Evaluation starts with an interesting unknown $x_i$ (e.g., the value at $\text{stop}$)

Then automatically all unknowns are evaluated which influence $x_i$ :-)

The number of evaluations is often smaller than during worklist iteration :-)

The algorithm is more complex but does not rely on pre-computation of variable dependencies :-)

It also works if variable dependencies during iteration change !!!

--- interprocedural analysis

Evaluation starts with an interesting unknown $x_i$ (e.g., the value at $\text{stop}$)

Then automatically all unknowns are evaluated which influence $x_i$ :-)

The number of evaluations is often smaller than during worklist iteration :-)

The algorithm is more complex but does not rely on pre-computation of variable dependencies :-)

It also works if variable dependencies during iteration change !!!

--- interprocedural analysis
1.7 Eliminating Partial Redundancies

Example:

```plaintext
x = f[0];
y = x + 1;
y1 = x + 1;
M[x] = y1 + y2;
```

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

Idea:

1. Insert assignments $T_x = e$: 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 $T_x$.

   we require a novel analysis :-(

Goal:

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

```plaintext
x = M[a];
T = x + 1;
M[x] = y1 + T;
```

```plaintext
x = M[a];
y1 = x + 1;
T = x + 1;
M[x] = y1 + 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 $T_e$.

$\implies$ we require a novel analysis \( \therefore \))

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:

$$B[u] = \bigcap\{[\pi]^2 \mid \pi : u \rightarrow^* stop\}$$

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

$$[\pi]^2 = [k_1]^2 \circ \ldots \circ [k_m]^2$$

Our complete lattice is given by:

$$B = 2^{Expr \setminus Vars} \quad \text{with} \quad \bot = \top$$

The effect $[k]^2$ of an edge $k = (u, lab, v)$ only depends on $lab$, i.e., $[k]^2 = [lab]^2$ where:

$$[\bot]^2 B = B$$
$$[\text{Pos}(e)]^2 B = [\text{Neg}(e)]^2 B = B \cup \{e\}$$
$$[x = e]^2 B = (B \setminus \text{Expr}_x) \cup \{e\}$$
$$[x = M[e]; e_2]^2 B = (B \setminus \text{Expr}_x) \cup \{e\}$$
$$[M[e_1] = e_2]^2 B = B \cup \{e_1, e_2\}$$
Our complete lattice is given by:
\[ \mathcal{B} = 2^{\text{Expr} \setminus \text{Vars}} \quad \text{with} \quad \subseteq = \sqsubseteq \]

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

where:

\[
\begin{align*}
[k]^B B &= B \\
[\text{Pos}(e)]^B B &= [\text{Neg}(e)]^B B = B \cup \{e\} \\
[x = e]^B B &= (B \setminus \text{Expr}_x) \cup \{e\} \\
[x = M[e]]^B B &= (B \setminus \text{Expr}_x) \cup \{e\} \\
[M[x_1] = e_2]^B B &= B \cup \{e_1, e_2\}
\end{align*}
\]

These effects are all **distributive**. Thus, the least solution of the constraint system yields precisely the MOP — given that \textit{stop} is reachable from every program point :-)

Example:

\[
\begin{array}{c|c}
\text{state} & \text{value} \\
\hline
7 & \emptyset \\
6 & \{y_1 + y_2\} \\
5 & \{x + 1\} \\
4 & \{x + 1\} \\
3 & \{x + 1\} \\
2 & \{x + 1\} \\
1 & \emptyset \\
0 & \emptyset \\
\end{array}
\]

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 \((u, \text{lab}, v)\) with \(e \in \mathcal{B}[v] \setminus [\text{lab}]^B, (\mathcal{A}[u] \cup \mathcal{B}[u])\)
Transformation 5.1:

\[ T_x = e_1 : (e \in \mathcal{E}[x]) \]

\[ T_x \leftarrow e_1 : (e \in \mathcal{E}[x]) \]

Transformation 5.2:

\[ x = e_1 \]

\[ x = T_x \]

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

---

Transformation 5.2:

\[ x = e_1 \]

\[ x = T_x \]

// 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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>3</td>
<td>{x + 1}</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>4</td>
<td>{x + 1}</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>5</td>
<td>0</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, y_1 + y_2}</td>
<td>0</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>3</td>
<td>{x + 1}</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>4</td>
<td>{x + 1}</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>5</td>
<td>0</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>0</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>3</td>
<td>{x + 1}</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>4</td>
<td>{x + 1}</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>5</td>
<td>0</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>0</td>
</tr>
</tbody>
</table>

Im Example:

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>3</td>
<td>{x + 1}</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>4</td>
<td>{x + 1}</td>
<td>{x + 1}</td>
</tr>
<tr>
<td>5</td>
<td>0</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>0</td>
</tr>
</tbody>
</table>
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 $k = (u, \text{lab}, u')$ in the suffix:

$$e \in [\text{lab}]_A^\pi (A[u] \cup B[u])$$

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

Then $T_e = e_i$ is inserted before the suffix.

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 $k = (u, \text{lab}, u')$ in the suffix:

$$e \in [\text{lab}]_A^\pi (A[u] \cup B[u])$$

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

Then $T_e = e_i$ is inserted before the suffix.
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 $e = (u, lab, u')$ in the suffix:

$$e \in \{alb\}_A^\star (A[u] \cup B[u])$$

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 $:-)$
  $\implies$ 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

1.8 Application: Loop-invariant Code

Example:

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

// The expression $b + 3$ is recomputed every iteration $:-($
// This should be avoided $:-)$
```

The Control-flow Graph:
Warning: \( T = b + 3; \) may not be placed before the loop:

1. \( i = 0; \)
2. \( T = b + 3; \)
3. \( y = T; \)
4. \( A_1 = A + i; \)
5. \( M[A_1] = y; \)
6. \( i = i + 1; \)

There is no decent place for \( T = b + 3; \) :(

Idea: Transform into a do-while-loop...

Warning: \( T = b + 3; \) may not be placed before the loop:

1. \( i = 0; \)
2. \( T = b + 3; \)
3. \( y = T; \)
4. \( A_1 = A + i; \)
5. \( M[A_1] = y; \)
6. \( i = i + 1; \)

There is no decent place for \( T = b + 3; \) :(

Idea: Transform into a do-while-loop...
Application of T5 (PRE):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>{b+3}</td>
</tr>
<tr>
<td>3</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

```plaintext
while (b) stmt  →  if (b)
    do stmt
    while (b);

→  Loop Rotation
```

Application of T5 (PRE):

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>{b+3}</td>
</tr>
<tr>
<td>3</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Application of T5 (PRE):

```plaintext
T = 5 + 6
```

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>{b+3}</td>
</tr>
<tr>
<td>3</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>{b+3}</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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:

  while (b) stmt ➔ if (b)
  do stmt
  while (b);

  ➔ Loop Rotation

Problem:

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

  ➔ Pre-dominators

u pre-dominates v, if every path \( \pi : start \rightarrow^* v \) 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:

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

\[
[(\ldots, v)]^2 P = P \cup \{v\}
\]

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

\[
P[v] = \bigcap\{\pi \} \{\text{start} | \pi : \text{start} \rightarrow^* v\}
\]

Since \([k]^2\) are distributive, the \( P[v] \) can computed by means of fixpoint iteration :-(

Example:

```
\[
\begin{array}{c|c}
\text{P} & \text{P} \\
0 & \{0\} \\
1 & \{0, 1\} \\
2 & \{0, 1, 2\} \\
3 & \{0, 1, 2, 3\} \\
4 & \{0, 1, 2, 3, 4\} \\
5 & \{0, 1, 5\} \\
\end{array}
\]
```
Since $[k]^2$ are distributive, the $\mathcal{P}[v]$ can be computed by means of fixpoint iteration

**Example:**

<table>
<thead>
<tr>
<th></th>
<th>$\mathcal{P}$</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 $\Rightarrow$ in the example:

### First Iteration

<table>
<thead>
<tr>
<th></th>
<th>$\mathcal{P}$</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>

### Second Iteration

<table>
<thead>
<tr>
<th></th>
<th>$\mathcal{P}$</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 $\Rightarrow$ in the example:
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 $\pi : \text{start} \rightarrow^* v$ :

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

If, however, $u_1, u_2$ are incomparable, then there is path: $\text{start} \rightarrow^* v$
avoiding $u_2$ :

Now for every $\pi : \text{start} \rightarrow^* v$ :

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

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

$$v \in \mathcal{P}[u]$$

Accordingly, we define:

We duplicate the entry check to all back edges.

... in the Example:

... in the Example:
Warning:

There are unusual loops which cannot be rotated:

... but also common ones which cannot be rotated:

Here, the complete block between back edge and conditional jump should be duplicated  :-(

Pre-dominators: