Lecture 2

Introduction to Data Flow Analysis

I  Example: Reaching definition analysis

II  Example: Liveness Analysis

III  A General Framework
     (Theory in next lecture)

Reading: Chapter 9.2
Data Flow Analysis

• **Data flow analysis:**
  - properties of data taken into consideration the control flow in a function (also known as flow-sensitive analysis)
  - 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)?
I. 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”?

```plaintext
B1
   a = 10

B2
   if input()
      exit

B3
   b = a
   a = 11
```
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 Basic Block

- A statement s (d: x = y + z)
  
  \( \text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s]-\text{Kill}[s]) \)

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

- \( \text{out}[B] = f_d_2 f_d_1 f_d_0(\text{in}[B]) \)

  \[
  \text{out}[B] = \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] \cup (\text{in}[B]-\text{Kill}[d_0]))) \cup \text{Kill}[d_1]) - \text{Kill}[d_2] \\
  = \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] - \text{Kill}[d_1]) - \text{Kill}[d_2]) \cup \text{in}[B] - (\text{Kill}[d_0] \cup \text{Kill}[d_1] \cup \text{Kill}[d_2]) \\
  = \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B])
  \]

- **Gen[B]:** locally exposed definitions (available at end of bb)
- **Kill[B]:** set of definitions killed by B
Effects of the Edges (acyclic)

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

- Equations still hold
  - \( \text{out}[b] = f_b(\text{in}[b]) \)
  - \( \text{in}[b] = \text{out}[p_1] \cup \text{out}[p_2] \cup ... \cup \text{out}[p_n], p_1, ..., p_n \text{ pred.} \)
- 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 $\text{OUT}$ occur) {
    For each basic block $B$ other than $\text{Entry}$ {
        $\text{in}[B] = \cup (\text{out}[p])$, 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>Reaching Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
</tr>
<tr>
<td><strong>Transfer function</strong></td>
</tr>
<tr>
<td>( f_b(x) )</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>( \text{Gen}_b ): definitions in ( b )</td>
</tr>
<tr>
<td>( \text{Kill}_b ): killed defs</td>
</tr>
<tr>
<td><strong>Meet Operation</strong></td>
</tr>
<tr>
<td><strong>Boundary Condition</strong></td>
</tr>
<tr>
<td><strong>Initial interior points</strong></td>
</tr>
</tbody>
</table>
II. 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
  - \( \text{def} \) def
  - \( \text{def} \)
  - \( \text{use} \)

- **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 \( b \)
  - \( \text{Def}[b]= \text{set of variables defined in } b.b. \)
Across Basic Blocks

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

- **Boundary condition:**
Example

out[entry] → entry

in[1] → out[1]

n = p
if g


r = n+r


m = n+q
p = m

out[entry] → in[exit]

exit

in[exit]
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 of B
      in[B] = \text{f}_B(out[B])  // in[B]=\text{Use}[B]∪(out[B]-\text{Def}[B])
   }
}
## III. 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(\text{in}[b]) )</td>
<td>in[b] = ( f_b(\text{out}[b]) )</td>
</tr>
<tr>
<td></td>
<td>in[b] = ( \land \text{out}[\text{pred}(b)] )</td>
<td>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>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>
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?
What are the algorithm properties?

- Correctness

- Precision: how good is the answer?

- Convergence: will the analysis terminate?

- Speed: how long does it take?