Computing Data-Flow Info.

- There are two principal methods of solving data-flow problems:
  1. Let gen, kill, in, out be AST attributes and the data-flow equations attribute evaluation rules. We’ll look at this later.
  2. Treat data-flow equations as recurrences, and iterate over the set of equations until a solution is found.
- Sets are stored as bit-vectors, with one element for each possible object.

\[
in[B] = \{d_3, d_5, d_7\} \\
\equiv \begin{array}{cccccccc}
     0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
  d_1 & d_2 & d_3 & d_4 & d_5 & d_6 & d_7 & d_8
\end{array}
\]

Slide 7–1

Reaching Definitions

Equations

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

- A definition \( d : a := b + c \) reaches a use of \( a \) at point \( p \), if the value given to \( a \) at \( d \) could be used at \( p \).
- \( \text{gen}[B] \) is the set of definitions generated within \( B \), that reach the end of \( B \).
- \( \text{kill}[B] \) is the set of definitions outside \( B \), killed by definitions within \( B \).
- \( \text{in}[B] \) is the set of definitions valid at the entrance to \( B \), \( \text{out}[B] \) is those valid at the exit of \( B \).
- The equations for in and out are valid for each basic block.

Slide 7–2

Iterative Algorithms I

1. Compute gen and kill for each block.
2. Set up the \( 2n \) in- and out-equations for the \( n \) basic blocks, and set \( \text{in}[B] = \text{out}[B] = \{\} \) for each block \( B \).
3. Repeat until no more changes:
   - For each block \( B \) eval. \( \text{in}[B] \& \text{out}[B] \).

Formal Algorithm:

\[
\text{FOR each block } B \text{ DO} \\
\text{out}[B] := \text{in}[B] := \{\} \; ; \\
\text{END;} \\
\text{WHILE any out}[B] \text{ has changed DO} \\
\text{FOR each block } B \text{ DO} \\
\quad \text{in}[B] := \bigcup_{\text{pred}P \text{ of } B} \text{out}[P] \\
\quad \text{out}[B] := \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B]) \\
\text{END;} \\
\text{END}
\]

Slide 7–3
Iterative Alg. Example I (a)

\[
\text{REPEAT} \\
\quad d_1: i := \ldots; \\
\quad \text{IF } \ldots \text{THEN} \\
\quad d_2: i := \ldots \\
\quad \text{ELSE} \\
\quad d_3: i := \ldots \\
\quad \text{ENDIF} \\
\quad d_4: k := \ldots \\
\text{UNTIL } \ldots; 
\]

- Start by setting up the \(2n\) in- and out-equations (slide (b)).
- Simplify the example by inlining gen and kill into the equations for in and out (slide (c)).
- Visit each block in turn (we use numerical order, B1, B2, B3, B4) and evaluate in and out (slides (d)–(g)).
Example II (b) – 2nd Iteration

\[ d_1: i := m - 1 \]
\[ d_2: j := n \]
\[ d_3: a := \ldots \]
\[ g = \{d_1, d_2, d_3\} \]
\[ k = \{d_4, d_5, d_6, d_7\} \]
\[ i = \{\} \]
\[ o = \cup (i \cdot k) \]
\[ \{d_1, d_2, d_3\} \]
\[ \{d_1 \ldots d_3\} \]
\[ \{d_3 \ldots d_6\} \]
\[ \{d_3 \ldots d_5\} \]
\[ = \{d_4 \ldots d_6\} \]
\[ = \{d_4 \ldots d_5\} \]
\[ = \{d_3 \ldots d_6\} \]
\[ = \{d_3 \ldots d_5\} \]

Live Variable Analysis I

- For each definition/use of a variable \( V \), Global Live Variable Analysis answers the question
  “Could the value of \( V \)
  computed/used here be used
  further on in the program?”
- If a variable \( V \) is stored in a register
  \( R5 \) and \( V \) is dead at the end of the block, then we don’t have to store \( R5 \)
  back into \( V \).
- Assignments to dead variables can be removed.

\[ \begin{align*}
R5 &: = V; \\
R5 &: = R5 + 1;
\end{align*} \]

V (stored in R5)
\[ \text{is incremented} \]

V is dead here.

No further used of \( V \) here.

Live Variable Analysis II

**in[B]** Variables live on entrance to B.

**out[B]** Variables live on exit from B.

**def[B]** Variables assigned values in B
before the variable is used:

\[ \begin{align*}
B &: = \ldots C \ldots; \\
C &: = \ldots; \\
\ldots &: = \ldots B \ldots;
\end{align*} \]

**use[B]** Variables whose values are used
before being assigned to:

\[ \begin{align*}
B &: = \ldots C \ldots; \\
C &: = \ldots; \\
\ldots &: = \ldots B \ldots;
\end{align*} \]

Live Variable Analysis III

**Data-Flow (Equations):**

\[ \text{in}[B] = \text{use}[B] \cup (\text{out}[B] - \text{def}[B]) \]

**Data-Flow (English):**

- \( V \) is live at the entrance to B if
  1. \( V \) is being used before it’s defined
     \( \text{i.e. } V \in \text{use}[B] \)
     \( \text{in} = \{\ldots C \ldots \} \)
     \( \text{use} = \{ C \} \)

\[ \begin{align*}
B &: = \ldots C \ldots; \\
C &: = \ldots;
\end{align*} \]

\( C \) is in \( \text{in}[B] \) since
its value is used
before \( C \) is defined.

2. \( V \) is not defined within the block (i.e.
\( V \notin \text{def}[B] \))
\( \text{in} = \{\ldots C \ldots \} \)
\( \text{def} = \{ B \} \)
\( \text{out} = \{ \ldots C,B \ldots \} \)
Live Variable Analysis IV

Data-Flow (Equations):

\[ \text{out}[B] = \bigcup \text{in}[S] \]

successors

\( S \) of \( B \)

Data-Flow (English):

- A variable \( V \) is live coming out of \( B \) if it is live going into any one of \( B \)'s successors.

\( B \quad \text{C := ...} \quad \text{out=} \{ \text{C} \} \)

\( \text{in=} \{ \} \quad \text{in=} \{ \text{C} \} \quad \ldots := \ldots \text{C...} \)

Slide 7–16

---

Live Variables VI – Example

\( \text{out}[B4]=\{\} \) since \text{out} is the union of all of \( B4 \)'s successor's \( \text{in} \), and \( B4 \) doesn't have any successors.

\( \text{in}[B4]=\{\} \) because both \( A \) & \( B \) are live coming in to \( B4 \), i.e. their values will be used before they are assigned new values.

\( \text{out}[B3]=\text{in}[B4]=\{A,B\} \) because the values of \( A \) and \( B \) will be used in \( B3 \)'s successor block, \( B4 \). Note that since \( C \not\in \text{out}[B3] \) \( C \)'s value is \text{dead} and the assignment \( C := 1 \) can be removed.

\( \text{out}[B1]=\{A\} \cup \{A,B\}=\{A,B\} \) since if we take the left branch (through \( B2 \)) \( A \) will be used further on, and if we take the right branch (through \( B3 \)) both \( A \) and \( B \) will have a future use.

\( \text{in}[B1]=\{B\} \) since \( B \)'s value is used but not defined in \( B \).

Slide 7–18

---

Summary I


- With \( B \) blocks & bit-vectors of length \( V \), iterative data-flow analysis is \( O(B^2 \times V) \) in the worst case.

- Data-flow problems can be classified according to the direction of flow:

  **Forward-flow problems**: Data flows from the initial block to the end block. Out-sets are computed from In-sets within basic blocks, In-sets are computed from Out-sets across basic blocks.

  **Backward-flow problems**: Data flows from the end block to the initial block. In-sets are computed from Out-sets within basic blocks, Out-sets are computed from In-sets across basic blocks.

Slide 7–19
Summary II

<table>
<thead>
<tr>
<th>Forward-Flow</th>
<th>Backward-Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>( o_B = g_B \cup (i_B - k_B) )</td>
<td>( i_B = \bigcap_{b \in S(B)} o_b )</td>
</tr>
</tbody>
</table>
| \( i_B = \bigcup_{b \in P(B)} (i_b - k_b) \) | \( o_B = \bigcup_{b \in P(B)} i_b \)

\( P(B) \) = Predecessors of \( B \), \( S(B) \) = Successors of \( B \).

\( i_B = \text{in}_B \), \( o_B = \text{out}_B \), \( g_B = \text{gen}_B \), \( k_B = \text{kill}_B \).

- Show each step of the iterative reaching definitions algorithm applied to the procedure body below:

\[
\begin{align*}
K &:= 1; I := 2; \\
\text{REPEAT} & \quad \text{IF I = 4 THEN} \\
& \quad \quad A := K + 1; \\
& \quad \text{ELSE} \\
& \quad \quad A := K + 2; \\
& \quad \quad I := I + A; \\
& \quad \text{ENDIF;} \\
& \quad \text{UNTIL I \leq 10;} \\
K &:= K + A;
\end{align*}
\]

Summary III

- We classify data-flow problems by the way they combine incoming information:

\[
\begin{array}{ll}
\text{Any-path problems:} & \quad \text{Any-Path problems:} \\
\text{All values coming in to a block are valid. Use \( \bigcup \).} & \quad \text{All-path problems:} \\
\text{Only values coming in to a block through every path are valid. Use \( \bigcap \).}
\end{array}
\]

<table>
<thead>
<tr>
<th>Forward-Flow</th>
<th>Backward-Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any Reaching Definitions</td>
<td>Live Variables</td>
</tr>
<tr>
<td>Path Uninitialized Variables</td>
<td>Du-chains</td>
</tr>
<tr>
<td>All Available Expressions</td>
<td>Very Busy Expressions</td>
</tr>
<tr>
<td>Paths Copy Propagation</td>
<td></td>
</tr>
</tbody>
</table>

Homework I

Exam Problem I (a) [07.430 '95]

- An expression \( E \) is very busy if – regardless of which path we take through the flow graph – \( E \)'s value will be used before it is killed. Example (\( A+3 \) is very busy):

\[
\begin{align*}
(1) & \quad \text{BEGIN} \\
(2) & \quad \text{IF expr THEN} \\
(3) & \quad \quad V := A + 3; \\
(4) & \quad \quad R := K + 3; \\
(5) & \quad \text{ELSE} \\
(6) & \quad \quad Z := A + 3; \\
(7) & \quad \quad K := 5; \\
(8) & \quad \quad L := K + 3; \\
(9) & \quad \text{END;} \\
(10) & \quad \text{END}
\end{align*}
\]
Exam Problem I (b) [07.430 ’95]

Data-Flow Equations:

- The data-flow equations for computing very busy expressions are:

\[ \text{in}[B] = \text{used}[B] \cup (\text{out}[B] \setminus \text{killed}[B]) \]
\[ \text{out}[B] = \bigcap_{\text{successors } S \text{ of } B} \text{in}[S] \]

Problems:

1. Give an iterative pseudo-code routine for computing \text{in} and \text{out}.
2. Is \textit{very-busy expressions} a forward-flow or a backward-flow problem?
3. Show the workings of the algorithm on the procedure body in the next slide:

---

Exam Problem I (c) [07.430 ’95]

\[
\text{BEGIN} \\
\text{X} := 5; \\
\text{Y} := 10; \\
\text{IF } e_1 \text{ THEN} \\
\text{IF } e_2 \text{ THEN} \\
\text{A} := \text{X} \times \text{Y}; \\
\text{ELSE} \\
\text{B} := 3; \\
\text{V} := \text{X} \times \text{Y}; \\
\text{X} := 1; \\
\text{END}; \\
\text{ELSE} \\
\text{Y} := 2; \\
\text{A} := \text{X} \times \text{Y}; \\
\text{END} \\
\text{END}
\]

---

Slide 7–24

Slide 7–25