(3) Constant Propagation:

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

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

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

\[
\neg a = \top \quad (a \in N)
\]

\[
M = \bigoplus_{i \in N} \alpha_i \mapsto \alpha_{i_1}^1 \ldots \alpha_{i_k}^k
\]

Problems:

- Addresses are from \( N \) :-(
  - 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 \( M :-( \)

\[\quad \text{constant propagation fails :-(} \]

\[\quad \text{memory accesses/pointers kill precision :-(} \]

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:
  - \( x = \text{new}(); \quad // \) allocation of a new block
  - \( x = y[e]; \quad // \) indexed read access to a block
  - \( y[e_1] = e_2; \quad // \) indexed write access to a block

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

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

The Semantics:

```

```

More Complex Example:

```plaintext
r = Null;
while (t != Null) {
    h = t;
    t = t[0];
    h[0] = r;
    r = h;
}
```

```

```

```plaintext
Neg(t != Null)

```

```

```

```plaintext
Pos(t != Null)

```

```

```

```plaintext
r = h;
```

```

```

```plaintext
h = t;
```

```

```

```plaintext
r = h;
```

```

```

```plaintext
h[0] = r;
```

```

```

```plaintext
r = h;
```

```

```

```plaintext
r = h;
```

```

```

```

```plaintext
r = h;
```

```

```

```plaintext
r = h;
```

```

```

```plaintext
r = h;
```

```

```

```plaintext
r = h;
```

More Complex Example:

```plaintext
r = \textbf{Null};
while (t \neq \textbf{Null}) {
    h = t;
    t = t[0];
    h[0] = r;
    r = h;
}
```

Concrete Semantics:

A store consists of a finite collection of blocks.
After \( h \) new-operations we obtain:

- \( \text{Addr}_h = \{ \text{ref } a \mid 0 \leq a < h \} \) \quad // addresses
- \( \text{Val}_h = \text{Addr}_h \cup \mathbb{Z} \) \quad // values
- \( \text{Store}_h = (\text{Addr}_h \times \mathbb{N}_h) \rightarrow \text{Val}_h \) \quad // store
- \( \text{State}_h = (\text{Vars} \rightarrow \text{Val}_h) \times \text{Store}_h \) \quad // states

For simplicity, we set: \( 0 = \textbf{Null} \)

Let \( (\rho, \mu) \in \text{State}_h \). Then we obtain for the new edges:

- \( [x = \text{new()}] (\rho, \mu) = (\rho \oplus \{ x \mapsto \text{ref } h \}, \mu \oplus \{ (\text{ref } h, i) \mapsto 0 \mid i \in \mathbb{N}_h \}) \)
- \( [x = y[e_1]] (\rho, \mu) = (\rho \oplus \{ x \mapsto \mu \oplus \{ (\text{ref } h, i) \mapsto [e_1] \rho \} \}, \mu) \)
- \( [y[e_1] = e_2] (\rho, \mu) = (\rho, \mu \oplus \{ (\rho y, [e_1] \rho) \mapsto [e_2] \rho \}) \)
Concrete Semantics:

A store consists of a finite collection of blocks. After \( h \) now-operations we obtain:

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

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

Let \( (\rho, \mu) \in \text{State}_h \). Then we obtain for the new edges:

\[
\begin{align*}
[x = \text{new()}] \ (\rho, \mu) &= (\rho \oplus \{ x \rightarrow \text{ref} \ h \}, \\
&\quad \mu \oplus \{ (\text{ref} \ h, i) \rightarrow (A \in \mathbb{N}_h) \}) \\
[y[e_1]] \ (\rho, \mu) &= (\rho \oplus \{ x \rightarrow \mu \ (\rho y, [e_1] \rho) \}, \mu) \\
[y[e_1] = e_2] \ (\rho, \mu) &= (\rho, \mu \oplus \{ (\rho y, [e_1] \rho) \rightarrow [e_2] \rho \})
\end{align*}
\]

Caveat:

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

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

are not considered as equivalent!!?

Possible Solution:

Define equivalence only up to permutation of addresses so...
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!

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

\[\begin{align*}
\text{Addr}^2 &= \text{Edges} & \text{creation edges} \\
\text{Val}^2 &= 2^{\text{Addr}^4} & \text{abstract values} \\
\text{Store}^4 &= \text{Addr}^4 \rightarrow \text{Val}^2 & \text{abstract store} \\
\text{State}^4 &= (\text{Vars} \rightarrow \text{Val}^2) \times \text{Store}^4 & \text{abstract states}
\end{align*}\]

\[\text{// complete lattice !!!}\]

... in the Simple Example:

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>(0, 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>{(0, 1)}</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>{(0, 1)}</td>
<td>{(1, 2)}</td>
<td>0</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:

\[
\begin{align*}
[\langle\cdot,\cdot\rangle]^2(D, M) &= (D, M) \\
[\langle\cdot, \text{Pos}(\cdot)\rangle]^2(D, M) &= (D, M) \\
[\langle\cdot, x = y;\rangle]^2(D, M) &= (D \oplus \{x \mapsto D y\}, M) \\
[\langle\cdot, x = e;\rangle]^2(D, M) &= (D \oplus \{x \mapsto \emptyset\}, M), \quad e \notin \text{Vars} \\
[\langle\cdot, x = \text{new}();\rangle]^2(D, M) &= (D \oplus \{x \mapsto \{(u, v)\}\}, M) \\
[\langle\cdot, y[c];\rangle]^2(D, M) &= (D \oplus \{x \mapsto \bigcup\{M(f) \mid f \in D y\}\}, M) \\
[\langle\cdot, y[e_i] = x;\rangle]^2(D, M) &= (D, M \oplus \{f \mapsto (M f \cup D x) \mid f \in D y\})
\end{align*}
\]

\[\text{Caveat:}\]

- The value \textbf{Null} has been ignored. Dereferencing of \textbf{Null} or negative indices are not detected \(:(\)
- \textbf{Destructive updates} are only possible for variables, not for blocks in storage!

\[\text{// 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 \textbf{instrument} the concrete semantics with extra information which records where a block has been created.
The Effects of Edges:

\[
\begin{align*}
[\text{\_, \_, \_}]^1(D, M) & = (D, M) \\
[\text{\_, Pos(e), \_}]^1(D, M) & = (D, M) \\
[\text{\_, x = y; \_}]^1(D, M) & = (D \oplus \{x \mapsto D \_y\}, M) \\
[\text{\_, x = e; \_}]^1(D, M) & = (D \oplus \{x \mapsto e\}, M), \quad e \notin Vars \\
[\text{\_, x = new(\_); e}]^1(D, M) & = (D \oplus \{x \mapsto \{(u, v)\}\}, M) \\
[\text{\_, x = y(c); \_}]^1(D, M) & = (D \oplus \{x \mapsto M(f) \cup \{f \in D \_y\}\}, M) \\
[\text{\_, y(e) = x; \_}]^1(D, M) & = (D, M \oplus \{f \mapsto (M \_f \cup D \_x) \cup \{f \in D \_y\}\})
\end{align*}
\]

... 
- 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: z[0] = y;
   3: y[1] = 7;
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0, 1}</td>
<td>{1, 2}</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[c];)</td>
<td>(P[x] \supseteq \bigcup {P[f] \mid f \in P[y]})</td>
</tr>
</tbody>
</table>
| \(y[c_1] = x;\) | \(P[f] \supseteq \{f \in P[y] \mid P[x] : \emptyset\}\)  
|          | for all \(f \in \text{Addr}^3\) |

Other edges have no effect :-(

Caveat:

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

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

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

are not considered as equivalent !?!

Possible Solution:

Define equivalence only up to permutation of addresses :-)
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]) \land \mathcal{P}[x] : \emptyset) for all (f \in \text{Addr})</td>
</tr>
</tbody>
</table>

Other edges have no effect :)
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 coarse!
- In order to prove correctness of a solution $s^t \in States^t$ we show:

```
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:

\[
\begin{align*}
\text{find} (\pi, p) &= p_i & \text{if } p \in P_i \\
\text{union} (\pi, p_1, p_2) &= \{ P_{i_1} \cup P_{i_2} \cup \{ P_j \mid i \neq j \neq i \} \} & \text{unions the represented classes}
\end{align*}
\]
```

383

```
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:

\[
\begin{align*}
\text{find} (\pi, p) &= p_i & \text{if } p \in P_i \\
\text{union} (\pi, p_1, p_2) &= \{ P_{i_1} \cup P_{i_2} \cup \{ P_j \mid i \neq j \neq i \} \} & \text{unions the represented classes}
\end{align*}
\]
```

385
\[ \text{union}^*(\pi, q_1, q_2) = \begin{cases} p_{q_2} = \text{find}(\pi, q_1) \\ p_{q_2} = \text{find}(\pi, q_2) \\ \text{if } p_{q_1} == p_{q_2} \text{ then } \pi \\ \text{else let } \pi = \text{union}(\pi, p_{q_1}, p_{q_2}) \\ \text{if } p_{q_1}, p_{q_2} \in \text{Vars} \text{ then } \text{union}^*(\pi, p_{q_1}, p_{q_2}) \end{cases} \]

The analysis iterates over all edges once:

\[ \pi = \{ \{ x \}, \{ x[0] \} \mid x \in \text{Vars} \}; \]

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

where:

\[ [x = y]^2 \pi = \text{union}^*(\pi, \pi, y) \]
\[ [x = y[0]]^2 \pi = \text{union}^*(\pi, \pi, y[0]) \]
\[ [y[0] = x]^2 \pi = \text{union}^*(\pi, \pi, y[0]) \]
\[ [\text{lab}]^2 \pi = \pi \text{ otherwise} \]

... in the Simple Example:

\[
\begin{array}{c|c}
\text{r = new();} & \{ \{ x \}, \{ y \}, \{ x[0] \}, \{ y[0] \} \} \\
\text{g = new();} & \{ \{ x \}, \{ y \}, \{ x[0] \}, \{ y[0] \} \} \\
\text{x[0] = g;} & \{ \{ x \}, \{ y \}, \{ x[0] \}, \{ y[0] \} \} \\
\text{y[1] = r;} & \{ \{ x \}, \{ y \}, \{ x[0] \}, \{ y[0] \} \}
\end{array}
\]

... in the More Complex Example:

\[
\begin{array}{c|c}
\{ \{ h \}, \{ r \}, \{ t \}, \{ h[0] \}, \{ t[0] \} \} & \{ \{ h \}, \{ r \}, \{ t \}, \{ h[0] \}, \{ t[0] \} \} \\
(2, 3) & \{ \{ h, t \}, \{ r \}, \{ h[0] \}, \{ t[0] \} \}
\end{array}
\]