Lecture 2
Introduction to Data Flow Analysis

I  Introduction
II  Example: Reaching definition analysis
III Example: Liveness Analysis
IV  A General Framework
    (Theory in next lecture)

Reading: Chapter 9.2

I. Compiler Organization

Program

Front end

Abstract Syntax Tree

High-level IR

Machine-Independent Intermediate Representations

High-level optimization
Parallelization
Loop transformations

Low-level IR

Low-level optimization
Redundancy elimination

Code generation

Machine code

Register allocation
Instruction scheduling
Flow Graph

- Basic block = a maximal sequence of consecutive instructions s.t.
  - flow of control only enters at the beginning
  - flow of control can only leave at the end
    (no halting or branching except perhaps at end of block)

- Flow Graphs
  - Nodes: basic blocks
  - Edges
    - $B_i \rightarrow B_j$; iff $B_j$ can follow $B_i$ immediately in some execution

What is Data Flow Analysis?

- Data flow analysis:
  - Flow-sensitive: sensitive to the control flow in a function
  - intraprocedural analysis

- Examples of optimizations:
  - Constant propagation
  - Common subexpression elimination
  - Dead code elimination

Value of $x$?
Which “definition” defines $x$?
Is the definition still meaningful (live)?
**Static program vs. dynamic execution**

- **Statically**: Finite program
- **Dynamically**: Can have infinitely many possible execution paths
- **Data flow analysis abstraction**:
  - For each static point in the program:
    combines information of all the dynamic instances of the same program point.
- **Example of a data flow question**:
  - Which definition defines the value used in statement “b = a”?

**II. Reaching Definitions**

- Every assignment is a definition
- A **definition** \( d \) **reaches** a point \( p \)
  if there exists a path from the point immediately following \( d \) to \( p \)
  such that \( d \) is not killed (overwritten) along that path.
- Problem statement
  - For each point in the program, determine
    if each definition in the program reaches the point
  - A bit vector per program point, vector-length = #defs
Data Flow Analysis Schema

- Build a flow graph (nodes = basic blocks, edges = control flow)
- Set up a set of equations between in\[b\] and out\[b\] for all basic blocks b
  - Effect of code in basic block:
    Transfer function \( f_b \) relates in\[b\] and out\[b\], for same b
  - Effect of flow of control:
    relates out\[b_1\], in\[b_2\] if b\_1 and b\_2 are adjacent
- Find a solution to the equations

Effects of a Statement

\[ \text{in[B0]} \]

\[
\begin{align*}
\text{d}_0: y &= 3 \\
\text{d}_1: x &= 10 \\
\text{d}_2: y &= 11
\end{align*}
\]

\[ \text{out[B0]} \]

\[
\begin{align*}
f_{d_0} \\
f_{d_1} \\
f_{d_2}
\end{align*}
\]

- \( f_s \): A transfer function of a statement
  abstracts the execution with respect to the problem of interest
- For a statement \( s \) (d: \( x = y + z \))
  \( \text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s] \cdot \text{Kill}[s]) \)
  - \( \text{Gen}[s] \): definitions generated: \( \text{Gen}[s] = \{d\} \)
  - Propagated definitions: \( \text{in}[s] \cdot \text{Kill}[s] \),
    where \( \text{Kill}[s] \) = set of all other defs to \( x \) in the rest of program
Effects of a Basic Block

- Transfer function of a statement $s$:
  \[ \text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s]-\text{Kill}[s]) \]

- Transfer function of a basic block $B$:
  Composition of transfer functions of statements in $B$
  \[ \text{out}[B] = f_B(\text{in}[B]) = f_{d_2} \circ f_{d_1} \circ f_{d_0}(\text{in}[B]) \]
  \[ = \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] \cup (\text{in}[B]-\text{Kill}[d_0])-\text{Kill}[d_1])) -\text{Kill}[d_2] \]
  \[ = \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] - \text{Kill}[d_1]) - \text{Kill}[d_2]) \cup \text{in}[B] - (\text{Kill}[d_0] \cup \text{Kill}[d_1] \cup \text{Kill}[d_2]) \]
  \[ = \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B]) \]
  - $\text{Gen}[B]$: locally exposed definitions (available at end of bb)
  - $\text{Kill}[B]$: set of definitions killed by $B$

Effects of the Edges (acyclic)

- Join node: a node with multiple predecessors
- $\text{meet}$ operator ($\land$): $\cup$
  \[ \text{in}[b] = \text{out}[p_1] \cup \text{out}[p_2] \cup ... \cup \text{out}[p_n], \text{where} \]
  \[ p_1, ..., p_n \text{ are predecessors of } b \]
Cyclic Graphs

- Equations still hold
  - \( \text{out}[b] = f_b(\text{in}[b]) \)
  - \( \text{in}[b] = \text{out}[p_1] \cup \text{out}[p_2] \cup \ldots \cup \text{out}[p_n], \ p_1, \ldots, p_n \text{ pred.} \)
- Find: fixed point solution

Reaching Definitions: Iterative Algorithm

input: control flow graph \( \text{CFG} = (N, E, \text{Entry}, \text{Exit}) \)

// Boundary condition
\( \text{OUT}[\text{Entry}] = \emptyset \)

// Initialization for iterative algorithm
For each basic block \( B \) other than \( \text{Entry} \)
\( \text{OUT}[B] = \emptyset \)

// iterate
While (changes to any OUT occur) {
  For each basic block \( B \) other than \( \text{Entry} \) {
    \( \text{in}[B] = \cup (\text{out}[p]), \text{ for all predecessors } p \) of \( B \)
    \( \text{out}[B] = f_B(\text{in}[B]) // \text{out}[B]=\text{gen}[B] \cup (\text{in}[B]-\text{kill}[B]) \)
  }
}
Summary of Reaching Definitions

<table>
<thead>
<tr>
<th>Domain</th>
<th>Reaching Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer function $f_b(x)$</td>
<td>forward: out$[b] = f_b(in[b])$</td>
</tr>
<tr>
<td></td>
<td>$f_b(x) = Gen_b \cup (x - Kill_b)$</td>
</tr>
<tr>
<td>Meet Operation</td>
<td>in$[b] = \cup out[predecessors]$</td>
</tr>
<tr>
<td>Boundary Condition</td>
<td>out[entry] = $\emptyset$</td>
</tr>
<tr>
<td>Initial interior points</td>
<td>out$[b] = \emptyset$</td>
</tr>
</tbody>
</table>

III. Live Variable Analysis

- **Definition**
  - A variable $v$ is **live** at point $p$ if the value of $v$ is used along some path in the flow graph starting at $p$.
  - Otherwise, the variable is **dead**.

- **Problem statement**
  - For each basic block $b$,
    - determine if each variable is live at the start/end point of $b$
  - Size of bit vector: one bit for each **variable**
Effects of a Basic Block (Transfer Function)

• Observation: Trace uses back to the definitions

\[ \text{def} \]
\[
\begin{array}{c}
\text{def} \\
\text{use}
\end{array}
\]

\[ \text{example:} \]
\[
\begin{array}{c}
m = n + q \\
p = m
\end{array}
\]

\[ \text{in}[b] = f_b(out[b]) \]

• Direction: backward: \( \text{in}[b] = f_b(out[b]) \)

• Transfer function for statement \( s: x = y + z \)
  • generate live variables: \( \text{Use}[s] = \{y, z\} \)
  • propagate live variables: \( \text{out}[s] - \text{Def}[s], \text{Def}[s] = x \)
  • \( \text{in}[s] = \text{Use}[s] \cup (\text{out}(s) - \text{Def}[s]) \)

• Transfer function for basic block \( b \):
  • \( \text{in}[b] = \text{Use}[b] \cup (\text{out}(b) - \text{Def}[b]) \)
  • \( \text{Use}[b] \), set of locally exposed uses in \( b \),
    uses not covered by definitions in \( b \)
  • \( \text{Def}[b] = \) set of variables defined in \( b \).

Across Basic Blocks

• Meet operator (\( \wedge \)):
  • \( \text{out}[b] = \text{in}[s_1] \cup \text{in}[s_2] \cup \ldots \cup \text{in}[s_n], s_1, \ldots, s_n \) are successors of \( b \)

• Boundary condition:
Example

```
\[
\begin{align*}
\text{entry} & \quad \text{in[1]} \quad \text{out[1]} \\
\text{if } g & \quad \text{n = p} \\
\text{in[2]} & \quad \text{out[2]} \\
\text{r = n+r} & \quad \text{in[3]} \quad \text{out[3]} \\
\text{exit} & \quad \text{in[exit]}
\end{align*}
\]
```

Liveness: Iterative Algorithm

```
// Boundary condition
IN[Exit] = ∅

// Initialization for iterative algorithm
For each basic block B other than Exit
IN[B] = ∅

// iterate
While (changes to any IN occur) {
    For each basic block B other than Exit {
        out[B] = \cup (in[s]), for all successors of B
        in[B] = f_B(out[B]) \quad \text{// in[B]=Use[B]∪(out[B]-Def[B])}
    }
}
```
IV. Framework

<table>
<thead>
<tr>
<th>Domain</th>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sets of definitions</td>
<td>Sets of variables</td>
</tr>
<tr>
<td>Direction</td>
<td>forward:</td>
<td>backward:</td>
</tr>
<tr>
<td></td>
<td>$\text{out}[b] = f_b(\text{in}[b])$</td>
<td>$\text{in}[b] = f_b(\text{out}[b])$</td>
</tr>
<tr>
<td></td>
<td>$\text{in}[b] = \land \text{out}[\text{pred}(b)]$</td>
<td>$\text{out}[b] = \land \text{in}[\text{succ}(b)]$</td>
</tr>
<tr>
<td>Transfer function</td>
<td>$f_b(x) = \text{Gen}_b \cup (x \cdot \text{-Kill}_b)$</td>
<td>$f_b(x) = \text{Use}_b \cup (x \cdot \text{-Def}_b)$</td>
</tr>
<tr>
<td>Meet Operator ($\land$)</td>
<td>$\cup$</td>
<td>$\cup$</td>
</tr>
<tr>
<td>Boundary Condition</td>
<td>$\text{out}[\text{entry}] = \emptyset$</td>
<td>$\text{in}[\text{exit}] = \emptyset$</td>
</tr>
<tr>
<td>Initial Interior points</td>
<td>$\text{out}[b] = \emptyset$</td>
<td>$\text{in}[b] = \emptyset$</td>
</tr>
</tbody>
</table>

Thought Problem 1. “Must-Reach” Definitions

- A definition $D (a = b+c)$ must reach point $P$ iff
  - $D$ appears at least once along on all paths leading to $P$
  - $a$ is not redefined along any path after last appearance of $D$ and before $P$
- How do we formulate the data flow algorithm for this problem?
Problem 2: A legal solution to (May) Reaching Def?

- Will the worklist algorithm generate this answer?

Problem 3. What are the algorithm properties?

- Correctness

- Precision: how good is the answer?

- Convergence: will the analysis terminate?

- Speed: how long does it take?