Background 1: An Operational Semantics

we choose a small-step operational approach.
Programs are represented as control-flow graphs.
In the example:

\[ A_1 = A_0 + 1 \times i; \]
\[ R_1 = M[A_1]; \]
\[ A_2 = A_0 + 1 \times j; \]
\[ R_2 = M[A_2]; \]
\[ A_3 = A_0 + 1 \times j; \]
Thereby, represent:

<table>
<thead>
<tr>
<th>vertex</th>
<th>program point</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>programm start</td>
</tr>
<tr>
<td>stop</td>
<td>program exit</td>
</tr>
<tr>
<td>edge</td>
<td>step of computation</td>
</tr>
</tbody>
</table>
Thereby, represent:

<table>
<thead>
<tr>
<th>vertex</th>
<th>program point</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>programm start</td>
</tr>
<tr>
<td>stop</td>
<td>program exit</td>
</tr>
<tr>
<td>edge</td>
<td>step of computation</td>
</tr>
</tbody>
</table>

**Edge Labelings:**

- **Test**: Pos \( (e) \) or Neg \( (e) \)
- **Assignment**: \( R = e; \)
- **Load**: \( R = M[e]; \)
- **Store**: \( M[e_1] = e_2; \)
- **Nop**: \( ; \)
Computations follow paths.

Computations transform the current state

\[ s = (\rho, \mu) \]

where:

<table>
<thead>
<tr>
<th>( \rho : Vars \rightarrow \text{int} )</th>
<th>contents of registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu : \mathbb{N} \rightarrow \text{int} )</td>
<td>contents of storage</td>
</tr>
</tbody>
</table>

Every edge \( k = (u, lab, v) \) defines a partial transformation

\[ [k] = [lab] \]

of the state:
\[
\begin{align*}
\llbracket;\rrbracket (\rho, \mu) &= (\rho, \mu) \\
\llbracket \text{Pos}(e) \rrbracket (\rho, \mu) &= (\rho, \mu) \quad \text{if } \llbracket e \rrbracket \rho \neq 0 \\
\llbracket \text{Neg}(e) \rrbracket (\rho, \mu) &= (\rho, \mu) \quad \text{if } \llbracket e \rrbracket \rho = 0
\end{align*}
\]
\[
\left[ ; \right] (\rho, \mu) = (\rho, \mu)
\]

\[
\left[ \text{Pos} (e) \right] (\rho, \mu) = (\rho, \mu) \quad \text{if } \left[ e \right] \rho \neq 0
\]

\[
\left[ \text{Neg} (e) \right] (\rho, \mu) = (\rho, \mu) \quad \text{if } \left[ e \right] \rho = 0
\]

// [e] : evaluation of the expression e, e.g.

// \[ x + y \] \{ x \mapsto 7, y \mapsto -1 \} = 6

// \[ ! (x == 4) \] \{ x \mapsto 5 \} = 1
\[ ; \] (\(\rho, \mu\)) = (\(\rho, \mu\))

\[\text{Pos}(e)\] (\(\rho, \mu\)) = (\(\rho, \mu\)) \quad \text{if} \ [e] \rho \neq 0

\[\text{Neg}(e)\] (\(\rho, \mu\)) = (\(\rho, \mu\)) \quad \text{if} \ [e] \rho = 0

// \([e]\) : evaluation of the expression e, e.g.

// \([x + y]\) \{x \mapsto 7, y \mapsto -1\} = 6

// \([!(x == 4)]\) \{x \mapsto 5\} = 1

\[[R = e;]\] (\(\rho, \mu\)) = (\(\rho \oplus \{R \mapsto [e] \rho\}, \mu\))

// where “\(\oplus\)” modifies a mapping at a given argument
$$\llbracket R = M[e]; \rrbracket (\rho, \mu) = (\rho \oplus \{ R \mapsto \mu(\llbracket e \rrbracket \rho) \}, \mu)$$

$$\llbracket M[e_1] = e_2; \rrbracket (\rho, \mu) = (\rho, \mu \oplus \{ [e_1] \rho \mapsto [e_2] \rho \})$$

Example:

$$\llbracket x = x + 1; \rrbracket (\{ x \mapsto 5 \}, \mu) = (\rho, \mu)$$ where:

$$\rho = \{ x \mapsto 5 \} \oplus \{ x \mapsto \llbracket x + 1 \rrbracket \{ x \mapsto 5 \} \}$$

$$= \{ x \mapsto 5 \} \oplus \{ x \mapsto 6 \}$$

$$= \{ x \mapsto 6 \}$$
A path \( \pi = k_1 k_2 \ldots k_m \) is a computation for the state \( s \) if:

\[
s \in \text{def} \left( [[k_m] \circ \ldots \circ [k_1]] \right)
\]

The result of the computation is:

\[
[[\pi]] s = ([[k_m] \circ \ldots \circ [k_1]]) s
\]

**Application:**

Assume that we have computed the value of \( x + y \) at program point \( u \):

![Diagram](x+y \quad \pi \quad v)

We perform a computation along path \( \pi \) and reach \( v \) where we evaluate again \( x + y \) ...
Idea:

If $x$ and $y$ have not been modified in $\pi$, then evaluation of $x + y$ at $v$ must return the same value as evaluation at $u$ :-)

We can check this property at every edge in $\pi$ :-}
Idea:

If $x$ and $y$ have not been modified in $\pi$, then evaluation of $x + y$ at $v$ must return the same value as evaluation at $u$ :-)

We can check this property at every edge in $\pi$ :-)

More generally:

Assume that the values of the expressions $A = \{e_1, \ldots, e_r\}$ are available at $u$. 
Idea:

If $x$ and $y$ have not been modified in $\pi$, then evaluation of $x + y$ at $v$ must return the same value as evaluation at $u$ :-)

We can check this property at every edge in $\pi$ :-}

More generally:

Assume that the values of the expressions $A = \{e_1, \ldots, e_r\}$ are available at $u$.

Every edge $k$ transforms this set into a set $[[k]# A$ of expressions whose values are available after execution of $k$ ...
... which transformations can be composed to the effect of a path $\pi = k_1 \ldots k_r$:

$$[\pi]^{\#} = [k_r]^{\#} \circ \ldots \circ [k_1]^{\#}$$
... which transformations can be composed to the effect of a path 
\( \pi = k_1 \ldots k_r \):

\[
[\pi]^\# = [k_r]^\# \circ \ldots \circ [k_1]^\#
\]

The effect \( [k]^\# \) of an edge \( k = (u, \text{lab}, v) \) only depends on
the label \( \text{lab} \), i.e., \( [k]^\# = [\text{lab}]^\# \)
... which transformations can be composed to the effect of a path $\pi = k_1 \ldots k_r$:

$$\left[\pi\right]^{\#} = \left[k_r\right]^{\#} \circ \ldots \circ \left[k_1\right]^{\#}$$

The effect $\left[k\right]^{\#}$ of an edge $k = (u, lab, v)$ only depends on the label $lab$, i.e., $\left[k\right]^{\#} = \left[lab\right]^{\#}$ where:

$$\left[;\right]^{\#} A = A$$
$$\left[\text{Pos}(e)\right]^{\#} A = \left[\text{Neg}(e)\right]^{\#} A = A \cup \{e\}$$
$$\left[x = e;\right]^{\#} A = (A \cup \{e\}) \setminus \text{Expr}_x$$

where $\text{Expr}_x$ all expressions which contain $x$
\begin{align*}
\llbracket x = M[e]; \rrbracket_A & \not\in A = (A \cup \{e\}) \setminus \mathit{Expr}_x \\
\llbracket M[e_1] = e_2; \rrbracket_A & = A \cup \{e_1, e_2\}
\end{align*}
\[ [x = M[e];] \triangleright^{\sharp} A \quad = \quad (A \cup \{e\}) \setminus \text{Expr}_x \]
\[ [M[e_1] = e_2;] \triangleright^{\sharp} A \quad = \quad A \cup \{e_1, e_2\} \]

By that, every path can be analyzed  :-)

A given program may admit several paths  :-(

For any given input, another path may be chosen  :-((
\[ [x = M[e];] A \supseteq (A \cup \{e\}) \setminus \text{Expr}_x \]
\[ [M[e_1] = e_2;] A = A \cup \{e_1, e_2\} \]

By that, every path can be analyzed  

A given program may admit several paths  

For any given input, another path may be chosen  

We require the set:

\[ \mathcal{A}[v] = \bigcap \{ [[\pi]] \setminus \emptyset \mid \pi : \text{start} \rightarrow^* v \} \]
Concretely:

→ We consider all paths $\pi$ which reach $v$.
→ For every path $\pi$, we determine the set of expressions which are available along $\pi$.
→ Initially at program start, nothing is available  :-)
→ We compute the intersection $\Rightarrow$ safe information
Concretely:

→ We consider all paths $\pi$ which reach $v$.
→ For every path $\pi$, we determine the set of expressions which are available along $\pi$.
→ Initially at program start, nothing is available :-)
→ We compute the intersection $\implies$ safe information

How do we exploit this information ???
Transformation 1.1:

We provide novel registers $T_e$ as storage for the $e$:

\[
x = e; \quad T_e = e; \quad x = T_e;
\]
Transformation 1.1:

We provide novel registers $T_e$ as storage for the $e$:

$\text{Pos}(e) = e$

$\text{Neg}(e) = T_e$

$\text{Pos}(T_e) = T_e$

$\text{Neg}(T_e) = e$
... analogously for \( R = M[e]; \) and \( M[e_1] = e_2; \).

**Transformation 1.2:**

If \( e \) is available at program point \( u \), then \( e \) need not be re-evaluated:

We replace the assignment with \( Nop \) :-)}
Example:

\[ x = y + 3; \]
\[ x = 7; \]
\[ z = y + 3; \]
Example:

\[ x = y + 3; \]
\[ x = 7; \]
\[ z = y + 3; \]
Example:

\[ x = y + 3; \]
\[ x = 7; \]
\[ z = y + 3; \]
Example:

\[
\begin{align*}
T &= y + 3; \\
\{y + 3\} &\rightarrow x = T; \\
\{y + 3\} &\rightarrow x = 7; \\
\{y + 3\} &\rightarrow z = T; \\
\{y + 3\} &\rightarrow \;
\end{align*}
\]

\[
\begin{align*}
x &= y + 3; \\
x &= 7; \\
z &= y + 3;
\end{align*}
\]
Correctness: (Idea)

Transformation 1.1 preserves the semantics and $A[u]$ for all program points $u$ :-) 

Assume $\pi : start \rightarrow^* u$ is the path taken by a computation. If $e \in A[u]$, then also $e \in [\pi]^\# \emptyset$.

Therefore, $\pi$ can be decomposed into:

\[ \text{start} \xrightarrow{\pi_1} u_1 \xrightarrow{k} u_2 \xrightarrow{\pi_2} u \]

with the following properties:
• The expression $e$ is evaluated at the edge $k$;
• The expression $e$ is not removed from the set of available expressions at any edge in $\pi_2$, i.e., no variable of $e$ receives a new value :-)

52
• The expression $e$ is evaluated at the edge $k$;
• The expression $e$ is not removed from the set of available expressions at any edge in $\pi_2$, i.e., no variable of $e$ receives a new value

The register $T_e$ contains the value of $e$ whenever $u$ is reached
Warning:

Transformation 1.1 is only meaningful for assignments \( x = e; \) where:

\[ \rightarrow x \notin Vars(e); \]
\[ \rightarrow e \notin Vars; \]
\[ \rightarrow \text{the evaluation of } e \text{ is non-trivial} \quad :-} \]
Warning:

Transformation 1.1 is only meaningful for assignments $x = e$; where:

$\rightarrow x \notin Vars(e);$  
$\rightarrow e \notin Vars;$  
$\rightarrow$ the evaluation of $e$ is non-trivial  ::- }

Which leaves us with the following question ...
Question:

How do we compute $A[u]$ for every program point $u$??

Idea:

We collect all restrictions to the values of $A[u]$ into a system of constraints:

$A[\text{start}] \subseteq \emptyset$

$A[v] \subseteq \{k\} \#(A[u])$

$k = (u, _{\text{edge}}, v)$