1.7 Eliminating Partial Redundancies

Example:

```
x = M[a];
y1 = x + 1;
y2 = x + 1;
M[x] = y1 + y2;
```

```
// x + 1 is evaluated on every path ...
// on one path, however, even twice :-(
```
Goal:

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

\[
x = M[a];\\
T = x + 1;\\
T = x + 1;\\
\]

\[
y_1 = T;\\
M[x] = y_1 + T;
\]
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 \textbf{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 \textbf{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 $\pi : u \rightarrow^* \text{stop}$.

Accordingly, we require:

$$B[u] = \bigcap \{ [\pi]^\sharp \emptyset \mid \pi : u \rightarrow^* \text{stop} \}$$

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

$$[\pi]^\sharp = [k_1]^\sharp \circ \ldots \circ [k_m]^\sharp$$
Our complete lattice is given by:

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

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:**

\[
x = M[a];
\]

\[
y_1 = x + 1;
\]

\[
y_2 = x + 1;
\]

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

\[
\begin{array}{|c|c|}
\hline
7 & \emptyset \\
\hline
6 & \emptyset \\
\hline
5 & \{x + 1\} \\
\hline
4 & \{x + 1\} \\
\hline
3 & \{x + 1\} \\
\hline
2 & \{x + 1\} \\
\hline
1 & \emptyset \\
\hline
0 & \emptyset \\
\hline
\end{array}
\]
A point \( u \) is called safe for \( e \), if \( e \in \mathcal{A}[u] \cup \mathcal{B}[u] \), 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
  \[
e \in \mathcal{B}[v] \backslash \langle lab \rangle^\mathcal{A}_A(\mathcal{A}[u] \cup \mathcal{B}[u])
  \]
Transformation 5.1:

\[ T_e = e; \quad (e \in B[v] \setminus [lab]_A^B (A[u] \cup B[u])) \]

\[ T_e = e; \quad (e \in B[v]) \]
Transformation 5.2:

\[ u \]

\[ x = e; \]

\[ u \]

\[ x = T_e; \]

// analogously for the other uses of \( e \)
// at old edges of the program.
Bernhard Steffen, Dortmund

Jens Knoop, Wien
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>(\emptyset)</td>
</tr>
<tr>
<td>7</td>
<td>({x + 1})</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>
In the Example:

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

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

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

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

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

In particular, no variable in \( e \) receives a new value \( :-) \)
Then \( T_e = e; \) is inserted before the suffix \( :-)) \)
We conclude:

- Whenever the value of $e$ is required, $e$ is available $:-)\Rightarrow$ correctness of the transformation

- Every $T = e; \text{ which is inserted into a path corresponds to an } e \text{ which is replaced with } T \Rightarrow$ non-degradation of the efficiency
1.8 Application: Loop-invariant Code

Example:

```
for (i = 0; i < n; i++)
    a[i] = b + 3;
```

// The expression  $b + 3$  is recomputed in every iteration  :-(
// This should be avoided  :-(
The Control-flow Graph:

0

\[ i = 0; \]

1

\[ \text{Neg}(i < n) \]

7

\[ y = b + 3; \]

2

\[ \text{Pos}(i < n) \]

3

\[ A_1 = A + i; \]

4

\[ M[A_1] = y; \]

5

\[ i = i + 1; \]

6
Warning: $T = b + 3$; may not be placed before the loop:

There is no decent place for $T = b + 3$; :-(

$\Rightarrow$ There is no decent place for $T = b + 3$; :-(
Idea: Transform into a do-while-loop ...

```
i = 0;
```

```
if \( i < n \) then

\[ A_1 = A + i; \]

\[ M[A_1] = y; \]

\[ i = i + 1; \]

end if

```

```
if \( i < n \) then

\[ y = b + 3; \]

\[ A_1 = A + i; \]

\[ M[A_1] = y; \]

\[ i = i + 1; \]

end if

```

```
... now there is a place for \( T = e; \) :-)

\[
\begin{align*}
0 & \quad i = 0; \\
1 & \quad \text{Pos}(i < n) \\
2 & \quad \text{Neg}(i < n) \\
3 & \quad T = b + 3; \\
4 & \quad y = T; \\
5 & \quad A_1 = A + i; \\
6 & \quad M[A_1] = y; \\
7 & \quad i = i + 1; \\
\end{align*}
\]
Application of $T5$ (PRE):

1. $i = 0$;
2. $y = b + 3$;
3. $A_1 = A + i$;
4. $M[A_1] = y$;
5. $i = i + 1$;
6. $\text{Neg}(i < n)$
7. $\text{Pos}(i < n)$

<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>${b + 3}$</td>
</tr>
<tr>
<td>3</td>
<td>${b + 3}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>4</td>
<td>${b + 3}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>5</td>
<td>${b + 3}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>6</td>
<td>${b + 3}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>7</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Application of T5 (PRE):
Conclusion:

- Elimination of partial redundancies may move loop-invariant code out of the loop :-(
- This only works properly for do-while-loops :-(
- To optimize other loops, we transform them into do-while-loops before-hand:

\[
\text{while } (b) \text{ stmt } \quad \Longrightarrow \quad \text{if } (b) \\
\quad \text{do stmt} \\
\quad \text{while } (b); \\
\quad \Longrightarrow \quad \text{Loop Rotation}
\]
Problem:

If we do not have the source program at hand, we must re-construct potential loop headers ;-) 

\[\Rightarrow\quad \text{Pre-dominators}\]

\(u\) pre-dominates \(v\), if every path \(\pi: \text{start} \rightarrow^* v\) contains \(u\). 
We write: \(u \Rightarrow v\).

\(\Rightarrow\) is reflexive, transitive and anti-symmetric :-)
**Computation:**

We collect the nodes along paths by means of the analysis:

\[ P = 2^{\text{Nodes}}, \quad \subseteq = \supseteq \]

\[ \left[ (\_, \_, v) \right]^\# P = P \cup \{v\} \]

Then the set \( \mathcal{P}[v] \) of pre-dominators is given by:

\[ \mathcal{P}[v] = \bigcap \{ [\pi]^\# \{\text{start}\} | \pi : \text{start} \rightarrow^* v \} \]
Since $[k]^{\#}$ are distributive, the $P[v]$ can be computed by means of fixpoint iteration :-)

**Example:**

```
<table>
<thead>
<tr>
<th></th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{0}</td>
</tr>
<tr>
<td>1</td>
<td>{0,1}</td>
</tr>
<tr>
<td>2</td>
<td>{0,1,2}</td>
</tr>
<tr>
<td>3</td>
<td>{0,1,2,3}</td>
</tr>
<tr>
<td>4</td>
<td>{0,1,2,3,4}</td>
</tr>
<tr>
<td>5</td>
<td>{0,1,5}</td>
</tr>
</tbody>
</table>
```
The partial ordering "⇒" in the example:

\[
\begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
\end{array}
\]

\[
\begin{array}{c|c}
& \mathcal{P} \\
0 & \{0\} \\
1 & \{0, 1\} \\
2 & \{0, 1, 2\} \\
3 & \{0, 1, 2, 3\} \\
4 & \{0, 1, 2, 3, 4\} \\
5 & \{0, 1, 5\} \\
\end{array}
\]
Apparently, the result is a tree :-) 

In fact, we have:

**Theorem:** 

Every node \( v \) has at most one immediate pre-dominator.

**Proof:**

**Assume:**

there are \( u_1 \neq u_2 \) which immediately pre-dominate \( v \).

If \( u_1 \Rightarrow u_2 \) then \( u_1 \) not immediate.

Consequently, \( u_1, u_2 \) are incomparable :-)
Now for every $\pi : start \rightarrow^* v$:

$$\pi = \pi_1 \pi_2 \quad \text{with} \quad \pi_1 : start \rightarrow^* u_1$$
$$\pi_2 : u_1 \rightarrow^* v$$

If, however, $u_1, u_2$ are incomparable, then there is path:

$start \rightarrow^* v$ avoiding $u_2$:
Now for every $\pi : start \rightarrow^* v$:

$$\pi = \pi_1 \pi_2 \quad \text{with} \quad \pi_1 : start \rightarrow^* u_1$$

$$\pi_2 : u_1 \rightarrow^* v$$

If, however, $u_1, u_2$ are incomparable, then there is path:

$\text{start} \rightarrow^* v$ avoiding $u_2$:
Observation:

The loop head of a while-loop pre-dominates every node in the body.

A back edge from the exit $u$ to the loop head $v$ can be identified through

$$v \in P[u]$$

Accordingly, we define:
Transformation 6:

$$u_2, v \in P[u]$$

$$u_1 \not\in P[u]$$

We duplicate the entry check to all back edges  :-}
... in the Example:

\[ i = 0; \]

\[ \text{Neg}(i < n) \quad \text{Pos}(i < n) \]

\[ y = b + 3; \]

\[ A_1 = A + i; \]

\[ M[A_1] = y; \]

\[ i = i + 1; \]
... in the Example:

```
0, 1, 7
```

```
0, 1, 2
```

```
0, 1, 2, 3
```

```
0, 1, 2, 3, 4
```

```
0, 1, 2, 3, 4, 5
```

```
0, 1, 2, 3, 4, 5, 6
```

```
i = 0;
```

```
Neg(i < n)  Pos(i < n)
```

```
y = b + 3;
```

```
A_1 = A + i;
```

```
M[A_1] = y;
```

```
i = i + 1;
```
... in the Example:

\[ i = 0; \]

\[ \text{Neg}(i < n) \]
\[ \text{Pos}(i < n) \]

\[ y = b + 3; \]

\[ A_1 = A + i; \]

\[ M[A_1] = y; \]

\[ i = i + 1; \]
... in the Example:

```
0, 1, 7

0
i = 0;

1
0, 1

Neg(i < n)  Pos(i < n)

2
0, 1, 2

2
0, 1, 2

y = b + 3;

3
0, 1, 2, 3

A_1 = A + i;

4
0, 1, 2, 3, 4

M[A_1] = y;

5
0, 1, 2, 3, 4, 5

i = i + 1;

6
0, 1, 2, 3, 4, 5, 6

Neg(i < n)  Pos(i < n)
```
Warning:

There are \textit{unusual} loops which cannot be rotated:

\begin{itemize}
\item \textbf{Pre-dominators:}
\end{itemize}
... but also common ones which cannot be rotated:

Here, the complete block between back edge and conditional jump should be duplicated  :-(
... but also *common ones* which cannot be rotated:

Here, the complete block between back edge and conditional jump should be duplicated  :-(
... but also **common ones** which cannot be rotated:

Here, the complete block between back edge and conditional jump should be duplicated  :-(
1.9 Eliminating Partially Dead Code

Example:

\[ T = x + 1; \]
\[ M[x] = T; \]

\[ x + 1 \] need only be computed along one path  ;-(

\[ \]
Idea:

\[ T = x + 1; \]

\[ M[x] = T; \]