Phases of a compiler

Figure 1.6, page 5 of text

Optimizations
Optimization

- The semantics of a program **must** be preserved by optimizations.
- The compiler does not know a programmer’s intent - it can only reason about the program as written.
Data-flow analysis

- View program execution as a sequence of state transformations.
- Each program state consists of all the variables in the program along with their current values.
State transformation

input state

intermediate instruction

output state

prog state

prog state
State transformation

Program states are called program points.

A sequence of program points is called a path.
Data-flow analysis

- Begin by considering only the flow graph for a single function.
Properties

- Within a basic block:
  - Program point after a statement is same as program point before the next statement.
- Why?
Properties

- Between basic blocks:

  "If there is an edge from block B1 to block B2, then the program point after the last statement of B1 may be followed immediately by the program point before the first statement of B2."

[p. 597]
Execution path

"An execution path (or just path) from point $p_1$ to point $p_n$ [is] a sequence of points $p_1, p_2, \ldots, p_n$ such that for each $i = 1, 2, \ldots, n-1$, either

1. $p_i$ is the point immediately preceding a statement and $p_{i+1}$ is the point immediately following that same statement, or

2. $p_i$ is the end of some block and $p_{i+1}$ is the beginning of a successor block."

[p. 597]
Example 9.8 (p. 598)

(1) \[ \text{d1: } a = 1 \]
(2) \[ \text{if read()} \leq 0 \text{ goto B4} \]
(3) \[ \text{d2: } b = a \]
(4) \[ \text{d3: } a = 243 \]
(5) \[ \text{goto B2} \]

Path: (1,2,3,4,9)

Path: (1,2,3,4,5,6,7,8,3,4,9)

a has value 1 first time (5) is executed.

d1 reaches (5) on the first iteration.

a has value 243 at (5) on the second and subsequent iterations.

d3 reaches (5) on those iterations.
Reaching definitions

"The definitions that may reach a program point along some path are known as reaching definitions." [p. 598]
Gathering different data for different uses

to determine possible values

"... at point (5) ... the value of \(a\) is one of \{ 1 , 243 \} and ... it may be defined by one of \{ d1 , d3 \}.

[p. 598]

"... at point (5) ... there is no definition that must be the definition of \(a\) at that point, so this set is empty for \(a\) at point (5). Even if a variable has a unique definition at a point, that definition must assign a constant to the variable. Thus, we may simply describe certain variables as 'not a constant', instead of collecting all their possible values or all their possible definitions."

[p. 599]

for 'constant folding'
9.2.2 Data-flow analysis schema

"In each application of data-flow analysis, we associate with every program point a data-flow value that represents an abstraction of the set of all possible program states that can be observed at that point."  [p. 599]

"The set of possible data-flow values is the domain..."  [p. 599]

"We denote the data-flow values before and after each statement s by IN[s] and OUT[s], respectively."  [p. 599]
9.2.2 Data-flow analysis schema

"The data-flow problem is to find a solution to a set of constraints on the IN[s]'s and OUT[s]'s, for all statements s. There are two sets of constraints: those based on the semantics of the statements ("transfer functions") and those based on the flow of control." [p. 599]
Transfer functions

Information can flow forwards or backwards.

**Forward flow:** $\text{OUT}[s] = f_s ( \text{IN}[s] )$

**Backward flow:** $\text{IN}[s] = g_s ( \text{OUT}[s] )$
Control flow constraints

In a sequence $s_1, s_2, \ldots, s_n$ without jumps,

$$\text{IN}[s_{i+1}] = \text{OUT}[s_i] \text{ for all } i=1,2,\ldots,n-1$$

For data-flow between blocks, take "the union of the definitions after last statements of each of the predecessor blocks." [p. 600]
9.2.3 Data-flow schemas on basic blocks

Suppose a basic block $B$ consists of the sequence of statements $s_1, s_2, ..., s_n$. Define $\text{IN}[B] = \text{IN}[s_1]$ and $\text{OUT}[B] = \text{OUT}[s_n]$.

The transfer function of $B$: $f_B = f_{s_n} \circ ... \circ f_{s_2} \circ f_{s_1}$

The transfer function of $B$: $\text{OUT}[B] = f_B(\text{IN}[B])$
9.2.3 Data-flow schemas on basic blocks

Forward flow problem

\[ \text{OUT}[B] = f_B(\text{IN}[B]) \]
\[ \text{IN}[B] = \bigcup_{P \text{ a predecessor of } B} \text{OUT}[P] \]

Backward flow problem

\[ \text{IN}[B] = g_B(\text{OUT}[B]) \]
\[ \text{OUT}[B] = \bigcup_{S \text{ a successor of } B} \text{IN}[S] \]
9.2.3 Data-flow schemas on basic blocks

"...data-flow equations usually do not have a unique solution. Our goal is to find the most 'precise' solution that satisfies the two sets of constraints: control-flow and transfer constraints. That is, we need a solution that encourages valid code improvements, but does not justify unsafe transformations..."

[p. 601]
9.2.4 Reaching definitions

“A definition d reaches a point p if there is a path from the point immediately following d to p, such that d is not ‘killed’ along that path.” [p. 601]

“We kill a definition of a variable x if there is any other definition of x anywhere along the path.” [p. 601]
9.2.4 Reaching definitions

"A definition of a variable $x$ is a statement that assigns, or may assign, a value to $x$.”

What is meant by "may assign"?
9.2.4 Reaching definitions

"Procedure parameters, array accesses, and indirect references all may have aliases, and it is not easy to tell if a statement is referring to a particular variable x." [p. 601]

"Program analysis must be conservative" [p. 601]
Transfer equations for reaching definitions

For this definition:

\[ d: u = v + \omega \]

The transfer equation is:

\[ f_d(\sigma) = \text{gend} \cup (\sigma - \text{kill}_d) \]

where \( \text{gend} = \{d\} \). \( \text{kill}_d \) is the set of all other definitions of \( u \) in the program.

The argument of a transfer function is a data-flow value, which "represents an abstraction of the set of all possible program states that can be observed for that point." [p. 599]

Recall too that a program state consists of all the variables in the program along with their current values.
Figure 9.13  
(p. 604)

\[ d_1: i = m - 1 \]
\[ d_2: j = n \]
\[ d_3: a = u_1 \]

\[ d_4: i = i + 1 \]
\[ d_5: j = j - 1 \]

\[ d_6: a = u_2 \]

\[ d_7: i = u_3 \]
Figure 9.13 (p. 604)

\begin{align*}
\text{d1: } & i = m - 1 \\
\text{d2: } & j = n \\
\text{d3: } & a = u1 \\
\text{d4: } & i = i + 1 \\
\text{d5: } & j = j - 1 \\
\text{d6: } & a = u2 \\
\text{d7: } & i = u3 \\
\end{align*}

\text{gen}_B1 = \{ \text{d1, d2, d3} \} \\
\text{kill}_B1 = \{ ? \} \\

\text{gen}_B2 = \{ ? \} \\
\text{kill}_B2 = \{ ? \} \\

\text{gen}_B3 = \{ ? \} \\
\text{kill}_B3 = \{ ? \} \\

\text{gen}_B4 = \{ ? \} \\
\text{kill}_B4 = \{ ? \}
**Figure 9.13**

(p. 604)

\[
\begin{align*}
d1: & \quad i = m - 1 \\
d2: & \quad j = n \\
d3: & \quad a = u_1 \\
d4: & \quad i = i + 1 \\
d5: & \quad j = j - 1 \\
d6: & \quad a = u_2 \\
d7: & \quad i = u_3
\end{align*}
\]

\[
\begin{align*}
\text{gen}_{B1} &= \{ d1, d2, d3 \} \\
\text{kill}_{B1} &= \{ d4, d5, d6, d7 \}
\end{align*}
\]

\[
\begin{align*}
\text{gen}_{B2} &= \{ ? \} \\
\text{kill}_{B2} &= \{ ? \}
\end{align*}
\]

\[
\begin{align*}
\text{gen}_{B3} &= \{ ? \} \\
\text{kill}_{B3} &= \{ ? \}
\end{align*}
\]

\[
\begin{align*}
\text{gen}_{B4} &= \{ ? \} \\
\text{kill}_{B4} &= \{ ? \}
\end{align*}
\]
Q: Why kill d4 – d7 here, since they are not on a path to B1?

\[ \begin{align*}
B1 & : \text{gen}_{B1} = \{ d1, d2, d3 \} \\
& \quad \text{kill}_{B1} = \{ d4, d5, d6, d7 \} \\
B2 & : \text{gen}_{B2} = \{ ? \} \\
& \quad \text{kill}_{B2} = \{ ? \} \\
B3 & : \text{gen}_{B3} = \{ ? \} \\
& \quad \text{kill}_{B3} = \{ ? \} \\
B4 & : \text{gen}_{B4} = \{ ? \} \\
& \quad \text{kill}_{B4} = \{ ? \}
\end{align*}\]
Q: Why kill d4 - d7 here, since they are not on a path to B1?

A: Here we are looking just at this block, and not trying to account for flow between blocks.

Inter-block flow is taken into account later.
Figure 9.13 (p. 604)

\[
\begin{align*}
d_1: & \quad i = m - 1 \\
d_2: & \quad j = n \\
d_3: & \quad a = u_1 \\
d_4: & \quad i = i + 1 \\
d_5: & \quad j = j - 1 \\
d_6: & \quad a = u_2 \\
d_7: & \quad i = u_3
\end{align*}
\]

\[\text{gen}_{B1} = \{ d_1, d_2, d_3 \}\]
\[\text{kill}_{B1} = \{ d_4, d_5, d_6, d_7 \}\]

\[\text{gen}_{B2} = \{ ? \}\]
\[\text{kill}_{B2} = \{ ? \}\]

\[\text{gen}_{B3} = \{ ? \}\]
\[\text{kill}_{B3} = \{ ? \}\]

\[\text{gen}_{B4} = \{ ? \}\]
\[\text{kill}_{B4} = \{ ? \}\]

ENTRY

EXIT
**Figure 9.13 (p. 604)**

\[
\begin{align*}
\text{ENTRY} \\
\downarrow \\
\text{B1} \\
\text{d1: } i &= m - 1 \\
\text{d2: } j &= n \\
\text{d3: } a &= u1 \\
\downarrow \\
\text{B2} \\
\text{d4: } i &= i + 1 \\
\text{d5: } j &= j - 1 \\
\downarrow \\
\text{B3} \\
\text{d6: } a &= u2 \\
\downarrow \\
\text{B4} \\
\text{d7: } i &= u3 \\
\downarrow \\
\text{EXIT}
\end{align*}
\]

\[
\begin{align*}
\text{gen}_{\text{B1}} &= \{ \text{d1, d2, d3} \} \\
\text{kill}_{\text{B1}} &= \{ \text{d4, d5, d6, d7} \} \\
\text{gen}_{\text{B2}} &= \{ \text{d4, d5} \} \\
\text{kill}_{\text{B2}} &= \{ \text{?} \} \\
\text{gen}_{\text{B3}} &= \{ \text{?} \} \\
\text{kill}_{\text{B3}} &= \{ \text{?} \} \\
\text{gen}_{\text{B4}} &= \{ \text{?} \} \\
\text{kill}_{\text{B4}} &= \{ \text{?} \}
\end{align*}
\]
Figure 9.13 (p. 604)

\begin{align*}
\text{d1: } & i = m - 1 \\
\text{d2: } & j = n \\
\text{d3: } & a = u1 \\
\text{d4: } & i = i + 1 \\
\text{d5: } & j = j - 1 \\
\text{d6: } & a = u2 \\
\text{d7: } & i = u3 \\
\end{align*}

\textbf{B1:}

\text{gen}_{B1} = \{ \text{d1, d2, d3} \}
\quad \text{kill}_{B1} = \{ \text{d4, d5, d6, d7} \}

\textbf{B2:}

\text{gen}_{B2} = \{ \text{d4, d5} \}
\quad \text{kill}_{B2} = \{ \text{d1, d2, d7} \}

\textbf{B3:}

\text{gen}_{B3} = \{ ? \}
\quad \text{kill}_{B3} = \{ ? \}

\textbf{B4:}

\text{gen}_{B4} = \{ ? \}
\quad \text{kill}_{B4} = \{ ? \}
Figure 9.13
(p. 604)

\[ d_1: i = m - 1 \]
\[ d_2: j = n \]
\[ d_3: a = u_1 \]
\[ d_4: i = i + 1 \]
\[ d_5: j = j - 1 \]
\[ d_6: a = u_2 \]
\[ d_7: i = u_3 \]

\[ \text{gen}_{B_1} = \{ d_1, d_2, d_3 \} \]
\[ \text{kill}_{B_1} = \{ d_4, d_5, d_6, d_7 \} \]

\[ \text{gen}_{B_2} = \{ d_4, d_5 \} \]
\[ \text{kill}_{B_2} = \{ d_1, d_2, d_7 \} \]

\[ \text{gen}_{B_3} = \{ d_6 \} \]
\[ \text{kill}_{B_3} = \{ ? \} \]

\[ \text{gen}_{B_4} = \{ ? \} \]
\[ \text{kill}_{B_4} = \{ ? \} \]
Figure 9.13 (p. 604)

\[
\begin{align*}
\text{d1: } i &= m - 1 \\
\text{d2: } j &= n \\
\text{d3: } a &= u_1 \\
\text{d4: } i &= i + 1 \\
\text{d5: } j &= j - 1 \\
\text{d6: } a &= u_2 \\
\text{d7: } i &= u_3
\end{align*}
\]

\[
\begin{align*}
gen_{B1} &= \{ \text{d1, d2, d3} \} \\
\text{kill}_{B1} &= \{ \text{d4, d5, d6, d7} \} \\
gen_{B2} &= \{ \text{d4, d5} \} \\
\text{kill}_{B2} &= \{ \text{d1, d2, d7} \} \\
gen_{B3} &= \{ \text{d6} \} \\
\text{kill}_{B3} &= \{ \text{d3} \} \\
gen_{B4} &= \{ ? \} \\
\text{kill}_{B4} &= \{ ? \}
\end{align*}
\]
Figure 9.13 (p. 604)

\[ d_1: i = m - 1 \]
\[ d_2: j = n \]
\[ d_3: a = u_1 \]

\[ d_4: i = i + 1 \]
\[ d_5: j = j - 1 \]

\[ d_6: a = u_2 \]

\[ d_7: i = u_3 \]
Figure 9.13 (p. 604)

\[
d1: i = m - 1
\]
\[
d2: j = n
\]
\[
d3: a = u_1
\]

\[
d4: i = i + 1
\]
\[
d5: j = j - 1
\]

\[
d6: a = u_2
\]

\[
d7: i = u_3
\]

\[
gen_{B1} = \{ d1, d2, d3 \}
\]
\[
kill_{B1} = \{ d4, d5, d6, d7 \}
\]

\[
gen_{B2} = \{ d4, d5 \}
\]
\[
kill_{B2} = \{ d1, d2, d7 \}
\]

\[
gen_{B3} = \{ d6 \}
\]
\[
kill_{B3} = \{ d3 \}
\]

\[
gen_{B4} = \{ d7 \}
\]
\[
kill_{B4} = \{ d1, d4 \}
\]

ENTRY

EXIT
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:

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

See footnote 4 on page 606
Iterative algorithm for reaching definitions

Algorithm [p. 606]

INPUT: A flow graph for which kill\(B\) and gen\(B\) have been computed for each block \(B\).

OUTPUT: IN\([B]\) and OUT\([B]\), the set of definitions reaching the entry and exit of each block \(B\) of the flow graph.

METHOD:

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

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

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

\[
\begin{align*}
gen_{B1} &= \{ d1, d2, d3 \} \\
\text{kill}_{B1} &= \{ d4, d5, d6, d7 \} \\

d1: i &= m - 1 \\
d2: j &= n \\
d3: a &= u1 \\

gen_{B2} &= \{ d4, d5 \} \\
\text{kill}_{B2} &= \{ d1, d2, d7 \} \\
d4: i &= i + 1 \\
d5: j &= j - 1 \\

gen_{B3} &= \{ d6 \} \\
\text{kill}_{B3} &= \{ d3 \} \\
d6: a &= u2 \\

gen_{B4} &= \{ d7 \} \\
\text{kill}_{B4} &= \{ d1, d4 \} \\
d7: i &= u3
\end{align*}
\]
Example 9.12 - building off figure 9.13

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

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 \( \text{ENTRY} \)) \{
    \( \text{IN}[B] = \cup \text{a predecessor of } B \ \text{OUT}[P] \)
    \( \text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B) \)
  \}
\}

Example 9.12 - building off figure 9.13

<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>( \text{B1} )</td>
<td>Represent ( d_i ) as a bit vector, where each ( d ) is a definition from 9.13</td>
<td>Union of sets ( A \cup B: A \text{ OR } B )</td>
<td>Difference of sets ( A - B: A \text{ AND } B' )</td>
<td>Compute in order ( \text{B1, B2, B3, B4, EXIT} )</td>
<td>For example:</td>
</tr>
<tr>
<td>( \text{B2} )</td>
<td>( \text{IN}[B2]^1 = \text{OUT}[B1]^1 \cup \text{OUT}[B4]^0 = 111 \ 0000 \cup 000 \ 0000 = 111 \ 0000 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{B3} )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{B4} )</td>
<td>( \text{OUT}[B2]^1 = \text{gen}<em>{B2} \cup (\text{IN}[B2]^1 - \text{kill}</em>{B2}) )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{EXIT} )</td>
<td>( = 000 \ 1100 + (111 \ 0000 - 110 \ 0001) )</td>
<td></td>
<td></td>
<td></td>
<td>( = 000 \ 1100 + 001 \ 0000 = 001 \ 1100 )</td>
</tr>
</tbody>
</table>

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
Example 9.12 - building off figure 9.13

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

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) {
        $$\text{IN}[B] = \cup \text{ a predecessor of } B \, \text{OUT}[P]$$
        $$\text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B)$$
    }
}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B2</td>
<td>000 0000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B3</td>
<td>000 0000</td>
<td></td>
<td></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

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

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

\[
\begin{array}{|c|c|c|c|c|}
\hline
& \text{OUT}^1[B] & \text{IN}^1[B] & \text{OUT}^1[B] & \text{IN}^2[B] & \text{OUT}^2[B] \\
\hline
B1 & 000 0000 & 000 0000 & 111 0000 & \text{IN}[B1] = \text{pred}(B1) = \text{ENTRY} \\
B2 & 000 0000 & \text{IN}[B1] = \text{pred}(B1) = \text{ENTRY} & \text{OUT}[B1] = \text{gen}_{B1} \cup (\text{IN}[B1] - \text{kill}_{B1}) \\
B3 & 000 0000 & \text{IN}[B1] = \text{pred}(B1) = \text{ENTRY} & \text{OUT}[B1] = \text{gen}_{B1} \cup (\text{IN}[B1] - \text{kill}_{B1}) \\
B4 & 000 0000 & \text{IN}[B1] = \text{pred}(B1) = \text{ENTRY} & \text{OUT}[B1] = \text{gen}_{B1} \cup (\text{IN}[B1] - \text{kill}_{B1}) \\
\text{EXIT} & 000 0000 & \text{IN}[B1] = \text{pred}(B1) = \text{ENTRY} & \text{OUT}[B1] = \text{gen}_{B1} \cup (\text{IN}[B1] - \text{kill}_{B1}) \\
\hline
\end{array}
\]
Example 9.12

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

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

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

\[
\text{IN}[B2] = \text{pred}(B2) = \text{OUT}[B1] \cup \text{OUT}[B4] \\
\text{OUT}[B2] = \text{gen}_{B2} \cup ( \text{IN}[B2] - \text{kill}_{B2} ) \\
\text{gen}_{B2} = \{ \text{d}4, \text{d}5 \} \\
\text{kill}_{B2} = \{ \text{d}1, \text{d}2, \text{d}7 \}
\]
Example 9.12

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

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

\[
\begin{array}{|c|c|c|c|c|}
\hline
 & \text{OUT}[B]^0 & \text{IN}[B]^1 & \text{OUT}[B]^1 & \text{IN}[B]^2 & \text{OUT}[B]^2 \\
\hline
B1 & 000 0000 & 000 0000 & 111 0000 &  & \\
\hline
B2 & 000 0000 & 111 0000 & 001 1100 &  & \\
\hline
B3 & 000 0000 & 001 1100 & 000 1110 &  & \\
\hline
B4 & 000 0000 & \text{IN}[B3] = \text{pred}(B3) = \text{OUT}[B2] & \text{OUT}[B3] = \text{gen}_{B3} \cup (\text{IN}[B3] - \text{kill}_{B3}) &  & \\
\hline
\text{EXIT} & 000 0000 & \text{gen}_{B3} = \{ d6 \} & \text{kill}_{B3} = \{ d3 \} &  & \\
\hline
\end{array}
\]
Example 9.12

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

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

while (changes to any OUT occurs) \{ 

\hspace{1em} for (each basic block B other than ENTRY) \{ 

\hspace{2em} \text{IN}[B] = \bigcup \text{P a predecessor of } B \text{ OUT}[P] 

\hspace{2em} \text{OUT}[B] = \text{gen}_{B} \cup (\text{IN}[B] - \text{kill}_{B}) 

\hspace{1em} \} 

\} 

\[ \begin{array}{|c|c|c|c|}
\hline
\text{B1} & \text{OUT}[B]^{0} & \text{IN}[B]^{1} & \text{OUT}[B]^{1} \\
\hline
000 & 00000 & 000 & 00000 \\
\hline
\text{B2} & \text{OUT}[B]^{0} & \text{IN}[B]^{1} & \text{OUT}[B]^{1} \\
\hline
000 & 00000 & 111 & 00000 \\
\hline
\text{B3} & \text{OUT}[B]^{0} & \text{IN}[B]^{1} & \text{OUT}[B]^{1} \\
\hline
000 & 00000 & 001 & 1100 \\
\hline
\text{B4} & \text{OUT}[B]^{0} & \text{IN}[B]^{1} & \text{OUT}[B]^{1} \\
\hline
000 & 00000 & 001 & 1110 \\
\hline
\text{EXIT} & \text{OUT}[B]^{0} & \text{IN}[B]^{1} & \text{OUT}[B]^{1} \\
\hline
000 & 00000 & \text{IN}[B4] = \text{OUT}[B2] \cup \text{OUT}[B3] \\
\hline
\end{array} \]

\[ \text{gen}_B = \{ \text{d7} \} \]
\[ \text{kill}_B = \{ \text{d1, d4} \} \]
Example 9.12

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] = ∪P a predecessor of B OUT[P]
        OUT[B] = genB ∪ (IN[B] - killB)
    }
}

---|---|---|---|---
B1 | 000 0000 | 000 0000 | 111 0000 | |
B2 | 000 0000 | 111 0000 | 001 1100 | |
B3 | 000 0000 | 001 1100 | 000 1110 | |
B4 | 000 0000 | 001 1110 | 001 0111 | |
EXIT | 000 0000 | 001 0111 | 001 0111 | |

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

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

\[ \text{d1: } i = m - 1 \]
\[ \text{d2: } j = n \]
\[ \text{d3: } a = u_1 \]
\[ \text{d4: } i = i + 1 \]
\[ \text{d5: } j = j - 1 \]
\[ \text{d6: } a = u_2 \]
\[ \text{d7: } i = u_3 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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></td>
<td></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 1110</td>
<td>001 0111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td>000 0000</td>
<td>001 0111</td>
<td>001 0111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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] = ∪ P a predecessor of B OUT[P]
        OUT[B] = gen_B ∪ (IN[B] - kill_B)
    }
}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></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 1110</td>
<td>001 0111</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td>000 0000</td>
<td>001 0111</td>
<td>001 0111</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ \text{OUT[ENTRY]} = \emptyset \]

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 \( \text{ENTRY} \)) {
        \( \text{IN}[B] = \cup P \) a predecessor of \( B \) \( \text{OUT}[P] \)
        \( \text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B) \)
    }
}

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

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 \( \text{ENTRY} \)) {
        \( \text{IN}[B] = \cup \text{a predecessor of } B \ \text{OUT}[P] \)
        \( \text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B) \)
    }
}

<table>
<thead>
<tr>
<th>( \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>
</tr>
<tr>
<td>( B2 )</td>
<td>000 0000</td>
<td>111 0000</td>
<td>001 1000</td>
<td>111 0111</td>
</tr>
<tr>
<td>( B3 )</td>
<td>000 0000</td>
<td>001 1100</td>
<td>000 1110</td>
<td>001 1110</td>
</tr>
<tr>
<td>( B4 )</td>
<td>000 0000</td>
<td>001 1110</td>
<td>001 0111</td>
<td>001 1110</td>
</tr>
<tr>
<td>( \text{EXIT} )</td>
<td>000 0000</td>
<td>001 0111</td>
<td>001 0111</td>
<td></td>
</tr>
</tbody>
</table>

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
\( \text{OUT}[\text{ENTRY}] = \emptyset \)

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 \( \text{ENTRY} \)) {
        \( \text{IN}[B] = \bigcup \text{P a predecessor of } B \ \text{OUT}[P] \)
        \( \text{OUT}[B] = \text{gen}_B \cup (\text{IN}[B] - \text{kill}_B) \)
    }
}

<table>
<thead>
<tr>
<th>( B1 )</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>( B2 )</td>
<td>000 0000</td>
<td>000 0000</td>
<td>111 0000</td>
<td>000 0000</td>
<td>111 0000</td>
</tr>
<tr>
<td>( B3 )</td>
<td>000 0000</td>
<td>111 0000</td>
<td>001 1100</td>
<td>111 0111</td>
<td>001 1110</td>
</tr>
<tr>
<td>( B4 )</td>
<td>000 0000</td>
<td>001 1100</td>
<td>000 1110</td>
<td>001 1110</td>
<td>000 1110</td>
</tr>
<tr>
<td>( \text{EXIT} )</td>
<td>000 0000</td>
<td>001 0111</td>
<td>001 0111</td>
<td>001 0111</td>
<td>001 0111</td>
</tr>
</tbody>
</table>

\( d1: i = m - 1 \)
\( d2: j = n \)
\( d3: a = u1 \)
\( d4: i = i + 1 \)
\( d5: j = j - 1 \)
\( d6: a = u2 \)
\( d7: i = u3 \)
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

\[
\begin{align*}
\text{IN[EXIT]} &= \emptyset \\
\text{IN[B]} &= \text{use}_B \cup (\text{OUT[B]} - \text{def}_B) \\
\text{OUT[B]} &= \bigcup_{S \text{ a successor of } B} \text{IN[S]}
\end{align*}
\]
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 * i in block B3 will be a common subexpression if 4 * i is available at the entry point of block B3.

[p 611]
"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 $t1$, 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]
## Example 9.15

<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>${ b + c }$</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 \( \text{IN}[B] \) be the set of expressions in \( U \) that are available at the point just before the beginning of \( B \). Let \( \text{OUT}[B] \) be the same for the point following the end of \( B \). Define \( \text{e}_{\_ \text{gen}}[B] \) to be the expressions generated by \( B \) and \( \text{e}_{\_ \text{kill}}[B] \) to be the set of expressions in \( U \) killed in \( B \). Note that \( \text{IN} \), \( \text{OUT} \), \( \text{e}_{\_ \text{gen}} \), and \( \text{e}_{\_ \text{kill}} \) can all be represented by bit vectors.” [p. 612]
9.2.6 Available expressions definitions, page 612

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

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

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

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

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

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

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:

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

\[
\text{OUT}[\text{ENTRY}] = \emptyset \\
\text{for (each basic block } B \text{ other than ENTRY)} \{ \text{ OUT}[B] = U \} \\
\text{while (changes to any OUT occur) } \{ \\
\text{ for (each basic block } B \text{ other than EXIT)} \{ \\
\text{ IN}[B] = \cap \text{ a predecessor of } B \text{ OUT}[?] \\
\text{ OUT}[B] = e_{\text{gen}}_B \cap (\text{IN}[B] - e_{\text{kill}}_B) \\
\text{ } \\
\text{ } \\
\} \\
\} \\
\]

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

<table>
<thead>
<tr>
<th></th>
<th>Reaching definitions</th>
<th>Live variables</th>
<th>Available expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
<td>sets of definitions</td>
<td>sets of variables</td>
<td>sets of expressions</td>
</tr>
<tr>
<td><strong>Direction</strong></td>
<td>forward</td>
<td>backward</td>
<td>forward</td>
</tr>
<tr>
<td><strong>Transfer function</strong></td>
<td>gen$_B$ U (x - kill$_B$)</td>
<td>use$_B$ U (x - def$_B$)</td>
<td>e$_-_gen$<em>B$ \cap (x - e$</em>-_kill$_B)</td>
</tr>
<tr>
<td><strong>Boundary</strong></td>
<td>OUT[ENTRY] = ∅</td>
<td>IN[EXIT] = ∅</td>
<td>OUT[ENTRY] = ∅</td>
</tr>
<tr>
<td><strong>Meet (∧)</strong></td>
<td>U</td>
<td>U</td>
<td>∩</td>
</tr>
<tr>
<td><strong>Equations</strong></td>
<td>OUT[$B$] = f$_B$(IN[$B$])</td>
<td>IN[$B$] = f$_B$(OUT[$B$])</td>
<td>OUT[$B$] = f$_B$(IN[$B$])</td>
</tr>
<tr>
<td></td>
<td>IN[$B$] = \land_{P,pred(B)}OUT[P]</td>
<td>OUT[$B$] = \land_{S,succ(B)}IN[S]</td>
<td>IN[$B$] = \land_{P,pred(B)}OUT[P]</td>
</tr>
</tbody>
</table>

© 2020 Carl Alphonce - Reproduction of this material is prohibited without the author's consent
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)
### 9.4.4 Monotonicity (p. 635)

<table>
<thead>
<tr>
<th></th>
<th>( m(y) )</th>
<th>( m(z) )</th>
<th>( m'(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{UNDEF} )</td>
<td>( \text{UNDEF} )</td>
<td>( \text{UNDEF} )</td>
<td>( \text{UNDEF} )</td>
</tr>
<tr>
<td></td>
<td>( \text{c1} )</td>
<td>( \text{UNDEF} )</td>
<td>( \text{UNDEF} )</td>
</tr>
<tr>
<td></td>
<td>( \text{c2} )</td>
<td>( \text{c1 + c2} )</td>
<td>( \text{NAC} )</td>
</tr>
<tr>
<td>( \text{NAC} )</td>
<td>( \text{UNDEF} )</td>
<td>( \text{NAC} )</td>
<td>( \text{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

a = b+c

evaluation outside loop

t = b+c

a = t
```
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.

The 9.31: $B_3 \rightarrow B_4$ is a critical edge.
9.5.3 Lazy code motion problem

"It is desirable for programs optimized with a partial-redundancy-elimination algorithm to have the following properties:

1. All redundant computations of expressions that can be eliminated without code duplication are eliminated.

2. The optimized program does not perform any computation that is not in the original program execution.

3. Expressions are computed at the latest possible time."
9.5.4 Anticipation of expressions

- "...an expression $b+c$ is anticipated at point $p$ if all paths leading from the point $p$ eventually compute the value of the expression $b+c$ from the values of $b$ and $c$ that are available at that point." [p. 645]

- "Anticipation limits how early an expression can be inserted." [p. 645]
9.5.5 Lazy code motion algorithm

(Summary of algorithm on page 646, full algorithm on pages 653 - 654)