store}^\# & \text{abstract states} \\
\end{align*}

\[
\text{complete lattice} \!!!
\]
... in the Simple Example:

- $y[1] = 7$
- $x[0] = y$

<table>
<thead>
<tr>
<th></th>
<th>$x$</th>
<th>$y$</th>
<th>$(0, 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\top$</td>
<td>$\top$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>1</td>
<td>${(0, 1)}$</td>
<td>$\top$</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:

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

- 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!

  \[\Rightarrow\] 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:

```
x[0] = y;
y[0] = new();
```

<table>
<thead>
<tr>
<th></th>
<th><code>x</code></th>
<th><code>{(0,1)}</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>y</code></td>
<td><code>{(1,2)}</code></td>
</tr>
<tr>
<td></td>
<td><code>(0,1)</code></td>
<td><code>{(1,2)}</code></td>
</tr>
<tr>
<td></td>
<td><code>(1,2)</code></td>
<td><code>∅</code></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>(P[x] \supseteq P[y])</td>
</tr>
<tr>
<td>(x = \text{new}();)</td>
<td>(P[x] \supseteq {(u, v)})</td>
</tr>
<tr>
<td>(x = y[e];)</td>
<td>(P[x] \supseteq \bigcup{P[f] \mid f \in P[y]})</td>
</tr>
<tr>
<td>(y[e_1] = x;)</td>
<td>(P[f] \supseteq (f \in P[y]) \cup P[x] : \emptyset)</td>
</tr>
<tr>
<td></td>
<td>for all (f \in \text{Addr}^#)</td>
</tr>
</tbody>
</table>

Other edges have no effect \(:-)\)
Discussion:

- The resulting constraint system has size $\mathcal{O}(k \cdot n)$ for $k$ abstract addresses and $n$ edges :-(
- The number of necessary iterations is $\mathcal{O}(k + \#\text{Vars})$ ... 
- The computed information is perhaps still too zuo precise !!?
- In order to prove correctness of a solution $s^\# \in States^\#$ we show:

![Diagram with states and transitions]

\( s \xrightarrow{[k]} s_1 \)
\( s^\# \xRightarrow{\Delta} \)
\( s^\# \xRightarrow{\Delta} \)
Alias Analysis

3. Idea:

Determine one equivalence relation \( \equiv \) on variables \( x \) and memory accesses \( y[\ ] \) with \( s_1 \equiv s_2 \) whenever \( s_1, s_2 \) may contain the same address at some \( u_1, u_2 \)

... in the Simple Example:

\[
\begin{align*}
0 & \quad x = \text{new}(); \\
1 & \quad y = \text{new}(); \\
2 & \quad x[0] = y; \\
3 & \quad y[1] = 7; \\
4 &
\end{align*}
\]

\[
\equiv = \{ \{x\}, \{y, x[\ ]\}, \{y[\ ]\}\}
\]
Discussion:

→ We compute a single information for the whole program.
→ The computation of this information maintains partitions
\[ \pi = \{P_1, \ldots, P_m\} \]
→ Individual sets \( P_i \) are identified by means of representatives \( p_i \in P_i \).
→ The operations on a partition \( \pi \) are:

\[
\text{find}(\pi, p) = p_i \quad \text{if} \quad p \in P_i \\
\text{// returns the representative}
\]

\[
\text{union}(\pi, p_{i_1}, p_{i_2}) = \{P_{i_1} \cup P_{i_2}\} \cup \{P_j \mid i_1 \neq j \neq i_2\} \\
\text{// unions the represented classes}
\]
→ If $x_1, x_2 \in Vars$ are equivalent, then also $x_1[\,]$ and $x_2[\,]$ must be equivalent :-)

→ If $P_i \cap Vars \neq \emptyset$, then we choose $p_i \in Vars$. Then we can apply $union$ recursively:

\[
union^*(\pi, q_1, q_2) = \begin{align*}
& \text{let } p_{i_1} = \text{find}(\pi, q_1) \\
& p_{i_2} = \text{find}(\pi, q_2) \\
& \text{in } \begin{cases} 
& \text{if } p_{i_1} == p_{i_2} \text{ then } \pi \\
& \text{else let } \pi = \text{union}(\pi, p_{i_1}, p_{i_2}) \\
& \text{in } \begin{cases} 
& \text{if } p_{i_1}, p_{i_2} \in Vars \text{ then } \\
& \text{union}^*(\pi, p_{i_1}[\,], p_{i_2}[\,]) 
\end{cases}
\end{cases}
\end{align*}
\]
The analysis iterates over all edges **once**:

\[
\pi = \{ \{ x \}, \{ x[ \] \} \mid x \in \text{Vars} \};
\]

forall \( k = (\_, \text{lab}, \_ \) do \( \pi = [\text{lab}]^\# \pi; \)

where:

\[
[x = y;]^\# \pi = \text{union}^* (\pi, x, y)
\]

\[
[x = y[e];]^\# \pi = \text{union}^* (\pi, x, y[ \])
\]

\[
[y[e] = x;]^\# \pi = \text{union}^* (\pi, x, y[ \])
\]

\[
[\text{lab}]^\# \pi = \pi \quad \text{otherwise}
\]
... in the Simple Example:

0  
  x = new();

1  
  y = new();

2  
  x[0] = y;

3  
  y[1] = 7;

4

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 1)</td>
<td>{{x}, {y}, {x[]}, {y[]}}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(1, 2)</td>
<td>{{x}, {y}, {x[]}, {y[]}}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2, 3)</td>
<td>{{x}, {y, x[]}, {y[]}}</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3, 4)</td>
<td>{{x}, {y, x[]}, {y[]}}</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
... in the More Complex Example:

Neg(t ≠ Null) → 0
  r = Null;

Pos(t ≠ Null) → 1

2 → 2
  h = t;

3 → 3
  t = t[0];

4 → 4
  h[0] = r;

5 → 5
  r = h;

6

<table>
<thead>
<tr>
<th></th>
<th>{{h}, {r}, {t}, {h[]}, {t[]}}</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2,3)</td>
<td>{{h, t}, {r}, {h[], t[]}}</td>
</tr>
<tr>
<td>(3,4)</td>
<td>{{h, t, h[], t[]}, {r}}</td>
</tr>
<tr>
<td>(4,5)</td>
<td>{{h, t, r, h[], t[]}}</td>
</tr>
<tr>
<td>(5,6)</td>
<td>{{h, t, r, h[], t[]}}</td>
</tr>
</tbody>
</table>
Warning:
In order to find something, we must assume that variables / addresses always receive a value before they are accessed.

Complexity:
we have:
\[
O(\# edges + \# Vars) \quad \text{calls of union}^\ast \\
O(\# edges + \# Vars) \quad \text{calls of find} \\
O(\# Vars) \quad \text{calls of union}
\]

⇒ We require efficient Union-Find data-structure :-)}

389