Lecture Outline

1. Review
   • The DAG representation of basic blocks
   • Finding local common subexpressions
   • The use of algebraic identities
   • Peephole optimizations

2. The Data-Flow Analysis Schema
   • A data-flow value at a program point represents the set of all possible programs states that can be observed for that point, for example, all definitions in the program that can reach that point.
   • Let IN[s] and OUT[s] be the set of data-flow values before and after a statement s in a program.
   • A transfer function $f_s$ relates the data-flow values before and after a statement s.
   • In a forward data-flow problem
     \[ \text{OUT}[s] = f_s(\text{IN}[s]) \]
     In a backward data-flow problem
     \[ \text{IN}[s] = f_s(\text{OUT}[s]) \]
   • A transfer function can be extended to a basic block by composing the transfer functions for all the statements in the block. Thus in a forward data-flow problem such as reaching definitions for a block $B$,
     \[ \text{OUT}[B] = f_B(\text{IN}[B]) \]
Given a flow graph, in a forward data-flow problem the IN set of a basic block $B$ is computed from the OUT sets of $B$'s predecessors:

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

In a backward data-flow problem such as live variable analysis:

$$\text{IN}[B] = f_B(\text{OUT}[B])$$

$$\text{OUT}[B] = \bigcup_{S \text{ a successor of } B} \text{IN}[S]$$

The data-flow problem for a flow graph is to compute the values of $\text{IN}[B]$ and $\text{OUT}[B]$ for all blocks $B$ in the flow graph.

3. Reaching Definitions

A definition $d$ reaches a program point $p$ if there is a path from the point immediately following $d$ to $p$ such that $d$ is not killed along that path.

Flow graph with gen and kill sets for each basic block:

- $\text{ENTRY}$
- $d_1: i = m - 1$
- $d_2: j = n$
- $d_3: a = u_1$
- $\text{B}_1$
  - $\text{gen}_{B_1} = \{ d_1, d_2, d_3 \}$
  - $\text{kill}_{B_1} = \{ d_4, d_5, d_6, d_7 \}$
- $d_4: i = i + 1$
- $d_5: j = j - 1$
- $\text{B}_2$
  - $\text{gen}_{B_2} = \{ d_4, d_5 \}$
  - $\text{kill}_{B_2} = \{ d_1, d_2, d_7 \}$
- $d_6: a = u_2$
- $\text{B}_3$
  - $\text{gen}_{B_3} = \{ d_6 \}$
  - $\text{kill}_{B_3} = \{ d_3 \}$
- $\text{B}_4$
  - $\text{gen}_{B_4} = \{ d_7 \}$
  - $\text{kill}_{B_4} = \{ d_1, d_4 \}$
- $\text{EXIT}$
• \( \text{gen}_B \) contains all definitions in block \( B \) that are visible immediately after block \( B \).

• \( \text{kill}_B \) is the union of all definitions killed by the statements in block \( B \).

4. Control-Flow Equations for Reaching Definitions

• The reaching definitions problem is defined by the following control-flow equations:

\[
\text{OUT}[\text{ENTRY}] = \text{empty\_set}
\]

For all blocks \( B \) other than \( \text{ENTRY} \):

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

5. Iterative Algorithm for Reaching Definitions

• Given a flow graph for which the \( \text{gen} \) and \( \text{kill} \) sets have been computed for each block, we can compute the set of definitions reaching the entry and exit of each block \( B \) using the following iterative algorithm:

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

• Example: Let us represent a set of definitions in the flow graph above by a bit vector. Thus \( 1110000 \) represents the set \{ \( d_1, d_2, d_3 \) \}. The following table represents the values taken on by the \( \text{IN} \) and \( \text{OUT} \) sets after each iteration of the while-loop of this algorithm. The superscript denotes the iteration. The initial values of \( \text{OUT} \), computed by the second statement of the algorithm, are indicated by the superscript 0.
IN and OUT sets for the basic blocks of the flowgraph in Section 2.

6. Live-Variable Analysis

- In live-variable analysis we want to determine for each variable $x$ and each program point $p$ whether the value $x$ at $p$ could be used along some path in the flow graph starting at $p$. If so, we say $x$ is live at $p$; if not, $x$ is dead at $p$. Live-variable analysis is crucial for register allocation.

- Live-variable analysis is an example of a backwards data-flow problem.

- Define $\text{def}_B$ as the set of variables defined in $B$ prior to any use of that variable in $B$. In the flow graph above $\text{def}_{B2} = \{ i, j \}$.

Define $\text{use}_B$ as the set of variables whose values may be used in $B$ prior to any definition of the variable. In the flow graph above $\text{use}_{B2} = \{ i, j \}$.

- Data-flow equations for live-variable analysis:

$$\text{IN[EXIT]} = \text{empty_set}$$

For all blocks $B$ other than EXIT:

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

$$\text{OUT[ } B \text{ ]} = \bigcup_{S \text{ a predecessor of } B} \text{IN[ } S \text{ ]}$$
Given a flow graph for which the \textit{def} and \textit{use} sets have been computed for each block, we can compute the set of variables live on entry and exit of each block \( B \) using the following iterative algorithm:

\[
\begin{align*}
\text{IN(EXIT)} &= \text{empty_set}; \\
\text{for (each basic block \( B \) other than EXIT)} & \quad \text{IN}(B) = \text{empty_set}; \\
\text{while (changes to any IN occur)} & \quad \text{for (each basic block \( B \) other than EXIT)} \{ \\
&\quad \quad \text{OUT}(B) = \bigcup \text{a predecessor of } B \text{ IN}(S); \\
&\quad \quad \text{IN}(B) = \text{use}_B \bigcup (\text{OUT}(B) - \text{def}_B); \\
&\quad \}
\end{align*}
\]

Unlike reaching definitions, the information flow for liveness travels backward in the data-flow graph, opposite to the direction of control flow.

However, as for reaching definitions, live-variable analysis uses union as the “meet” operator. We are interested in whether any path with the desired properties exists, not whether something is true along all paths.

Also as for reaching definitions, the solution to the data-flow equation is not necessarily unique. We want the solution with the smallest sets of live variables.

7. Reading