Discussion:

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

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

\[
\begin{align*}
\text{find} (\pi, p) &= p_i \quad \text{if } p \in P_i \\
\text{union} (\pi, p_{i_1}, p_{i_2}) &= \{ P_{i_1} \cup P_{i_2} \} \cup \{ P_j \mid i_1 \neq j \neq i_2 \} \\
\end{align*}
\]

// returns the representative

// 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 \( \text{union} \) recursively:

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

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

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

where:

\[
\begin{align*}
[x = y;]^\# \pi &= \text{union}^* (\pi, x, y) \\
[x = y[e];]^\# \pi &= \text{union}^* (\pi, x, y[\]) \\
y[e] = x;]^\# \pi &= \text{union}^* (\pi, x, y[\]) \\
[\text{lab}]^\# \pi &= \pi \quad \text{otherwise}
\end{align*}
\]
... in the Simple Example:

```plaintext
x = new();
y = new();
x[0] = y;
y[1] = 7;
```

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

```
r = Null;
Pos(t ≠ Null)
Neg(t ≠ Null)
```

![Diagram showing the flow of the example]

| (2, 3) | \{\{h, t\}, \{r\}, \{h[], t[]\} \} |
| (3, 4) | \{\{h, t, h[], t[]\}, \{r\} \} |
| (4, 5) | \{\{h, t, r, h[], t[]\} \} |
| (5, 6) | \{\{h, t, r, h[], t[]\} \} |
Caveat:
In order to find something, we must assume that variables / addresses always receive a value before they are accessed.

Complexity:
we havve:

\[ O(\# \text{edges} + \# \text{Vars}) \text{ calls of union}^* \]
\[ O(\# \text{edges} + \# \text{Vars}) \text{ calls of find} \]
\[ O(\# \text{Vars}) \text{ calls of union} \]

\[ \Rightarrow \text{ We require efficient Union-Find data-structure} \quad :-) \]
Idea:

Represent partition of $U$ as directed forest:

- For $u \in U$ a reference $F[u]$ to the father is maintained;
- Roots are elements $u$ with $F[u] = u$.

Single trees represent equivalence classes.

Their roots are their representatives ...
→ find $(\pi, u)$ follows the father references

→ union $(\pi, u_1, u_2)$ re-directs the father reference of one $u_i$...
The Costs:

\[
\begin{align*}
\text{union} & : \mathcal{O}(1) : -) \\
\text{find} & : \mathcal{O}(\text{depth}(\pi)) : (-
\end{align*}
\]

Strategy to Avoid Deep Trees:

- Put the smaller tree below the bigger !
- Use find to compress paths ...
Robert Endre Tarjan, Princeton
Note:

- By this data-structure, \( n \) union- und \( m \) find operations require time \( \mathcal{O}(n + m \cdot \alpha(n, n)) \)
  \[
  // \quad \alpha \text{ the inverse Ackermann-function} \quad :)\]
- For our application, we only must modify union such that roots are from \( Vars \) whenever possible.
- This modification does not increase the asymptotic run-time.  

Summary:

The analysis is extremely fast — but may not find very much.
Background 3: Fixpoint Algorithms

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

Observation:

RR-Iteration is inefficient:

\[ \rightarrow \] We require a complete round in order to detect termination \ :-(

\[ \rightarrow \] If in some round, the value of just one unknown is changed, then we still re-compute all \ :-(

\[ \rightarrow \] The practical run-time depends on the ordering on the variables \ :-(