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.
\[ \text{in}[B1] = \{d_3, d_5, d_7\} \]
\[
\begin{array}{cccccccc}
  & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
\text{d_1} & \text{d_2} & \text{d_3} & \text{d_4} & \text{d_5} & \text{d_6} & \text{d_7} & \text{d_8}
\end{array}
\]

Slide 18–1

Reaching Definitions
\[ \begin{align*}
\text{out}[B] &= \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B]) \\
\text{in}[B] &= \bigcup_{\text{preds}_P \text{ of } B} \text{out}[P]
\end{align*} \]
• 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 18–2

Iterative Algorithms I
1. Compute \( \text{gen} \) and \( \text{kill} \) 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. \( \text{in}[B] \& \text{out}[B] \).

\[
\begin{align*}
\text{FOR each block } B \text{ DO} \\
\text{out}[B] := \text{in}[B] := \{\} \\
\text{END;}
\end{align*}
\]
\[
\begin{align*}
\text{WHILE any } \text{out}[B] \text{ has changed DO} \\
\text{FOR each block } B \text{ DO} \\
\text{in}[B] := \bigcup_{\text{preds}_P \text{ of } B} \text{out}[P] \\
\text{out}[B] := \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B]) \\
\text{END;} \\
\text{END}
\end{align*}
\]
Iterative Alg. Example I (a)

REPEAT
   \( d_1 \): i := ...;
   IF ... THEN
      \( d_2 \): i := ...
   ELSE
      \( d_3 \): i := ...
   ENDIF;
   \( d_4 \): k := ...
UNTIL ...;

- 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

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

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

\[ d_6: a := ... \]
\[ d_7: i := ... \]

\[ g =\{d_1, d_2, d_3\} \]
\[ k =\{d_4, d_5, d_6, d_7\} \]
\[ i =\{} \]
\[ o = g \cup (i-k) \]
\[ =\{d_1, d_2, d_3\} \]
\[ i_{old} =\{d_1 \cdots d_3\} \]
\[ o_{old} =\{d_3 \cdots d_5\} \]
\[ g =\{d_4, d_5\} \]
\[ k =\{d_1, d_2, d_3\} \]
\[ i =\{B_1 \cup o[B_3] \cup o[B_4]\} \]
\[ =\{d_1 \cdots d_7\} \]
\[ o = g \cup (i-k) \]
\[ =\{d_3 \cdots d_6\} \]

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.

\[ R5 := V; \]
\[ R5 := R5 + 1; \]
\[ V \] is incremented

\[ V \] is dead here.

No further used of \( V \) here.

Live Variable Analysis II

- \( \text{in}[B] \) Variables live on entrance to \( B \).
- \( \text{out}[B] \) Variables live on exit from \( B \).
- \( \text{def}[B] \) Variables assigned values in \( B \) before the variable is used:
  - \( B := \ldots C \ldots \)
  - \( C := \ldots \)
  - \( \ldots := \ldots B \ldots \)

- \( \text{use}[B] \) Variables whose values are used before being assigned to:
  - \( B := \ldots C \ldots \)
  - \( C := \ldots \)
  - \( \ldots := \ldots B \ldots \)

Live Variable Analysis III

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

\[ \text{Data-Flow (English)}: \]

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

\[ \text{in} =\{\ldots C \ldots\} \]
\[ \text{use} =\{ C \} \]
\[ B := \ldots C \ldots; \]
\[ C := \ldots; \]
\[ \ldots := \ldots B \ldots; \]

\[ \text{def} =\{ B \} \]

\[ \text{out} =\{\ldots C, B \ldots\} \]
Live Variable Analysis IV

Data-Flow (Equations):

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

Data-Flow (English):

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

\[
\text{B} \quad \text{C := ...} \quad \text{out}=\{ \text{C} \} \\
\text{in}=\{} \quad \text{in}=\{ \text{C} \} \quad \ldots:=...C...
\]

Live Variables VI – Example

\( \text{out}[B4]=\{} \) since out is the union of all of \( B4 \)'s successor's 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 \( C \notin \text{out}[B3] \) \( C \)'s value is \textit{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 \( B \).

Summary I


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

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

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

  \textbf{Backward-flow problems}: Data flows from the end block to the initial block. \textit{In-sets} are computed from \textit{Out-sets} within basic blocks, \textit{Out-sets} are computed from \textit{In-sets} across basic blocks.
Summary II

<table>
<thead>
<tr>
<th></th>
<th>Forward-Flow</th>
<th>Backward-Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any</td>
<td>$i_B = g_B \cup (o_B - k_B)$</td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td>$o_B = \bigcup_{k \in P(B)} o_b$</td>
<td></td>
</tr>
<tr>
<td>All Paths</td>
<td>$i_B = \bigcap_{k \in P(B)} i_b$</td>
<td></td>
</tr>
<tr>
<td>Paths</td>
<td>$o_B = \bigcup_{k \in S(B)} o_b$</td>
<td></td>
</tr>
<tr>
<td>Paths</td>
<td>$i_B = \bigcap_{k \in S(B)} i_b$</td>
<td></td>
</tr>
</tbody>
</table>

* $P(B)$ = Predecessors of $B$, $S(B)$ = Successors of $B$.
* $i_B = \text{in}_B$, $o_B = \text{out}_B$, $g_B = \text{gen}_B$, $k_B = \text{kill}_B$.

- We classify data-flow problems by the way they combine incoming information:
  - Any-path problems: All values coming in to a block are valid. Use $\bigcup$.
  - All-path problems: Only values coming in to a block through every path are valid. Use $\bigcap$.

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

Summary III

Homework I

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

```plaintext
K := 1; I := 2;
REPEAT
  IF I = 4 THEN
    A := K + 1;
  ELSE
    A := K + 2;
    I := I + A;
  ENDIF;
UNTIL I <= 10;
K := K + A;
```

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):

```plaintext
(1) BEGIN
(2) IF expr THEN
(3) V := A + 3;
(4) R := K + 3;
(5) ELSE
(6) Z := A + 3;
(7) K := 5;
(8) L := K + 3;
(9) END;
(10) END
```
Exam Problem I (b) [07.430 ’95]

Data-Flow Equations:

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

\[
\begin{align*}
\text{in}[B] &= \text{used}[B] \cup (\text{out}[B] - \text{killed}[B]) \\
\text{out}[B] &= \bigcap \text{successors} \\
&\quad S \text{ of } B
\end{align*}
\]

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]

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

END

---