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

1. Program
2. Front end
3. Abstract Syntax Tree
4. High-level IR
   - High-level optimization
   - Parallelization
   - Loop transformations
6. Low-level IR
   - Low-level optimization
   - Redundancy elimination
7. 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
- Examples of optimizations:
  - Constant propagation
  - Common subexpression elimination
  - Dead code elimination

```
Value of x?
Which "definition" defines x?
Is the definition still meaningful (live)?
```

```latex
\begin{align*}
a &= b + c \\
d &= 7 \\
e &= b + c \\
a &= 243 \\
e &= d + 3 \\
g &= a
\end{align*}
```
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 $\text{in}[^b]$ and $\text{out}[^b]$ for all basic blocks $b$
  - Effect of code in basic block:
    - Transfer function $f_b$ relates $\text{in}[^b]$ and $\text{out}[^b]$, for same $b$
  - Effect of flow of control:
    - relates $\text{out}[^{b_1}]$, $\text{in}[^{b_2}]$ 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 $s$ ($d: x = y + z$)
  - $\text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s] - \text{Kill}[s])$
  - $\text{Gen}[s]$: definitions generated: $\text{Gen}[s] = \{d\}$
  - $\text{Propagated}$ definitions: $\text{in}[s] - \text{Kill}[s]$, where $\text{Kill}[s]=\text{set of all other defs to } x \text{ in the rest of program}$
Effects of a Basic Block

\[\text{in}[B_0]\]

\[
\begin{array}{c}
d_0: y = 3 \quad f_{d_0} \\
d_1: x = 10 \quad f_{d_1} \quad f_B = f_{d_1} \cdot f_{d_0}
\end{array}
\]

\[\text{out}[B_0]\]

- 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 \(B\):
  \[\text{Composition of transfer functions of statements in } B\]
  \[\text{out}[B] = f_d(\text{in}[B])\]
  \[= f_{d_1} \cdot f_{d_0}(\text{in}[B])\]
  \[= \text{Gen}[d_1] \cup (\text{Gen}[d_0] \cup (\text{in}[B] - \text{Kill}[d_0])) - \text{Kill}[d_1]\]
  \[= (\text{Gen}[d_1] \cup (\text{Gen}[d_0] - \text{Kill}[d_1])) \cup \text{in}[B] - (\text{Kill}[d_0] \cup \text{Kill}[d_1])\]
  \[= \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B])\]
  
  \(\text{Gen}[B]\): locally exposed definitions (available at end of \(B\))
  
  \(\text{Kill}[B]\): set of definitions killed by \(B\)

Effects of the Edges (acyclic)

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

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

Reaching Definitions: Iterative Algorithm

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

// 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 \( \text{Entry} \)
    \( \text{in}[B] = \cup (\text{out}[p]), \text{for all predecessors } p \text{ of } B \)
    \( \text{out}[B] = f_B(\text{in}[B]) \)  // \( \text{out}[B] = \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B]) \)
  }
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

\[
\begin{align*}
\text{def} & : \text{example:} \\
\text{def} & : m = n - q \\
\text{def} & : p = m \\
\text{out} & : f_b
\end{align*}
\]

• Direction: backward: in[b] = f_b(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 (\( \land \)):
  - \( \text{out}[b] = \text{in}[s_1] \cup \text{in}[s_2] \cup \ldots \cup \text{in}[s_n], s_1, \ldots, s_n \) 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 Exit
$\text{in}[B] = \emptyset$

// iterate
While (Changes to any $\text{in}[]$ occur) {
    For each basic block $B$ other than Exit {
        $\text{out}[B] = \cup (\text{in}[s]),$ for all successors $s$ of $B$
        $\text{in}[B] = f_n(\text{out}[B])$ // $\text{in}[B] = \text{Use}[B] \cup (\text{out}[B] - \text{Def}[B])$
    }
}
IV. Framework

<table>
<thead>
<tr>
<th>Domain</th>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets of definitions</td>
<td>Sets of variables</td>
<td></td>
</tr>
</tbody>
</table>

Direction

- **forward:**
  - \( \text{out}[b] = f_b(\text{in}[b]) \)
  - \( \text{in}[b] = \bigwedge \text{out}[\text{pred}(b)] \)
- **backward:**
  - \( \text{in}[b] = f_b(\text{out}[b]) \)
  - \( \text{out}[b] = \bigwedge \text{in}[\text{succ}(b)] \)

Transfer function

- \( f_b(x) = \text{Gen}_b \cup (x - \text{Kill}_b) \)
- \( f_b(x) = \text{Use}_b \cup (x - \text{Def}_b) \)

Meet Operation (\(\bigwedge\))

- \( \bigcup \)

Boundary Condition

- \( \text{out}[\text{entry}] = \emptyset \)
- \( \text{in}[\text{exit}] = \emptyset \)

Initial interior points

- \( \text{out}[b] = \emptyset \)
- \( \text{in}[b] = \emptyset \)

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?