Phases of a compiler

Figure 1.6, page 5 of text

Optimizations
Extending transfer equations from statements to blocks

Composition of $f_1$ and $f_2$:

$$f_1(x) = \text{gen}_1 \cup (x - \text{kill}_1)$$

$$f_2(x) = \text{gen}_2 \cup (x - \text{kill}_2)$$

$$f_2(f_1(x)) = \text{gen}_2 \cup ((\text{gen}_1 \cup (x - \text{kill}_1)) - \text{kill}_2)$$

$$= \text{gen}_2 \cup ((\text{gen}_1 - \text{kill}_2) \cup ((x - \text{kill}_1) - \text{kill}_2))$$

$$= \text{gen}_2 \cup (\text{gen}_1 - \text{kill}_2) \cup (x - (\text{kill}_1 \cup \text{kill}_2))$$
Extending transfer equations from statements to blocks

In general:

\[ f_B(x) = \text{gen}_B \cup (x - \text{kill}_B) \]

\[ \text{kill}_B = \bigcup_{i \in n} \text{kill}_i \]

\[ \text{gen}_B = \text{gen}_n \cup (\text{gen}_{n-1} - \text{kill}_n) \cup (\text{gen}_{n-2} - \text{kill}_{n-1} - \text{kill}_n) \cup \ldots \cup (\text{gen}_1 - \text{kill}_2 - \text{kill}_3 - \ldots - \text{kill}_n) \]
Extending transfer equations from statements to blocks

"The gen set contains all the definitions inside the block that are "visible" immediately after the block - we refer to them as downwards exposed. A definition is downwards exposed in a basic block only if it is not "killed" by a subsequent definition to the same variable inside the same basic block." [p. 605]
Iterative algorithm for reaching definitions

Algorithm [p. 606]

INPUT: A flow graph for which $\text{kill}_B$ and $\text{gen}_B$ have been computed for each block $B$.

OUTPUT: $\text{IN}[B]$ and $\text{OUT}[B]$, the set of definitions reaching the entry and exit of each block $B$ of the flow graph

METHOD:

\[
\text{OUT}[\text{ENTRY}] = \emptyset \\
\text{for (each basic block } B \text{ other than ENTRY) } \{ \text{OUT}[B] = \emptyset \} \\
\text{while (changes to any OUT occurs) } \{ \\
\hspace{1em} \text{for (each basic block } B \text{ other than ENTRY) } \{ \\
\hspace{2em} \text{IN}[B] = \bigcup P \text{ a predecessor of } B \text{ OUT}[P] \\
\hspace{2em} \text{OUT}[B] = \text{gen}_B \cup ( \text{IN}[B] - \text{kill}_B ) \\
\hspace{1em} \} \\
\} \\
\]

See footnote 4 on page 606
Iterative algorithm for reaching definitions

Algorithm [p. 606]

INPUT: A flow graph for which $\text{kill}_B$ and $\text{gen}_B$ have been computed for each block $B$.

OUTPUT: $\text{IN}[B]$ and $\text{OUT}[B]$, the set of definitions reaching the entry and exit of each block $B$ of the flow graph.

METHOD:

\[
\text{OUT}[\text{ENTRY}] = \emptyset
\]

for (each basic block $B$ other than \text{ENTRY}) \{ $\text{OUT}[B] = \emptyset$ \}

while (changes to any $\text{OUT}$ occurs) {
    for (each basic block $B$ other than \text{ENTRY}) {
        $\text{IN}[B] = \bigcup_{\text{a predecessor of } B} \text{OUT}[P]$
        $\text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B)$
    }
}

Written this way to allow different entry conditions for different data flow algorithms.

See footnote 4 on page 606
**Figure 9.13** (p. 604)

- **B1**
  - $d_1: i = m - 1$
  - $d_2: j = n$
  - $d_3: a = u_1$
  - $\text{gen}_{B1} = \{ d_1, d_2, d_3 \}$
  - $\text{kill}_{B1} = \{ d_4, d_5, d_6, d_7 \}$

- **B2**
  - $d_4: i = i + 1$
  - $d_5: j = j - 1$
  - $\text{gen}_{B2} = \{ d_4, d_5 \}$
  - $\text{kill}_{B2} = \{ d_1, d_2, d_7 \}$

- **B3**
  - $d_6: a = u_2$
  - $\text{gen}_{B3} = \{ d_6 \}$
  - $\text{kill}_{B3} = \{ d_3 \}$

- **B4**
  - $d_7: i = u_3$
  - $\text{gen}_{B4} = \{ d_7 \}$
  - $\text{kill}_{B4} = \{ d_1, d_4 \}$
Example 9.12 - building off figure 9.13

\[
\begin{align*}
\text{OUT}[\text{ENTRY}] &= \emptyset \\
\text{for (each basic block B other than ENTRY) } \{ \\
\text{OUT}[B] &= \emptyset \} \\
\text{while (changes to any OUT occurs) } \{ \\
\text{for (each basic block B other than ENTRY) } \{ \\
\text{IN}[B] &= \cup \text{ a predecessor of B OUT}[P] \\
\text{OUT}[B] &= \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B) \\
\} \\
\} \\
\end{align*}
\]

Represent \(d_i\) as a bit vector, where each \(d\) is a definition from 9.13

Union of sets \(A \cup B\): \(A \text{ OR } B\)

Difference of sets \(A - B\): \(A \text{ AND } B'\)

Compute in order \(B_1, B_2, B_3, B_4, \text{EXIT}\)

For example:

\[
\begin{align*}
\text{IN}[B_2]^1 &= \text{OUT}[B_1]^1 \cup \text{OUT}[B_4]^0 = 111 \, 0000 \cup 000 \, 0000 = 111 \, 0000 \\
\text{OUT}[B_2]^1 &= \text{gen}_{B_2} \cup (\text{IN}[B_2]^1 - \text{kill}_{B_2}) \\
&= 000 \, 1100 + (111 \, 0000 - 110 \, 0001) \\
&= 000 \, 1100 + 001 \, 0000 = 001 \, 1100
\end{align*}
\]
Example 9.12 - building off figure 9.13

\[
\text{OUT[ENTRY]} = \emptyset \\
\text{for (each basic block } B \text{ other than ENTRY) } \{ \text{ OUT} [B] = \emptyset \} \\
\text{while (changes to any OUT occurs) } \{
\text{ for (each basic block } B \text{ other than ENTRY) } \{
\quad \text{IN}[B] = \cup \text{ a predecessor of } B \text{ OUT}[P] \\
\quad \text{OUT}[B] = \text{ gen } B \cup (\text{ IN}[B] - \text{ kill } B )
\}
\}
\]
Example 9.12

\[ \text{OUT[ENTRY]} = \emptyset \]

for (each basic block B other than ENTRY) \{ \text{OUT[B]} = \emptyset \} 

while (changes to any OUT occurs) \{ 
    for (each basic block B other than ENTRY) \{ 
        \text{IN[B]} = \cup \text{ a predecessor of B OUT[P]} 
        \text{OUT[B]} = \text{gen}_B \cup (\text{IN[B]} - \text{kill}_B) 
    \} 
\} 

d4: i = i + 1

d5: j = j - 1

d7: i = u3

d6: a = u2

d1: i = m - 1

d2: j = n

d3: a = u1

\[ \text{B1} \]
\[ \text{000 0000} \]
\[ \text{000 0000} \]
\[ \text{111 0000} \]

\[ \text{IN[B1]} = \text{pred(B1)} = \text{ENTRY} \]
\[ \text{OUT[B1]} = \text{gen}_{B1} \cup (\text{IN[B1]} - \text{kill}_{B1}) \]
\[ \text{gen}_{B1} = \{ \text{d1, d2, d3} \} \]
\[ \text{kill}_{B1} = \{ \text{d4, d5, d6, d7} \} \]
Example 9.12

\[ \text{OUT}[\text{ENTRY}] = \emptyset \]

for (each basic block \( B \) other than ENTRY) \{ \( \text{OUT}[B] = \emptyset \) \}

while (changes to any \( \text{OUT} \) occurs) \{

\[ \text{for (each basic block } B \text{ other than ENTRY) } \{ \]

\[ \text{IN}[B] = \cup \text{ a predecessor of } B \text{ OUT}[P] \]

\[ \text{OUT}[B] = \text{gen}_B \cup ( \text{IN}[B] - \text{kill}_B ) \]

\[ \} \]

\[ \} \]

<table>
<thead>
<tr>
<th></th>
<th>( \text{OUT}[B]^0 )</th>
<th>( \text{IN}[B]^1 )</th>
<th>( \text{OUT}[B]^1 )</th>
<th>( \text{IN}[B]^2 )</th>
<th>( \text{OUT}[B]^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>000 0000</td>
<td>000 0000</td>
<td>111 0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>000 0000</td>
<td>111 0000</td>
<td>001 1100</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>\text{gen}_{B2} = { d4, d5 }</td>
<td>\text{kill}_{B2} = { d1, d2, d7 }</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td>000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td>000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 9.12

\[ \text{OUT}[\text{ENTRY}] = \emptyset \]

for (each basic block \(B\) other than \(\text{ENTRY}\)) \{ \(\text{OUT}[B] = \emptyset\) \}

while (changes to any \(\text{OUT}\) occurs) \{
    for (each basic block \(B\) other than \(\text{ENTRY}\)) \{
        \(\text{IN}[B] = \cup \) a predecessor of \(B\) \(\text{OUT}[P]\)
        \(\text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B)\)
    \}
\}

\(B1\)
\(\text{OUT}[B1] = 000 0000\)
\(\text{IN}[B1] = 000 0000\)
\(\text{OUT}[B1] = 111 0000\)
\(\text{ENTRY}\)

\(B2\)
\(\text{OUT}[B2] = 000 0000\)
\(\text{IN}[B2] = 111 0000\)
\(\text{OUT}[B2] = 001 1100\)

\(B3\)
\(\text{OUT}[B3] = 000 0000\)
\(\text{IN}[B3] = \text{pred}(B3) = \text{OUT}[B2]\)
\(\text{OUT}[B3] = \text{gen}_{B3} \cup (\text{IN}[B3] - \text{kill}_{B3})\)
\(\text{gen}_{B3} = \{d6\}\)
\(\text{kill}_{B3} = \{d3\}\)

\(B4\)
\(\text{OUT}[B4] = 000 0000\)

\(\text{EXIT}\)
\(\text{OUT}[\text{EXIT}] = 000 0000\)
Example 9.12

\[ \text{OUT[ENTRY]} = \emptyset \]

for (each basic block \( B \) other than \text{ENTRY}) \{ \text{OUT}[B] = \emptyset \}

while (changes to any \text{OUT} occurs) \{ 
    for (each basic block \( B \) other than \text{ENTRY}) \{ 
        \text{IN}[B] = \cup \text{a predecessor of } B \text{ OUT}[P] 
        \text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B) 
    \}
\}

\begin{array}{|c|c|c|c|c|}
\hline
\text{OUT}[B]^0 & \text{IN}[B]^1 & \text{OUT}[B]^1 & \text{IN}[B]^2 & \text{OUT}[B]^2 \\
\hline
\text{B1} & 000 0000 & 000 0000 & 111 0000 & \\
\hline
\text{B2} & 000 0000 & 111 0000 & 001 1100 & \\
\hline
\text{B3} & 000 0000 & 001 1100 & 000 1110 & \\
\hline
\text{B4} & 000 0000 & 001 1110 & 001 0111 & \\
\hline
\text{EXIT} & 000 0000 & \text{IN}[B4] = \text{OUT}[B2] \cup \text{OUT}[B3] & \text{OUT}[B4] = \text{gen}_{B4} \cup (\text{IN}[B4] - \text{kill}_{B4}) & \\
\hline
\end{array}

\text{gen}_{B4} = \{ \text{d7} \} 
\text{kill}_{B4} = \{ \text{d1, d4} \}
Example 9.12

\[ \text{OUT[ENTRY]} = \emptyset \]

for (each basic block \(B\) other than ENTRY) \{ OUT\[B\] = \emptyset \}

while (changes to any \(\text{OUT}\) occurs) \{
    for (each basic block \(B\) other than ENTRY) \{
        \(\text{IN}[B] = \cup \text{a predecessor of } B \text{ OUT}[^P]\)
        \(\text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B)\)
    \}
\}

\[
\begin{array}{c|c|c|c|c|c}
\text{OUT}[B]^0 & \text{IN}[B]^1 & \text{OUT}[B]^1 & \text{IN}[B]^2 & \text{OUT}[B]^2 \\
\hline
\text{B1} & 000 0000 & 000 0000 & 111 0000 & \\
\text{B2} & 000 0000 & 111 0000 & 001 1100 & \\
\text{B3} & 000 0000 & 001 1100 & 000 1110 & \\
\text{B4} & 000 0000 & 001 1110 & 001 0111 & \\
\text{EXIT} & 000 0000 & 001 0111 & 001 0111 & \\
\end{array}
\]

\[ \text{IN(EXIT)} = \text{OUT[B4]} \]

\[ \text{OUT(EXIT) = IN(EXIT)} \]
\[ \text{OUT[ENTRY]} = \emptyset \]
for (each basic block \( B \) other than \( \text{ENTRY} \)) { \( \text{OUT}[B] = \emptyset \) }
while (changes to any \( \text{OUT} \) occurs) {
    for (each basic block \( B \) other than \( \text{ENTRY} \)) {
        \( \text{IN}[B] = \bigcup \text{a predecessor of } B \ \text{OUT}[P] \)
        \( \text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B) \)
    }
}
\[
\text{OUT}[\text{ENTRY}] = \emptyset
\]

for (each basic block \( B \) other than \( \text{ENTRY} \)) \{ OUT\( [B] \) = \( \emptyset \) \}

while (changes to any \( \text{OUT} \) occurs) \{
   for (each basic block \( B \) other than \( \text{ENTRY} \)) \{
      IN\( [B] \) = U \( \text{a predecessor of } B \) \( \text{OUT}[P] \)
      \( \text{OUT}[B] = \text{gen}_B \cup ( \text{IN}[B] - \text{kill}_B ) \)
   \}
\}

<table>
<thead>
<tr>
<th>( B )</th>
<th>( \text{OUT}[B]^0 )</th>
<th>( \text{IN}[B]^1 )</th>
<th>( \text{OUT}[B]^1 )</th>
<th>( \text{IN}[B]^2 )</th>
<th>( \text{OUT}[B]^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B1 )</td>
<td>000 0000</td>
<td>000 0000</td>
<td>111 0000</td>
<td>000 0000</td>
<td>111 0000</td>
</tr>
<tr>
<td>( B2 )</td>
<td>000 0000</td>
<td>111 0000</td>
<td>001 1100</td>
<td>111 0111</td>
<td>001 1110</td>
</tr>
<tr>
<td>( B3 )</td>
<td>000 0000</td>
<td>001 1100</td>
<td>000 1110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( B4 )</td>
<td>000 0000</td>
<td>001 1100</td>
<td>001 0111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{EXIT} )</td>
<td>000 0000</td>
<td>001 0111</td>
<td>001 0111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ \text{OUT}[\text{ENTRY}] = \emptyset \]

for (each basic block \( \text{B} \) other than \( \text{ENTRY} \)) { \( \text{OUT}[\text{B}] = \emptyset \) }

while (changes to any \( \text{OUT} \) occurs) {
    for (each basic block \( \text{B} \) other than \( \text{ENTRY} \)) {
        \( \text{IN}[\text{B}] = \cup \text{P} \) a predecessor of \( \text{B} \) \( \text{OUT}[\text{P}] \)
        \( \text{OUT}[\text{B}] = \text{gen}_\text{B} \cup (\text{IN}[\text{B}] - \text{kill}_\text{B}) \)
    }
}

d1: \( i = m - 1 \)
d2: \( j = n \)
d3: \( a = u_1 \)
d4: \( i = i + 1 \)
d5: \( j = j - 1 \)
d6: \( a = u_2 \)
d7: \( i = u_3 \)

\[ \text{ENTRY} \]

\[ \text{EXIT} \]
\[
\text{OUT}[\text{ENTRY}] = \emptyset
\]
for (each basic block \( B \) other than \( \text{ENTRY} \)) \{ \text{OUT}[B] = \emptyset \} 
while (changes to any OUT occurs) \{ 
    for (each basic block \( B \) other than \( \text{ENTRY} \)) \{ 
        \text{IN}[B] = \bigcup \{ \text{OUT}[\text{P}] \mid \text{P} \text{ a predecessor of } B \} 
        \text{OUT}[B] = \text{gen}_B \cup ( \text{IN}[B] - \text{kill}_B ) 
    \}
\}

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
& \text{OUT}[B]_0 & \text{IN}[B]_1 & \text{OUT}[B]_1 & \text{IN}[B]_2 & \text{OUT}[B]_2 \\
\hline
\text{B1} & 000 0000 & 000 0000 & 111 0000 & 000 0000 & 111 0000 \\
\hline
\text{B2} & 000 0000 & 111 0000 & 001 1100 & 111 0111 & 001 1110 \\
\hline
\text{B3} & 000 0000 & 001 1100 & 000 1110 & 001 1110 & 000 1110 \\
\hline
\text{B4} & 000 0000 & 001 1110 & 001 0111 & 001 1110 & 001 0111 \\
\hline
\text{EXIT} & 000 0000 & 001 0111 & 001 0111 & & \\
\hline
\end{array}
\]
OUT[ENTRY] = ∅
for (each basic block B other than ENTRY) { OUT[B] = ∅ }
while (changes to any OUT occurs) {
    for (each basic block B other than ENTRY) {
        IN[B] = \bigcup P \text{ a predecessor of } B \text{ OUT}[P]
        OUT[B] = gen_B \cup (IN[B] - kill_B)
    }
}
9.2.4 Reaching definitions

Useful for constant propagation and constant folding (§8.5.4 - p. 536, §9.4 - p. 632). Additional discussion and examples:

en.wikipedia.org/wiki/Constant_folding

Useful for global common subexpression elimination (§9.1.4 - p. 588, §9.2.6 - p. 610, §9.5 - p. 639). Additional discussion and examples:

en.wikipedia.org/wiki/Common_subexpression_elimination
9.2.5 Live variable analysis

Useful for effective register management.

"After a value is computed in a register, and presumably used within a block, it is not necessary to store that value if it is dead at the end of the block. Also, if all registers are full and we need another register, we should favor using a register with a dead value, since that value does not have to be stored." [p. 608]
9.2.5 Live variable analysis

"In live variable analysis we wish to know for variable \( x \) and point \( p \) whether the value of \( x \) at \( p \) could be used along some path in the flow graph starting at \( p \). If so, we say \( x \) is live at \( p \); otherwise, \( x \) is dead at \( p \)." [p. 608]

In contrast to reaching analysis, which used a forward transfer function, live variable analysis uses a backward transfer function.
9.2.5 Live variable analysis

definitions, page 609

def$B$ is "the set of variables defined in $B$ prior to any use of that variable in $B$"

use$B$ is "the set of variables whose values may be used in $B$ prior to any definition of the variable"
9.2.5 Live variable analysis

definitions, page 609

\( \text{IN[EXIT]} = \emptyset \)

\( \text{IN}[B] = \text{use}_B \cup (\text{OUT}[B] - \text{def}_B) \)

\( \text{OUT}[B] = \bigcup_{S \text{ a successor of } B} \text{IN}[S] \)
9.2.5 Live variable analysis

Algorithm [p. 610]

INPUT: A flow graph with def and use computed for each block.

OUTPUT: IN[B] and OUT[B], the set of variables live on entry and exit of each block of the flow graph.

METHOD:

\[ \text{IN[EXIT]} = \emptyset \]

\[
\text{for (each basic block B other than EXIT) } \{ \text{IN[B]} = \emptyset \} \\
\text{while (changes to any IN occur) } \{
\text{for (each basic block B other than EXIT) } \{
\text{OUT[B]} = \bigcup_{S \text{ a successor of B}} \text{IN[S]} \\
\text{IN[B]} = \text{use}_B \cup (\text{OUT[B]} - \text{def}_B)
\}
\}
\]
9.2.6 Available expressions

"An expression $x+y$ is available at a point $p$ if every path from the entry node to $p$ evaluates to $x+y$, and after the last such evaluation prior to reaching $p$, there are no subsequent assignments to $x$ or $y."$ [p. 610]
9.2.6 Available expressions

"...a block kills expression x+y if it assigns (or may assign) x or y and does not subsequently recompute x+y." [p. 610]

"A block generates expression x+y if it definitely evaluates x+y and does not subsequently define x or y." [p. 611]
...the expression $4 \times i$ in block B3 will be a common subexpression if $4 \times i$ is available at the entry point of block B3.

[p 611]
Figure 9.17

\[ t_1 = 4 \times i \]

\[ t_2 = 4 \times i \]

"It will be available if \( i \) is not assigned a new value in block B2, ..." [p 611]

Here \( 4 \times i \) in B3 can be replaced by value of \( t_1 \), regardless of which branch is taken.
"... or if ... 4 * i is recomputed after i is assigned in B2." [p 611]

Again, 4 * i in B3 can be replaced by value of t1, regardless of which branch is taken (since t1 contains the correct value of 4 * i in both cases)
9.2.6 Available expressions

Informally:

"If at point $p$ set $S$ of expressions is available, and $q$ is the point after $p$, with statement $x=y+z$ between them, then we form the set of expressions available at $q$ by the following steps:

1. Add to $S$ the expression $y+z$.
2. Delete from $S$ any expression involving variable $x$.

[p. 611]
<table>
<thead>
<tr>
<th>Statement</th>
<th>Available expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = b + c$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$b = a - d$</td>
<td>${ a - d }$</td>
</tr>
<tr>
<td>$c = b + c$</td>
<td>${ a - d }$</td>
</tr>
<tr>
<td>$d = a - d$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
9.2.6 Available expressions

“We can find available expressions in a manner reminiscent of the way reaching definitions are computed. Suppose U is the ‘universal’ set of all expressions appearing on the right of one or more statement of the program. For each block B, let IN[B] be the set of expressions in U that are available at the point just before the beginning of B. Let OUT[B] be the same for the point following the end of B. Define e_gen to be the expressions generated by B and e_kill to be the set of expressions in U killed in B. Note that IN, OUT, e_gen, and e_kill can all be represented by bit vectors.” [p. 612]
9.2.6 Available expressions definitions, page 612

\( \text{OUT}[\text{ENTRY}] = \emptyset \)

\( \text{OUT}[B] = \text{e}_{\text{gen}_B} \cap (\text{IN}[B] - \text{e}_{\text{kill}_B}) \)

\( \text{IN}[B] = \bigcap_{\text{P a predecessor of } B} \text{OUT}[P] \)
9.2.6 Available expressions
definitions, page 612

\[ \text{OUT[ENTRY]} = \emptyset \]

\[ \text{OUT[B]} = e_{genB} \cap (\text{IN[B]} - e_{killB}) \]

\[ \text{IN[B]} = \bigcap_P \text{a predecessor of } B \text{ } \text{OUT[P]} \]

Note use of \( \cap \) rather than \( \cup \).

"...an expression is available at the beginning of a block only if it is available at the end of ALL its predecessors." [p. 612]
9.2.6 Available expressions

Algorithm [p. 614]

INPUT: A flow graph with $e_{\text{kill}}_B$ and $e_{\text{gen}}_B$ computed for each block $B$. The initial block is $B_1$.

OUTPUT: $\text{IN}[B]$ and $\text{OUT}[B]$, the set of expressions available at the entry and exit of each block of the flow graph.

METHOD:

\[
\text{OUT}[\text{ENTRY}] = \emptyset
\]

\begin{align*}
\text{for (each basic block } B \text{ other than ENTRY)} & \{ \text{ OUT}[B] = \emptyset \} \\
\text{while (changes to any OUT occur)} & \{ \\
\text{ for (each basic block } B \text{ other than EXIT)} & \{ \\
\text{ IN}[B] & = \bigcap \{ \text{ a predecessor of } B \} \text{ OUT}[] \\
\text{ OUT}[B] & = e_{\text{gen}}_B \cap (\text{IN}[B] - e_{\text{kill}}_B) \\
\} \}
\end{align*}
9.2.6 Available expressions

Algorithm [p. 614]

INPUT: A flow graph with $e_{\text{kill}}_B$ and $e_{\text{gen}}_B$ computed for each block $B$. The initial block is $B_1$.

OUTPUT: $\text{IN}[B]$ and $\text{OUT}[B]$, the set of expressions available at the entry and exit of each block of the flow graph.

METHOD:

\[
\begin{align*}
\text{OUT}[\text{ENTRY}] &= \emptyset \\
\text{for}\ (\text{each basic block } B \text{ other than } \text{ENTRY}) &\{ \text{OUT}[B] = U \} \\
\text{while}\ (\text{changes to any OUT occur}) &\{ \\
&\quad \text{for}\ (\text{each basic block } B \text{ other than } \text{EXIT}) \{ \\
&\qquad \text{IN}[B] = \bigcap P \text{ a predecessor of } B\ \text{OUT}[P] \\
&\qquad \text{OUT}[B] = e_{\text{gen}}_B \cap (\text{IN}[B] - e_{\text{kill}}_B) \\
&\quad \} \\
&\} \\
\end{align*}
\]

Recall: $U$ is set of all expressions
## 9.2 Summary

<table>
<thead>
<tr>
<th>Domain</th>
<th>Reaching definitions</th>
<th>Live variables</th>
<th>Available expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>sets of definitions</td>
<td>sets of variables</td>
<td>sets of expressions</td>
</tr>
<tr>
<td></td>
<td>forward</td>
<td>backward</td>
<td>forward</td>
</tr>
<tr>
<td>Transfer function</td>
<td>( \text{gen}_B \cup (x - \text{kill}_B) )</td>
<td>( \text{use}_B \cup (x - \text{def}_B) )</td>
<td>( \text{e-gen}_B \cap (x - \text{e-kill}_B) )</td>
</tr>
<tr>
<td>Boundary</td>
<td>( \text{OUT}[\text{ENTRY}] = \emptyset )</td>
<td>( \text{IN}[\text{EXIT}] = \emptyset )</td>
<td>( \text{OUT}[\text{ENTRY}] = \emptyset )</td>
</tr>
<tr>
<td>Meet (( \land ))</td>
<td>( \cup )</td>
<td>( \cup )</td>
<td>( \cap )</td>
</tr>
<tr>
<td>Equations</td>
<td>( \text{OUT}[B] = f_B(\text{IN}[B]) ) ( \text{IN}[B] = \land_{p,\text{pred}(B)} \text{OUT}[P] )</td>
<td>( \text{IN}[B] = f_B(\text{OUT}[B]) ) ( \text{OUT}[B] = \land_{s,\text{succ}(B)} \text{IN}[S] )</td>
<td>( \text{OUT}[B] = f_B(\text{IN}[B]) ) ( \text{IN}[B] = \land_{p,\text{pred}(B)} \text{OUT}[P] )</td>
</tr>
<tr>
<td>Initialize</td>
<td>( \text{OUT}[B] = \emptyset )</td>
<td>( \text{IN}[B] = \emptyset )</td>
<td>( \text{OUT}[B] = \text{U} )</td>
</tr>
</tbody>
</table>
9.4 Constant Propagation

- Constant propagation is a forward data-flow problem.
- Lattice consists of:
  - All constants appropriate for the type of the variable.
  - The value NAC ("not a constant") - we know the variable is not a constant.
  - The value UNDEF ("undefined") - we don't know the variable's status.

[p. 633]
9.4.3 Transfer functions

Reminders:
- $f_s$ is transfer function for statement $s$
- $m' = f_s(m)$

"If $s$ is not an assignment statement, $f_s$ is simply the identity function."

"If $s$ is an assignment to variable $x$, then $m'(v) = m(v)$, for all variables $v \neq x$, and $m'(x)$ is defined as follows:

- If the right-hand-side (RHS) of the statement $s$ is a constant $c$, then $m'(x) = c$.
- If the RHS is of the form $y+z$, then $m'(x) = m(y) + m(z)$ if $m(y)$ and $m(z)$ are constant values, NAC if either $m(y)$ or $m(z)$ is NAC, UNDEF otherwise.
- If the RHS is any other expression (e.g. a function call or assignment through a pointer), then $m'(x) = \text{NAC}$.

[p. 634]
9.4.4 Monotonicity (p. 635)

\[
\begin{array}{ccccc}
\vdots & c_1 & \rightarrow & c_2 & \leftarrow & \cdots & \leftarrow & c_n \\
\downarrow & & & & & & & \downarrow \\
& (UNDEF) & & (NAC) \end{array}
\]
### 9.4.4 Monotonicity (p. 635)

<table>
<thead>
<tr>
<th>(m(y))</th>
<th>(m(z))</th>
<th>(m'(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDEF</td>
<td>UNDEF</td>
<td>UNDEF</td>
</tr>
<tr>
<td>UNDEF</td>
<td>(c_2)</td>
<td>UNDEF</td>
</tr>
<tr>
<td>UNDEF</td>
<td>NAC</td>
<td>NAC</td>
</tr>
<tr>
<td>(c_1)</td>
<td>UNDEF</td>
<td>UNDEF</td>
</tr>
<tr>
<td>(c_1)</td>
<td>(c_2)</td>
<td>(c_1 + c_2)</td>
</tr>
<tr>
<td>(c_1)</td>
<td>NAC</td>
<td>NAC</td>
</tr>
<tr>
<td>NAC</td>
<td>UNDEF</td>
<td>NAC</td>
</tr>
<tr>
<td>NAC</td>
<td>(c_2)</td>
<td>NAC</td>
</tr>
<tr>
<td>NAC</td>
<td>NAC</td>
<td>NAC</td>
</tr>
</tbody>
</table>
9.5 Partial-Redundancy Elimination

Idea: reduce number of (run time) evaluations of an expression.

Can move code to evaluate expression around.

May result in more occurrences of the expression, but fewer evaluations.
9.5.1 Sources of redundancy

"...three forms of redundancy: common subexpressions, loop-invariant expressions, and partially redundant expressions." [p. 639]
Fig 9.30 (a)

Global common subexpression
Fig 9.30 (b)

Loop-invariant code motion

evaluation inside loop

evaluation outside loop

t = b+c

a = t

a = b+c
Fig 9.30 (b)

Must be careful: don’t evaluate $b+c$ in optimized code if not evaluated in non-optimized code: might throw exception

if loop doesn’t run, takes more time
Fig 9.30 (b)

while exp {
  ...
  a = b + c;
  ...
}

is translated as

if exp {
  t = b + c;
  repeat {
    ...
    a = t;
    ...
  } until not exp;
}
Fig 9.30 (c)

partial redundancy elimination
9.5.2 Can all redundancy be eliminated?

9.5. PARTIAL-REDUNDANCY ELIMINATION

Critical edge:

"...any edge leading from a node with more than one successor node to a node with more than one predecessor."

Example 9.31: $B_3 \rightarrow B_4$ is a critical edge