Lecture 13 (Data flow analysis)
Analysis of Software Artifacts
Somesh Jha
Motivation

- global information about the program

- examples
  - does variable $v$ effect a conditional?
  - can a certain code fragment be eliminated?
  - is a conditional always true?
What is it used for?

- constructing tests (we will see this later)
- program understanding
  - all the variables that effect a certain statement
  - slicing
- code optimization
  - an expression inside a loop always has the same value
  - dead code elimination
Data-flow analysis

• assume that your program can be represented as a graph

• each node in the program represents a *basic block*

• a *basic block* is a segment of code with no *conditional* statements in it

• edges in the graph represent conditional statements

• conditions corresponding to edges not shown
Real Languages

- most compilers do these kinds of analysis
- pointers make life harder
- if \texttt{x} is of type (\texttt{int *}) what does it point to?
- aliasing analysis
- OO languages can be handled quite easily
- concurrent programs are challenging
Generally conservative information

- if data-flow analysis claims that expression
- \( x+y*y \) has value 10 inside a loop
- it \textit{will} have the value
- if analysis says no, it means \textit{may be}
Reaching definitions analysis

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

- a definition $d$ reaches a point $p$ if there is a path from $d$ to $p$, such that $d$ is not killed along that path.
**Figure 1: Sample program**
Notation

• $\text{out}[B]$
  these are the definitions that are $alive$ after executing basic-block $B$

• $\text{in}[B]$
  these are the definitions that are $alive$
  coming $into$ the basic-block $B$

• $\text{kill}[B]$
  these are definitions that are $killed$ while traversing block $B$

• $\text{gen}[B]$
  these are the definitions that are $generated$
  in the basic-block $B$
Notation (Contd)

- For block $B_1$
  \[
  \text{gen}[B_1] = \{d_1, d_2, d_3\} \\
  \text{kill}[B_1] = \{d_4, d_5, d_6, d_7\}
  \]

- For block $B_2$
  \[
  \text{gen}[B_2] = \{d_4, d_5\} \\
  \text{kill}[B_2] = \{d_1, d_2, d_7\}
  \]

- For block $B_3$
  \[
  \text{gen}[B_3] = \{d_6\} \\
  \text{kill}[B_3] = \{d_3\}
  \]

- For block $B_4$
  \[
  \text{gen}[B_4] = \{d_7\} \\
  \text{kill}[B_4] = \{d_1, d_4\}
  \]
Fix-point equation

• initially, out[B] is empty for each basic block B

• update out[B] according to the following equations until you reach a fix-point:

\[
\begin{align*}
in(B) & = \cup_{P \in \text{pred}(B)} \text{out}(P) \\
\text{out}(B) & = \text{gen}(B) \cup (\text{in}(B) - \text{kill}(B))
\end{align*}
\]

• fix-point: for all basic blocks B the sets in(B) and out(B) do not change after an iteration
Pass 0

- sets are represented as bit-vectors
- each bit represents a definition
- computation for pass 0 is shown below:

<table>
<thead>
<tr>
<th>block</th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_1$</td>
<td>000 0000</td>
<td>111 0000</td>
</tr>
<tr>
<td>$B_2$</td>
<td>000 0000</td>
<td>000 1100</td>
</tr>
<tr>
<td>$B_3$</td>
<td>000 0000</td>
<td>000 0010</td>
</tr>
<tr>
<td>$B_4$</td>
<td>000 0000</td>
<td>000 0001</td>
</tr>
</tbody>
</table>

Figure 2: Pass 0
Pass 1

- using the fix-point equations we get the following table for results after pass 1

<table>
<thead>
<tr>
<th>block</th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>000 0000</td>
<td>111 0000</td>
</tr>
<tr>
<td>B2</td>
<td>111 0011</td>
<td>001 1110</td>
</tr>
<tr>
<td>B3</td>
<td>001 1110</td>
<td>000 1110</td>
</tr>
<tr>
<td>B4</td>
<td>001 1110</td>
<td>001 0111</td>
</tr>
</tbody>
</table>

Figure 3: Pass 1
Pass 2

- we reach a fix-point.

- the results are shown below:

<table>
<thead>
<tr>
<th>block</th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
<td>000 0000</td>
<td>111 0000</td>
</tr>
<tr>
<td>B2</td>
<td>111 1111</td>
<td>001 1110</td>
</tr>
<tr>
<td>B3</td>
<td>001 1110</td>
<td>000 1110</td>
</tr>
<tr>
<td>B4</td>
<td>001 1110</td>
<td>001 0111</td>
</tr>
</tbody>
</table>

Figure 4: Pass 2
Available expressions analysis

- an expression $x + y$ is available at a point $p$

- if every path from the initial node to $p$
  evaluates $x + y$

- after last such evaluation prior to reaching $p$ there are no subsequent assignments to $x$ or $y$. 
Live-variable analysis

- a variable \( x \) is called \textit{live} at a point \( p \) if \( x \) could be used along some path in the flow-graph starting at \( p \).

- if the condition given above is not true, the variable \( x \) is called \textit{dead}.