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 PRE

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
              - Register allocation
              - Instruction scheduling
            - Machine code
**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

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 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[b1], in[b2] if b1 and b2 are adjacent
- Find a solution to the equations

**Effects of a Statement**

\[
\begin{align*}
\text{in}[B0] & \quad \text{out}[B0] \\
\text{d0}: y &= 3 & f_{d0} \\
\text{d1}: x &= 10 & f_{d1} \\
\text{d2}: y &= 11 & f_{d2}
\end{align*}
\]

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

\[
\begin{align*}
\text{in}[B_0] \quad & \quad \text{d0: } y = 3 \quad f_{d0} \\
\downarrow \quad & \quad \downarrow \quad \downarrow \\
\text{d1: } x = 10 \quad f_{d1} \quad f_B = f_{d1} \cdot f_{d0} \\
\text{out}[B_0] 
\end{align*}
\]

- 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 \( B \)
  - \( \text{out}[B] = f_B(\text{in}[B]) \)
    \( = f_{d1} \cdot f_{d0}(\text{in}[B]) \)
    \( = \text{Gen}[d_1] \cup (\text{Gen}[d_2] \cup (\text{in}[B] - \text{Kill}[d_1])) \cdot \text{Kill}[d_1] \)
    \( = (\text{Gen}[d_1] \cup (\text{Gen}[d_2] - \text{Kill}[d_1])) \cup \text{in}[B] - (\text{Kill}[d_1] \cup \text{Kill}[d_2]) \)
    \( = \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B]) \)

\( \text{Gen}[B] \): 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 (\( \land \)): \( \cup \)
  \( \text{in}[b] = \text{out}[p_1] \cup \text{out}[p_2] \cup \ldots \cup \text{out}[p_n] \), where
  \( p_1, \ldots, p_n \) are all predecessors of \( b \)
**Cyclic Graphs**

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

**Reaching Definitions: Iterative Algorithm**

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

// Boundary condition
out[Entry] = \( \emptyset \)

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

// iterate
While (Changes to any out[\( ] \) occur) {
  For each basic block \( B \) other than \( \text{Entry} \) {
    in[\( B \)] = \( \cup (\text{out}[p]) \), for all predecessors \( p \) of \( B \)
    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: $\text{out}[b] = f_b(\text{in}[b])$</td>
</tr>
<tr>
<td></td>
<td>$f_b(x) = \text{Gen}_b \cup (x \cdot \text{Kill}_b)$</td>
</tr>
<tr>
<td></td>
<td>$\text{Gen}_b$: definitions in $b$</td>
</tr>
<tr>
<td></td>
<td>$\text{Kill}_b$: killed defs</td>
</tr>
<tr>
<td>Meet Operation</td>
<td>$\text{in}[b] = \cup \text{out}[\text{predecessors}]$</td>
</tr>
<tr>
<td>Boundary Condition</td>
<td>$\text{out}[\text{entry}] = \emptyset$</td>
</tr>
<tr>
<td>Initial interior points</td>
<td>$\text{out}[b] = \emptyset$</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
    - 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
  - \( \text{def} \) → \( \text{use} \)

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

- **Boundary condition:**
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 $\text{in}[\cdot]$ occur) {
  For each basic block $B$ other than $\text{Exit}$ {
    $\text{out}[B] = \cup (\text{in}[s])$, for all successors $s$ of $B$
    $\text{in}[B] = f_s(\text{out}[B])$ // $\text{in}[B]=\text{Use}[B]\cup(\text{out}[B]-\text{Def}[B])$
  }
}
IV. Framework

<table>
<thead>
<tr>
<th>Domain</th>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direction</td>
<td>out[b] = f_b(in[b])</td>
<td>in[b] = \wedge out[pred(b)]</td>
</tr>
<tr>
<td>Transfer function</td>
<td>f_b(x) = Gen_b \cup (x - \text{Kill}_b)</td>
<td>f_b(x) = Use_b \cup (x - \text{Def}_b)</td>
</tr>
<tr>
<td>Meet Operation (\wedge)</td>
<td>\wedge</td>
<td>\wedge</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?