Background 1: An Operational Semantics

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

In the example:

```
A1 = A0 + 1 * i;
R1 = M[A1];
A2 = A0 + 1 * j;
R2 = M[A2];
Neg (R1 > R2)
Pos (R1 > R2)
A3 = A0 + 1 * j;
...
```

Remark

\( B \) is a repeated computation of the value of \( y + z \). if:

1. \( A \) is always executed before \( B \); and
2. \( y \) and \( z \) at \( B \) have the same values as at \( A \).

\[\rightarrow\] We need:

- an operational semantics;
- a method which identifies at least some repeated computations ...

Thereby, represent:

<table>
<thead>
<tr>
<th>vertex</th>
<th>program point</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>program start</td>
</tr>
<tr>
<td>stop</td>
<td>program exit</td>
</tr>
<tr>
<td>edge</td>
<td>step of computation</td>
</tr>
</tbody>
</table>
Background 1: An Operational Semantics

We choose a small-step operational approach. Programs are represented as control-flow graphs.

In the example:

```
A1 = A0 + 1 * i;
R1 = M[A1];
A2 = A0 + 1 * j;
R2 = M[A2];
A3 = A0 + 1 * j;
```

Thereby, represent:

<table>
<thead>
<tr>
<th>vertex</th>
<th>program point</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>program 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:

\[ \rho : \text{Vars} \rightarrow \text{int} \]
\[ \mu : \text{N} \rightarrow \text{int} \]

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

\[ [k] = [\text{lab}] \]

of the state:

\[ [s] = [\rho', \mu'] \]
Thereby, represent:

<table>
<thead>
<tr>
<th>vertex</th>
<th>program point</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>program 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**: \( \text{Pos}(e) \) or \( \text{Neg}(e) \)
- **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:

| \( \rho : \text{Vars} \to \text{int} \) | contents of registers |
| \( \mu : \mathbb{N} \to \text{int} \) | contents of storage |

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

\[ [k] = [\text{lab}] \]

of the state:
Computations follow paths.

Computations transform the current state

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

where:

\[
\begin{array}{|c|c|}
\hline
\rho : \text{Vars} \rightarrow \text{int} & \text{contents of registers} \\
\mu : \mathbb{N} \rightarrow \text{int} & \text{contents of storage} \\
\hline
\end{array}
\]

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

\[ [k] = [lab] \]

of the state:

\[
\begin{align*}
[k] (\rho, \mu) &= (\rho, \mu) \\
[\text{Pos}(e)] (\rho, \mu) &= (\rho, \mu) & \text{if } [e] \rho \neq 0 \\
[\text{Neg}(e)] (\rho, \mu) &= (\rho, \mu) & \text{if } [e] \rho = 0
\end{align*}
\]

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

// \[ x + y \] \{ x \mapsto 7, y \mapsto -1 \} = 6
// \[ ! (x == 4) \] \{ x \mapsto 5 \} = 1

\[ [R = e] (\rho, \mu) = (\rho \uplus [R \mapsto [e] \rho], \mu) \]

// where "\( \uplus \)" modifies a mapping at a given argument
\[ \| R = M[c]; ! (\rho, \mu) = (\rho \oplus \{ R \mapsto \mu([c] \rho)) \}, \mu) \]
\[ \| M[e_1] = e_2 ! (\rho, \mu) = (\rho, \mu \oplus \{ [e_1] \rho \mapsto [e_2] \rho\}) \]

Example

\[ [x = x + 1; ](\{x \mapsto 5\}, \mu) = (\rho, \mu) \quad \text{where:} \]

\[ \rho = \{ x \mapsto 5 \} \oplus \{ x \mapsto [x + 1] \{ x \mapsto 5\} \}
= \{ x \mapsto 5 \} \oplus \{ x \mapsto 6 \}
= \{ x \mapsto 6 \} \]

A path \( \pi = k_1k_2 \ldots k_m \) is a computation for the state \( s \) if:

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

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 \( v \):

\[ x+y \quad \pi \quad v \]

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 \( \pi \) ...

More generally:

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

... which transformations can be composed to the effect of a path

\( \pi = k_1 \ldots k_r \):

\[
\langle \pi \rangle^x = \langle k_r \rangle^x \circ \ldots \circ \langle k_1 \rangle^x
\]

The effect \( \langle k \rangle^x \) of an edge \( k = (u, \text{lab}, v) \) only depends on the label \( \text{lab} \), i.e.,

\[
\langle k \rangle^x = \langle \text{lab} \rangle^x
\]

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 \( \langle k \rangle^x A \) of expressions whose values are available after execution of \( k \) ...

\[
\langle \pi \rangle^x = \langle k_r \rangle^x \circ \ldots \circ \langle k_1 \rangle^x
\]
... which transformations can be composed to the **effect** of a path 
\( \pi = k_1 \ldots k_n \):

\[
\llbracket \pi \rrbracket^2 = \llbracket k_n \rrbracket^2 \circ \ldots \circ \llbracket k_1 \rrbracket^2
\]

The **effect** \( \llbracket k \rrbracket^2 \) of an edge \( k = (u, \text{lab}, v) \) only depends on the label \( \text{lab} \), i.e., \( \llbracket k \rrbracket^2 = \llbracket \text{lab} \rrbracket^2 \) where:

\[
\begin{align*}
\llbracket k \rrbracket^2 A & = A \\
\llbracket \text{Pos}(e) \rrbracket^2 A & = \llbracket \text{Neg}(e) \rrbracket^2 A = A \cup \{ e \} \\
\llbracket x = e \rrbracket^2 A & = (A \cup \{ e \}) \setminus \text{Expr}_x \quad \text{where} \\
& \text{Expr}_x \text{ all expressions which contain } x
\end{align*}
\]

By that, **every path** can be analyzed.

A given program may admit **several paths**.

For any given input, another path may be chosen ...

\[
\begin{align*}
\llbracket x = M[e] \rrbracket^2 A & = (A \cup \{ e \}) \setminus \text{Expr}_x \\
\llbracket M[e_1] = e_2 \rrbracket^2 A & = A \cup \{ e_1, e_2 \}
\end{align*}
\]

By that, **every path** can be analyzed.

A given program may admit **several paths**.

For any given input, another path may be chosen ...

\[
\begin{align*}
\llbracket x = M[e] \rrbracket^2 A & = (A \cup \{ e \}) \setminus \text{Expr}_x \\
\llbracket M[e_1] = e_2 \rrbracket^2 A & = A \cup \{ e_1, e_2 \}
\end{align*}
\]

\[
\begin{align*}
\mathcal{A}[v] & = \bigcap \{ \llbracket \pi \rrbracket^2 \theta \mid \pi : \text{start} \rightarrow^* v \}
\end{align*}
\]
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

How do we exploit this information ???

Transformation 1.1:

We provide novel registers \( T_e \) as storage for the \( e \):

\[ T_e = \epsilon; \]

\[ T_e = \epsilon; \]

\[ T_e = \epsilon; \]

\[ T_e = \epsilon; \]
Transformation 1.1:

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

... 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:

$x = y + 3$;
$x = 7$;
$z = y + 3$;
$T = y + 3$;
$x = T$;
$x = 7$;
$z = y + 3$;
$T = y + 3$;
$z = T$;
Example:

\[
\emptyset
\]

\[
x = y + 3;
\]
\[
x = 7;
\]
\[
z = y + 3;
\]
\[
\{ y + 3 \} \quad x = T;
\]
\[
\{ y + 3 \} \quad x = 7;
\]
\[
\{ y + 3 \} \quad T = y + 3;
\]
\[
\{ y + 3 \} \quad z = T;
\]

Correctness: (Idea)

Transformation 1.1 preserves the semantics and \( A[u] \) for all program points \( u \rightarrow \).

Assume \( \pi : \text{start} \rightarrow^* u \) is the path taken by a computation.

If \( e \in A[u] \), then also \( e \in [\pi]^P \emptyset \).

Therefore, \( \pi \) can be decomposed into:

\[
\begin{aligned}
\text{start} &\rightarrow \pi_1 \rightarrow k \rightarrow \pi_2 \rightarrow u
\end{aligned}
\]

with the following properties:

Example:

\[
\emptyset
\]

\[
x = y + 3;
\]
\[
x = 7;
\]
\[
z = y + 3;
\]
\[
\{ y + 3 \} \quad x = T;
\]
\[
\{ y + 3 \} \quad x = 7;
\]
\[
\{ y + 3 \} \quad T = y + 3;
\]
\[
\{ y + 3 \} \quad z = T;
\]

Correctness: (Idea)

Transformation 1.1 preserves the semantics and \( A[u] \) for all program points \( u \rightarrow \).

Assume \( \pi : \text{start} \rightarrow^* u \) is the path taken by a computation.

If \( e \in A[u] \), then also \( e \in [\pi]^P \emptyset \).

Therefore, \( \pi \) can be decomposed into:

\[
\begin{aligned}
\text{start} &\rightarrow \pi_1 \rightarrow k \rightarrow \pi_2 \rightarrow u
\end{aligned}
\]

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 :)

---

Warning:

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

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

---

• 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 :)

$\longrightarrow$

The register $T_r$ contains the value of $e$ whenever $u$ is reached :)})
Warning:

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

$\rightarrow e \notin Vars$;

$\rightarrow$ the evaluation of $e$ is non-trivial $:}$