1 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:

\[
\begin{align*}
    z &= 1; \\
    y &= M[17]; \\
    A: \quad x_1 &= y + z; \\
        \vdots \\
    B: \quad x_2 &= y + z;
\end{align*}
\]

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 \( A \)  \(\Rightarrow\)

⇒⇒⇒ We need:

→ an operational semantics  \(\Rightarrow\)
→ a method which identifies at least some repeated computations ...
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:

$\rightarrow$ an operational semantics

$\rightarrow$ a method which identifies at least some repeated computations ...
\[ \times = M \cdot \nabla \]

Computations follow paths.

Computations transform the current state

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

where:

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

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

\[ [k] = [lab] \]

of the state:

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

// [e] : evaluation of the expression \( e \), e.g.
// \([x + y] \{ x \mapsto 7, y \mapsto 1 \} = 6 \)
// \([x \cdot 4] \{ x \mapsto 4 \} = 1 \)

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

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

// [e] : evaluation of the expression \( e \), e.g.
// \([x + y] \{ x \mapsto 7, y \mapsto 1 \} = 6 \)
// \([x \cdot 4] \{ x \mapsto 4 \} = 1 \)

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

// where “\( \oplus \)” modifies a mapping at a given argument
\[

\xi \times \nabla \text{ } \mu \text{ } \nabla \rightarrow \xi \oplus \mu \nabla \rightarrow \xi \nabla \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\]

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

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

Example:

\([x = x + 1;] (\{x \mapsto 5\}, \mu) = (\rho, \mu) \quad \text{where:}\]
\[ \rho = \{x \mapsto 5\} \oplus \{x \mapsto [x + 1] \{x \mapsto 5\}\} \]
\[ = \{x \mapsto 5\} \oplus \{x \mapsto 6\} \]
\[ = \{x \mapsto 6\} \]

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

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

Example:

\([x = x + 1;] (\{x \mapsto 5\}, \mu) = (\rho, \mu) \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:
\[
\llbracket \pi \rrbracket s = (\[k_m\] \circ \ldots \circ \[k_1\]) s
\]

**Application:**

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

\[
\begin{array}{c}
x+y \\
0 \rightarrow \pi \rightarrow 1
\end{array}
\]

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 \( u \).

---

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

**Application:**

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

\[
\begin{array}{c}
A \\
0 \rightarrow \pi \rightarrow 1
\end{array}
\]

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

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

The effect \( [k]^f \) of an edge \( k = (u, lab, v) \) only depends on the label \( lab \), i.e., \( [k]^f = [lab]^f \).

... which transformations can be composed to the effect of a path \( \pi = k_1 \ldots k_r \):

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

The effect \( [k]^f \) of an edge \( k = (u, lab, v) \) only depends on the label \( lab \), i.e., \( [k]^f = [lab]^f \) where:

\[
[\emptyset]^f \cdot A = A
\]
\[
[\text{Neg}(c)]^f \cdot A = A \cup \{ c \}
\]
\[
[x = c]^f \cdot A = (A \cup \{ c \}) \setminus \text{Expr}_e
\]

\[ \chi = \chi + 1 \]
\[ [x = M[e];] A = (A \cup \{ e \}) \setminus \text{Expr}_x \]
\[ [M[e_1] = e_2] A = (A \cup \{ e_1, e_2 \}) \setminus \text{Expr}_x \]
\[ x = \underbrace{M[e_1]} \]
\[ y = \underbrace{\gamma + 1} \]
\[ \underbrace{M[y - n]} = S \]

By that, every path can be analyzed.
A given program may admit several paths.
For any given input, another path may be chosen.

\[ [x = M[e];] A = (A \cup \{ e \}) \setminus \text{Expr}_x \]
\[ [M[e_1] = e_2] A = A \cup \{ e_1, e_2 \} \]

Concretely:

\[ \rightarrow \text{ We consider all paths } \pi \text{ which reach } v. \]
\[ \rightarrow \text{ For every path } \pi, \text{ we determine the set of expressions which are available along } \pi. \]
\[ \rightarrow \text{ Initially at program start, nothing is available } \rightarrow \]
\[ \rightarrow \text{ We compute the intersection } \implies \text{ safe information} \]
\[ \{ x = M[e]; \}^A A = (A \cup \{ e \}) \setminus \text{Expr}_x \]
\[ \{ M[e] = e_i \}^A A = A \cup \{ e_1, e_2 \} \]

By that, every path can be analyzed  
A given program may admit several paths  
For any given input, another path may be chosen  

\[ \Rightarrow \text{ We require the set:} \]
\[ A[e] = \bigcap \{ \{ \pi \}^A | \pi : \text{start} \rightarrow^* e \} \]

---

**Transformation 1.1:**

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

---

**Concretely:**

\[ \Rightarrow \text{ We consider all paths } \pi \text{ which reach } e. \]
\[ \Rightarrow \text{ For every path } \pi, \text{ we determine the set of expressions which are available along } \pi. \]
\[ \Rightarrow \text{ Initially at program start, nothing is available } \Rightarrow \]
\[ \Rightarrow \text{ We compute the intersection } \Rightarrow \text{ safe information} \]

How do we exploit this information ???

---

**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 $Nop :-)$