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 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
- Formulate individual optimizations in the same DataFlow framework:
  equations with respect to nodes in a graph
  - Theory: prove properties on the framework
  - Software engineering:
    implement / debug / optimize framework once
- 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 PRE
- Plan: L2: basic examples; L3: theory; L4: full examples; L5: PRE
I. Compiler Organization

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

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

---

Static Program vs. Dynamic Execution

- **Statically**: Finite program
- **Dynamically**: Can have infinitely many possible execution paths

- **Data flow analysis abstraction**:  
  - For each point in the program: combines information of all the instances of the same program point.

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

---
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

\[
\text{in}[B0] = \begin{cases} 
  d0: & x = 10 \\
  d1: & y = 3 \\
  d2: & y = 11
\end{cases} \]

\[
\text{out}[B0] = \begin{cases} 
  f_{d0} \\
  f_{d1} \\
  f_{d2}
\end{cases}
\]

- \( f_s \): 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: \( \text{Gen}[s] = \{d\} \)
  - \text{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

\[
\text{in}[B0] = \begin{cases} 
  d0: & x = 10 \\
  d1: & y = 3
\end{cases} \]

\[
\text{out}[B0] = \begin{cases} 
  f_{d0} \\
  f_{d1}
\end{cases} \]

\[ f_B = f_{d1}\cdot f_{d0} \]

- Transfer function of a statement \( s \):
  - \( \text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s]\cdot \text{Kill}[s]) \)
- Transfer function of a basic block \( B \):
  - Composition of transfer functions of statements in \( B \)
  - \( \text{out}[B] = f_B(\text{in}[B]) \)

\[ = \text{Gen}[d_i] \cup (\text{Gen}[d_i] \cup (\text{in}[B]\cdot \text{Kill}[d_i])) - \text{Kill}[d_i]) \]
\[ = (\text{Gen}[d_i] \cup (\text{Gen}[d_i] - \text{Kill}[d_i])) \cup \text{in}[B] - (\text{Kill}[d_i] \cup \text{Kill}[d_i]) \]
\[ = \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B]) \]

- \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
- **meet operator** ($\cap$): $U$
  
  $in[b] = out[p_1] \cup out[p_2] \cup ... \cup out[p_n]$, where $p_1, ..., p_n$ are all predecessors of $b$

Cyclic Graphs

- Equations still hold
  - $out[b] = f_b(in[b])$
  - $in[b] = out[p_1] \cup out[p_2] \cup ... \cup out[p_n]$, $p_1, ..., p_n$ pred.
- Find: fixed point solution
Reaching Definitions: Iterative Algorithm

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

// Boundary condition
out[Entry] = Ø

// Initialization for iterative algorithm
For each basic block B other than Entry
out[B] = Ø

// iterate
While (Changes to any out[] occur) {
For each basic block B other than Entry {
in[B] = \( \bigcup \) (out[p]), for all predecessors p of B
out[B] = \( f_B \) (in[B]) // 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>Domain</td>
<td>Sets of definitions</td>
</tr>
<tr>
<td>Transfer function</td>
<td>( f_b(x) ) ( f_b(x) = \text{Gen}_b \cup (x-\text{Kill}_b) ) ( \text{Gen}_b ): definitions in b ( \text{Kill}_b ): killed defs</td>
</tr>
<tr>
<td>Meet Operation</td>
<td>( \text{in}[b] = \text{out}[\text{predecessors}] )</td>
</tr>
<tr>
<td>Boundary Condition</td>
<td>out[Entry] = Ø</td>
</tr>
<tr>
<td>Initial interior points</td>
<td>out[B] = Ø</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
    - 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

  - directed graph

  - def

  - use

  - example:

  - $m = n + q$

  - $p = m$

  - $b = f_b$ (out[b])

  - $in[b] = f_b(out[b])$

- **Direction:** backward: $in[b] = f_b(out[b])$

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

- **Transfer function** for basic block $b$:
  - $in[b] = Use[b] \cup (out(b)-Def[b])$
  - $Use[b]$: set of locally exposed uses in $b$, uses not covered by definitions in $b$
  - $Def[b]$: set of variables defined in $b$. 
Across Basic Blocks

- **Meet operator ($\land$):**
  - $\text{out}[b] = \text{in}[s_1] \cup \text{in}[s_2] \cup ... \cup \text{in}[s_n]$, $s_1, ..., s_n$ are successors of $b$

- **Boundary condition:**

---

**Example**

```
entry
  n = p
  if g

r = n + r
  m = n + q
  p = m

out[1]

out[2]

out[3]
```

```
in[3]

in[2]
in[1]
```

```
exit
```
### Liveness: Iterative Algorithm

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

// **Boundary condition**
\[ \text{in}[\text{Exit}] = \emptyset \]

// **Initialization for iterative algorithm**
For each basic block $B$ other than $\text{Exit}$
\[ \text{in}[B] = \emptyset \]

// **Iterate**
While (Changes to any in[] occur) {
    For each basic block $B$ other than $\text{Exit}$ {
        \[ \text{out}[B] = \cup (\text{in}[s]), \text{for all successors } s \text{ of } B \]
        \[ \text{in}[B] = \mathcal{f}_B(\text{out}[B]) \]
    }
}

IV. Framework

<table>
<thead>
<tr>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
<td>Sets of definitions</td>
</tr>
<tr>
<td><strong>Direction</strong></td>
<td>forward: $\text{out}[b] = \mathcal{f}_b(\text{in}[b])$</td>
</tr>
<tr>
<td></td>
<td>$\text{in}[b] = \cup \text{out}[^{\text{pred}(b)}]$</td>
</tr>
<tr>
<td><strong>Transfer function</strong></td>
<td>$\mathcal{f}_b(x) = \text{Gen}_b \cup (x - \text{Kill}_b)$</td>
</tr>
<tr>
<td><strong>Meet Operation ($\land$)</strong></td>
<td>$\cup$</td>
</tr>
<tr>
<td><strong>Boundary Condition</strong></td>
<td>$\text{out}[\text{entry}] = \emptyset$</td>
</tr>
<tr>
<td><strong>Initial interior points</strong></td>
<td>$\text{out}[b] = \emptyset$</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 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?