1.13 Eliminating Partial Redundancies

Example:

\[ x = M[a]; \]
\[ y_1 = x + 1; \]
\[ y_2 = x + 1; \]
\[ M[x] = y_1 + y_2; \]

// \( x + 1 \) is evaluated on every path
// on one path, however, even twice
Removal of a partially available assignment:

\[ x = M[a]; \]
\[ y_1 = x + 1; \]
\[ y_2 = x + 1; \]
\[ M[x] = y_1 + y_2; \]
Terminological Confusion

In Section 1.4 we used, in describing the analysis:

- availability along a path,
- definite availability, i.e., availability along all paths.

secretly switched to availability meaning definite availability.

The compiler literature uses the terms

- redundancy elimination for the ensuing transformation,
- partial redundancy elimination for the actual transformation, and
- partially redundant for available along some paths.
Making assignments totally available without introducing redundant computations

\[ x \leftarrow e \quad x \leftarrow e \]

\[ x \leftarrow e \quad x \leftarrow e \quad x \leftarrow e \quad x \leftarrow e \]

\[ \cdot u \]

\[ x \leftarrow e \quad x \leftarrow e \quad x \leftarrow e \]

\[ x \leftarrow e \quad x \leftarrow e \]

\[ x \leftarrow e \quad x \leftarrow e \quad x \leftarrow e \quad x \leftarrow e \]

\[ x \leftarrow e \text{ part. red. at } u \quad \text{new red. computation} \quad x \leftarrow e \text{ very busy at } u \]
Idea:

(1) Insert assignments $T_e = e$; such that $e$ is available at all points where the value of $e$ is required.

(2) Thereby spare program points where $e$ either is already available or will definitely be computed in future.

Expressions with the latter property are called very busy.

(3) Replace the original evaluations of $e$ by accesses to the variable $T_e$.

$\rightarrow$ we require a novel analysis
An expression $e$ is called **busy** along a path $\pi$, if the expression $e$ is evaluated before any of the variables $x \in \text{Vars}(e)$ is overwritten.

// backward analysis!

$e$ is called **very busy** at $u$, if $e$ is busy along every path $\pi : u \rightarrow ^* \text{stop}$. 
An expression $e$ is called busy along a path $\pi$, if the expression $e$ is evaluated before any of the variables $x \in Vars(e)$ is overwritten.

// backward analysis!

e is called very busy at $u$, if $e$ is busy along every path $\pi : u \rightarrow^{*} stop$.

Accordingly, we require:

$$B[u] = \bigcap \{ \llbracket \pi \rrbracket^{\#} \emptyset \mid \pi : u \rightarrow^{*} stop \}$$

where for $\pi = k_1 \ldots k_m$:

$$\llbracket \pi \rrbracket^{\#} = \llbracket k_1 \rrbracket^{\#} \circ \ldots \circ \llbracket k_m \rrbracket^{\#}$$
Our complete lattice is given by:

\[ \mathbb{B} = 2^{\text{Expr}\setminus\text{Vars}} \quad \text{with} \quad \subseteq = \supseteq \]

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

\[
\begin{align*}
[;]^\# B &= B \\
[\text{Pos}(e)]^\# B &= [\text{Neg}(e)]^\# B = B \cup \{e\} \\
[x = e;]^\# B &= (B \setminus \text{Expr}_x) \cup \{e\} \\
[x = M[e];]^\# B &= (B \setminus \text{Expr}_x) \cup \{e\} \\
[M[e_1] = e_2;]^\# B &= B \cup \{e_1, e_2\}
\end{align*}
\]
These effects are all distributive. Thus, the least solution of the constraint system yields precisely the MOP — given that stop is reachable from every program point.

Example:

\[
x = M[a]; \\
y_1 = x + 1; \\
y_2 = x + 1; \\
M[x] = y_1 + y_2;
\]

\[
\begin{array}{c|c}
0 & \emptyset \\
1 & \emptyset \\
2 & \{x + 1\} \\
3 & \{x + 1\} \\
4 & \{x + 1\} \\
5 & \{y_1 + y_2\} \\
6 & \{x + 1\} \\
7 & \emptyset \\
\end{array}
\]
A program point $u$ is called safe for $e$, if $e \in A[u] \cup B[u]$, i.e., $e$ is either available or very busy.

Idea:

- We insert computations of $e$ such that $e$ becomes available at all safe program points.
- We insert $T_e = e$; after every edge $(u, lab, v)$ with

  $$e \in B[v] \backslash [\text{lab}]_A(A[u] \cup B[u])$$
Transformation PRE-1:

\[ T_e = e; \quad (e \in B[v] \setminus \llbracket lab \rrbracket_{A} (A[u] \cup B[u])) \]

\[ T_e = e; \quad (e \in B[v]) \]
Transformation PRE-2:

\[ u \xrightarrow{x} e; \]

// analogously for the other uses of \( e \)
// at old edges of the program.
In the Example:

\[ x = M[a]; \]
\[ y_1 = x + 1; \]
\[ y_2 = x + 1; \]
\[ M[x] = y_1 + y_2; \]

<table>
<thead>
<tr>
<th></th>
<th>( A )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>1</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>2</td>
<td>( \emptyset )</td>
<td>( { x + 1 } )</td>
</tr>
<tr>
<td>3</td>
<td>( \emptyset )</td>
<td>( { x + 1 } )</td>
</tr>
<tr>
<td>4</td>
<td>( { x + 1 } )</td>
<td>( { x + 1 } )</td>
</tr>
<tr>
<td>5</td>
<td>( \emptyset )</td>
<td>( { x + 1 } )</td>
</tr>
<tr>
<td>6</td>
<td>( { x + 1 } )</td>
<td>( { y_1 + y_2 } )</td>
</tr>
<tr>
<td>7</td>
<td>( { x + 1 } )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
In the Example:

\[ x = M[a]; \]
\[ y_1 = x + 1; \]
\[ y_2 = x + 1; \]
\[ M[x] = y_1 + y_2; \]

<table>
<thead>
<tr>
<th></th>
<th>( A )</th>
<th>( B )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>1</td>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>2</td>
<td>( \emptyset )</td>
<td>( { x + 1 } )</td>
</tr>
<tr>
<td>3</td>
<td>( \emptyset )</td>
<td>( { x + 1 } )</td>
</tr>
<tr>
<td>4</td>
<td>( { x + 1 } )</td>
<td>( { x + 1 } )</td>
</tr>
<tr>
<td>5</td>
<td>( \emptyset )</td>
<td>( { x + 1 } )</td>
</tr>
<tr>
<td>6</td>
<td>( { x + 1 } )</td>
<td>( { y_1 + y_2 } )</td>
</tr>
<tr>
<td>7</td>
<td>( { x + 1 } )</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Im Example:

1. \( x = M[a] \);
2. \( T = x + 1 \);
3. \( y_1 = T \);
4. \( y_2 = T \);
5. \( M[x] = y_1 + y_2 \);

\[
\begin{array}{|c|c|c|}
\hline
& A & B \\
\hline
0 & \emptyset & \emptyset \\
1 & \emptyset & \emptyset \\
2 & \emptyset & \{x + 1\} \\
3 & \emptyset & \{x + 1\} \\
4 & \{x + 1\} & \{x + 1\} \\
5 & \emptyset & \{x + 1\} \\
6 & \{x + 1\} & \{y_1 + y_2\} \\
7 & \{x + 1\} & \emptyset \\
\hline
\end{array}
\]
Correctness:

Let $\pi$ denote a path reaching $v$ after which a computation of an edge with $e$ follows.

Then there is a maximal suffix of $\pi$ such that for every edge $k = (u, lab, u')$ in the suffix:

$$e \in [lab]_A^\# (A[u] \cup B[u])$$
Correctness:

Let $\pi$ denote a path reaching $v$ after which a computation of an edge with $e$ follows.

Then there is a maximal suffix of $\pi$ such that for every edge $k = (u, lab, u')$ in the suffix:

$$e \in \left[ \frac{\text{lab}}{\text{A}} \right]_{\text{A}} (\mathcal{A}[u] \cup \mathcal{B}[u])$$

In particular, no variable in $e$ receives a new value.

Then $T_e = e;$ is inserted before the suffix.

\[\text{Diagram:}\]

```
A A A A A A
```

\[\text{Diagram:}\]

```
A A A A A A
```

```
T = e;
```

```
v
```
We conclude:

- Whenever the value of $e$ is required, $e$ is available
  $\implies$ correctness of the transformation

- Every $T = e$; which is inserted into a path corresponds to an $e$
  which is replaced with $T$
  $\implies$ non-degradation of the efficiency