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
Overview of Data Flow Lectures 2-5

• **High-level programming languages generate a lot of redundancy**
• **Many useful optimizations independently developed originally**
  – Constant propagation
  – Common subexpressions
  – Loop invariant code motion
  – Dead code elimination
• **A common framework: Dataflow (recurrent equations, fixed-points)**
  – Theory: prove properties on the framework
  – Software engineering: implement / debug / optimize framework once
• **Plan**
  – L2: Basic examples to build intuition about dataflow
  – L3: Theory
  – L4: Optimization examples
  – L5: Partial redundancy elimination (PRE)
    Subsumes multiple optimizations by setting up 4 DataFlow problems
Practice Today

- Many compilers use SSA (static single assignment) – an abstraction on top of dataflow
- Idea to be covered by the homework
- Useful for many optimizations, but cannot naturally support partial redundancy elimination (PRE)
I. Compiler Organization

- Program
- Front end
- Abstract Syntax Tree
- High-level IR (High-level optimization, Parallelization, Loop transformations)
- Machine-Independent Intermediate Representations
- 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 execution
What is Data Flow Analysis?

- **Data flow analysis:**
  - Flow-sensitive: sensitive to the control flow in a function
  - Intraprocedural analysis; only on pseudo variables (no aliases)

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

Examples of questions:

- 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

**Example of a data flow question:**
- Which definition defines the value used in statement “b = a”?

**Data flow analysis abstraction:**
- For each point in the program:
  - combines information of all the instances of the same program point.
- The definitions that can reach point o are
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 =

\[
\begin{align*}
B1 & \quad \text{d0: } y = 3 \\
& \quad \text{d1: } x = 10 \\
& \quad \text{d2: } y = 11 \\
& \quad \text{if } e \\
B2 & \quad \text{d3: } x = 1 \\
& \quad \text{d4: } y = 2 \\
B3 & \quad \text{d5: } z = x \\
& \quad \text{d6: } x = 4
\end{align*}
\]
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

\[\text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s] - \text{Kill}[s])\]

- \text{Gen}[s]: definitions generated: Gen[s] = \{d\}
- \text{Propagated} definitions: in[s] - Kill[s],
  where \text{Kill}[s]=\text{set of all other defs to x in the rest of program}

Ignoring control flow

\[\begin{align*}
d0: & \quad y = 3 \\
d1: & \quad x = 10 \\
d2: & \quad y = 11 \\
& \quad \text{if e} \\
d3: & \quad x = 1 \\
d4: & \quad y = 2 \\
d5: & \quad z = x \\
d6: & \quad x = 4
\end{align*}\]
Effects of a Basic Block

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

- 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 \(B\)
  - \(\text{out}[B] = f_B(\text{in}[B])\)
    \[= f_{d1} f_{d0}(\text{in}[B])\]
    \[= \text{Gen}[d_1] \cup (\text{Gen}[d_0] \cup (\text{in}[B] - \text{Kill}[d_0])) - \text{Kill}[d_1])\]
    \[= (\text{Gen}[d_1] \cup (\text{Gen}[d_0] - \text{Kill}[d_1])) \cup \text{in}[B] - (\text{Kill}[d_0] \cup \text{Kill}[d_1])\]
    \[= \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B]).\]
  - \(\text{Gen}[B] : (\text{Gen}[d_1] \cup (\text{Gen}[d_0] - \text{Kill}[d_1]))\)
    locally exposed definitions (available at end of \(bb\))
  - \(\text{Kill}[B] : \text{Kill}[d_0] \cup \text{Kill}[d_1]\) : set of definitions killed by \(B\)
Effects of the Edges (acyclic)

- **Join node**: a node with multiple predecessors
- **meet operator** \((\land)\): \(U\)
  \[
in[b] = out[p_1] \lor out[p_2] \lor ... \lor out[p_n],\]
  where \(p_1, ..., p_n\) are all predecessors of \(b\)
Cyclic Graphs

- Equations still hold
  - $\text{out}[b] = f_b(\text{in}[b])$
  - $\text{in}[b] = \text{out}[p_1] \cup \text{out}[p_2] \cup \ldots \cup \text{out}[p_n]$, $p_1, \ldots, p_n$ 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 Entry
\text{out}[B] = \emptyset

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

<table>
<thead>
<tr>
<th></th>
<th>Reaching Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
<td>Sets of definitions</td>
</tr>
<tr>
<td><strong>Transfer function</strong> $f_b(x)$</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></td>
<td>$Gen_b$: definitions in $b$</td>
</tr>
<tr>
<td></td>
<td>$Kill_b$: killed defs</td>
</tr>
<tr>
<td><strong>Meet Operation</strong></td>
<td>$in[b] = \cup out[predecessors]$</td>
</tr>
<tr>
<td><strong>Boundary Condition</strong></td>
<td>$out[entry] = \emptyset$</td>
</tr>
<tr>
<td><strong>Initial interior points</strong></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 $p$ (that is, $v$ is not redefined along the path).
  – Otherwise, the variable is **dead**.

• Problem statement
  – For each basic block
    • determine if each variable is live in each basic block
  – Size of bit vector: one bit for each variable
Effects of a Basic Block (Transfer Function)

• **Observation:** Trace uses back to the definitions

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

• **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

\{p, q, r, g\} → \{n, q, r\} → \{n, r\} → \} → \}
Liveness: Iterative Algorithm

input: control flow graph CFG = (N, E, Entry, Exit)

// 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] = ∪ (in[s]), for all successors s of B
   }
}
### IV. Framework

<table>
<thead>
<tr>
<th>Domain</th>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>forward:</td>
<td>backward:</td>
</tr>
<tr>
<td></td>
<td>out[b] = ( f_b(in[b]) )</td>
<td>in[b] = ( f_b(out[b]) )</td>
</tr>
<tr>
<td></td>
<td>in[b] = ( \sqcap out[pred(b)] )</td>
<td>out[b] = ( \sqcap in[succ(b)] )</td>
</tr>
<tr>
<td>Transfer function</td>
<td>( f_b(x) = Gen_b \cup (x - Kill_b) )</td>
<td>( f_b(x) = Use_b \cup (x - Def_b) )</td>
</tr>
<tr>
<td>Meet Operation (( \sqcap ))</td>
<td>( \sqcup )</td>
<td>( \sqcup )</td>
</tr>
<tr>
<td>Boundary Condition</td>
<td>out[entry] = ( \emptyset )</td>
<td>in[exit] = ( \emptyset )</td>
</tr>
<tr>
<td>Initial interior points</td>
<td>out[b] = ( \emptyset )</td>
<td>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?