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
    - Machine-Independent Intermediate Representations
      - High-level IR
        - 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 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”?

```
B1  a = 10

B2  if input()  exit

B3  b = a
    a = 11
```
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

- $f_s$: A transfer function of a statement abstracts the execution with respect to the problem of interest.
- For a statement $s$ (d: $x = y + z$)
  \[
  \text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s] - \text{Kill}[s])
  \]
  - **Gen**[$s$]: definitions generated: $\text{Gen}[s] = \{d\}$
  - **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

- 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 $B$

\[ \text{out}[B] = f_B(\text{in}[B]) = f_{d2}f_{d1}f_{d0}(\text{in}[B]) \]
\[ = \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] \cup (\text{in}[B]\text{-Kill}[d_0]))\text{-Kill}[d_1])) -\text{Kill}[d_2] \]
\[ = \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] - \text{Kill}[d_1]) - \text{Kill}[d_2]) \cup \text{in}[B] - (\text{Kill}[d_0] \cup \text{Kill}[d_1] \cup \text{Kill}[d_2]) \]
\[ = \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B]) \]

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

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

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

// 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 {
    $\text{in}[B] = \cup \ (\text{out}[p])$, 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>
</table>
| Transfer function $f_b(x)$ | forward: $\text{out}[b] = f_b(\text{in}[b])$
| | $f_b(x) = \text{Gen}_b \cup (x - \text{Kill}_b)$
| | $\text{Gen}_b$: definitions in $b$
| | $\text{Kill}_b$: killed defs |
| Meet Operation | $\text{in}[b] = \bigcup \text{out}[\text{predecessors}]$ |
| Boundary Condition | $\text{out}[\text{entry}] = \emptyset$ |
| Initial interior points | $\text{out}[b] = \emptyset$ |
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

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

- **Boundary condition:**
Example

```
out[entry] --> entry
  
in[1] --> n = p
  |  \     if g
  v  \    
in[1] --- out[1]
  
in[2] --> r = n+r
  
  
in[3] --> m = n+q
  |    p = m
  v  
  
  
in[exit] --> exit
```

m = n + q
n = p
if g
Liveness: Iterative Algorithm

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

// 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 $\text{IN}$ occur) {
    For each basic block $B$ other than $\text{Exit}$ {
        $\text{out}[B] = \cup (\text{in}[s])$, for all successors of $B$
        $\text{in}[B] = f_B(\text{out}[B])$ \hspace{1em} // $\text{in}[B] = \text{Use}[B] \cup (\text{out}[B] - \text{Def}[B])$
    }
}
### IV. Framework

<table>
<thead>
<tr>
<th></th>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Sets of definitions</td>
<td>Sets of variables</td>
</tr>
<tr>
<td>Direction</td>
<td>forward: out[b] = f_b(in[b])</td>
<td>backward: in[b] = f_b(out[b])</td>
</tr>
<tr>
<td></td>
<td>in[b] = ∨ out[pred(b)]</td>
<td>out[b] = ∨ in[succ(b)]</td>
</tr>
<tr>
<td>Transfer function</td>
<td>f_b(x) = Gen_b ∪ (x -Kill_b)</td>
<td>f_b(x) = Use_b ∪ (x -Def_b)</td>
</tr>
<tr>
<td>Meet Operator (∧)</td>
<td>∪</td>
<td>∪</td>
</tr>
<tr>
<td>Boundary Condition</td>
<td>out[entry] = ∅</td>
<td>in[exit] = ∅</td>
</tr>
<tr>
<td>Initial Interior</td>
<td>out[b] = ∅</td>
<td>in[b] = ∅</td>
</tr>
<tr>
<td>points</td>
<td></td>
<td></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?