Removing superfluous computations

1.1 Repeated computations

Idea:

If the same value is computed repeatedly, then

→ store it after the first computation;
→ replace every further computation through a look-up!

⇒ Availability of expressions
⇒ Memoization
Problem: Identify repeated computations!

Example:

\[ z = 1; \]
\[ y = M[17]; \]
\[ A : \quad x_1 = y + z; \]
\[ \ldots \]
\[ B : \quad x_2 = y + z; \]
Note:

$B$ 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 $A$ :) 

We need:

→ an operational semantics :) 

→ a method which identifies at least some repeated computations ...
Background 1: An Operational Semantics

we choose a small-step operational approach.

Programs are represented as control-flow graphs.

In the example:

\[
\begin{align*}
A_1 &= A_0 + 1 \times i; \\
R_1 &= M[A_1]; \\
A_2 &= A_0 + 1 \times j; \\
R_2 &= M[A_2]; \\
A_3 &= A_0 + 1 \times j;
\end{align*}
\]

\begin{align*}
\text{Neg} (R_1 > R_2) & \quad \text{Pos} (R_1 > R_2)
\end{align*}

\[\ldots\]
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>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:

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

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

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

of the state:
\[
[;] (\rho, \mu) = (\rho, \mu)
\]

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

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

\[\text{Pos}(e)\] (\(\rho, \mu\)) = (\(\rho, \mu\)) \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

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

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

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

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

Example:

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

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 $\pi = k_1 k_2 \ldots k_m$ 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 $u$:

$x+y$

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 \( k = (u, lab, v) \) only depends on the label \( lab \), i.e., \( [k]^\# = [lab]^\# \)
... 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., \([k]^\# = [\text{lab}]^\#\) where:

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

\(\text{Expr}_x\) all expressions which contain \(x\)
\[ \begin{align*}
[x = M[e];]^\# A &= (A \cup \{e\}) \setminus \text{Expr}_x \\
[M[e_1] = e_2;]^\# A &= A \cup \{e_1, e_2\}
\end{align*} \]
\[
[x = M[e];]^\# A = (A \cup \{e\}) \setminus Expr_x
\]
\[
[M[e_1] = e_2;]^\# A = A \cup \{e_1, e_2\}
\]

By that, every path can be analyzed  :-)

A given program may admit several paths  :-(

For any given input, another path may be chosen  :-((