Lecture 4
Introduction to Data Flow Analysis

I Structure of data flow analysis
II Example 1: Reaching definition analysis
III Example 2: Liveness analysis
IV Generalization

Reference: Chapter 8, 8.1-4

Data Flow Analysis

• Local analysis (e.g. value numbering)
  • analyze effect of each instruction
  • compose effects of instructions to derive information from beginning of basic block to each instruction

• Data flow analysis
  • analyze effect of each basic block
  • compose effects of basic blocks to derive information at basic block boundaries
  • (from basic block boundaries, apply local technique to generate information on instructions)
Effects of a basic block

- Effect of a statement: a = b+c
  - Uses variables (b, c)
  - Kills an old definition (old definition of a)
  - new definition (a)
- Compose effects of statements -> Effect of a basic block
  - A locally exposed use in a b.b. is a use of a data item which is not pre-ceded in the b.b. by a definition of the data item
  - any definition of a data item in the basic block kills all definitions of the same data item reaching the basic block.
  - A locally available definition = last definition of data item in b.b.

```
t1 = r1+r2
r2 = t1
t2 = r2+r1
r1 = t2
t3 = r1*r1
r2 = t3
if r2>100 goto L1
```

Across Basic Blocks

- Static program vs. dynamic execution

```
a = x
if input() exit
b = a
a = y
```

- Statically: Finite program
  - Dynamically: Potentially infinite possible execution paths
- Can reason about each possible path as if all instructions executed are in one basic block
- Data flow analysis:
  - Associate with each static point in the program information true of the set of dynamic instances of that program point
II. Reaching Definitions

- A definition of a variable $x$ is a statement that assigns, or may assign, a value to $x$.

- 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 along that path.

Problem statement
- For each basic block $b$, determine if each definition in the program reaches $b$

A representation:
- $\text{IN}[b], \text{OUT}[B]$: a bit vector, one bit for each definition

Describing Effects of the Nodes (basic blocks)

Schema: $\text{IN}[b] \xrightarrow{f_b} \text{OUT}[b] = f_b(\text{IN}[b])$

Example:

- $a = x$
- $b = a$
- $a = y$
- if input() -> exit

- $d1: a = 10$
- $d2: b = 11$
- if e
- $d3: a = 1$
- $d4: b = 2$
- $d5: c = a$
- $d6: a = 4$

- A transfer function $f_b$ of a basic block $b$:
  \[ \text{OUT}[b] = f_b(\text{IN}[b]) \]
  incoming reaching definitions -> outgoing reaching definitions

- A basic block $b$
  - generate definitions: $\text{Gen}[b]$, set of locally available definitions in $b$
  - propagate definitions: $\text{in}[b] - \text{Kill}[b]$, where $\text{Kill}[b]$=set of defs (in rest of program) killed by defs in $b$

- $\text{out}[b] = \text{Gen}[b] \cup (\text{in}(b) - \text{Kill}[b])$
**Effects of the Edges (acyclic)**

- $\text{out}[b] = f_b(\text{in}[b])$
- Join node: a node with multiple predecessors
- **meet** operator:  
  $$\text{in}[b] = \text{out}[p_1] \cup \text{out}[p_2] \cup ... \cup \text{out}[p_n], \text{where}$$
  $$p_1, ..., p_n \text{ are all predecessors of } b$$

---

**Cyclic Graphs**

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

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

// Initialize
out[Entry] = ∅  // can set out[Entry] to special def
// if reaching then undefined use
For all nodes i
out[i] = ∅  // can optimize by out[i]=gen[i]
ChangedNodes = N

// iterate
While ChangedNodes ≠ ∅ {
   Remove i from Changed Nodes
   in[i] = U (out[p]), for all predecessors p of i
   oldout = out[i]
   out[i] = f_i(in[i])  // out[i]=gen[i]U(in[i]-kill[i])
   if oldout ≠ out[i] {
      for all successors s of i
         add s to ChangedNodes
   }
}

**Example**

```
B1
entry
  d1: i = m-1
  d2: j = n
  d3: a = u1

B2
  d4: i = i+1
  d5: j = j-1

B3
  d6: a = u2

B4
  d7: i = u3

exit
```
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**.

- **Motivation**
  - e.g. register allocation
    
    ```plaintext
    for i = 0 TO n
      .. i ..
    ...
    for i = 0 to n
      .. i ..
    ```

- **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 backwards to the definitions

<table>
<thead>
<tr>
<th>Execution Path</th>
<th>Control Flow</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>def</td>
<td>IN[b]</td>
<td>d3: a = 1</td>
</tr>
<tr>
<td>def</td>
<td>OUT[b]</td>
<td>d4: b = 1</td>
</tr>
<tr>
<td>use</td>
<td>fb</td>
<td>d5: c = a</td>
</tr>
<tr>
<td></td>
<td>fb(OUT[b])</td>
<td>d6: a = 4</td>
</tr>
</tbody>
</table>

- **A basic block \( b \) can**
  - generate live variables:
    - \( \text{Use}[b] \), set of locally exposed uses in \( b \)
  - propagate incoming live variables: \( \text{OUT}[b] - \text{Def}[b] \)
    - where \( \text{Def}[b] = \) set of variables defined in \( b \).
  - **transfer function** for block \( b \):
    - \( \text{in}[b] = \text{Use}[b] \cup (\text{out}(b) - \text{Def}[b]) \)
**Flow Graph**

- \( \text{in}[b] = f_b(\text{out}[b]) \)
- Join node: a node with multiple successors
- **meet** operator:
  \[
  \text{out}[b] = \text{in}[s_1] \cup \text{in}[s_2] \cup \ldots \cup \text{in}[s_n], \text{where} \ s_1, \ldots, s_n \text{ are all successors of } b
  \]

---

**Live Variable: Worklist Algorithm**

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

```plaintext
// Initialize
\text{in}[^{\text{Exit}}] = \emptyset \quad //\text{local variables}
\text{for all nodes } i
\text{in}[i] = \emptyset \quad //\text{can optimize by } \text{in}[i]=\text{use}[i]
\text{ChangedNodes} = N

// iterate
\text{While} \text{ChangedNodes} \neq \emptyset \{ 
    \text{Remove } i \text{ from Changed Nodes}
    \text{out}[i] = \bigcup (\text{in}[s]), \text{for all successors } s \text{ of } i
    \text{oldin} = \text{in}[i]
    \text{in}[i] = f_i(\text{out}[i]) \quad //\text{in}[i]=\text{use}[i]\cup(\text{out}[i]-\text{def}[i])
    \text{if } \text{oldin} \neq \text{in}[i] \{ 
        \text{for all predecessors } p \text{ of } i
        \quad \text{add } p \text{ to ChangedNodes}
    \}
\}
```
**Example**

```
entry

B1

\[\begin{align*}
  d1: & \quad i = m-1 \\
  d2: & \quad j = n \\
  d3: & \quad a = u_1 \\
\end{align*}\]

\[\begin{align*}
  d4: & \quad i = i+1 \\
  d5: & \quad j = j-1 \\
\end{align*}\]

B2

B3

\[d6: \quad a = u_2\]

B4

\[d7: \quad i = u_3\]

exit
```

### IV. Framework

<table>
<thead>
<tr>
<th>Domain</th>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transfer function (f_b(x))</td>
<td>Sets of definitions</td>
<td>Sets of variables</td>
</tr>
<tr>
<td>direction of function</td>
<td>forward: out[b] = (f_b(\text{in}[b]))</td>
<td>backward: (\text{in}[b] = f_b(\text{out}[b]))</td>
</tr>
<tr>
<td>Generate</td>
<td>(\text{Gen}_b: \text{definitions in } b)</td>
<td>(\text{Use}_b: \text{var. used in } b)</td>
</tr>
<tr>
<td>Propagate</td>
<td>(\text{in}[b]-\text{Kill}_b: \text{killedDefs})</td>
<td>(\text{out}[b]-\text{Def}_b: \text{var defined})</td>
</tr>
<tr>
<td>Merge operation</td>
<td>(\text{U}(\text{in}[b]=\text{U out}\text{predecessors}))</td>
<td>(\text{U}(\text{out}[b]=\text{U in}\text{successors}))</td>
</tr>
<tr>
<td>Initialization</td>
<td>(\text{out}[\text{entry}] = \emptyset)</td>
<td>(\text{in}[\text{exit}] = \emptyset)</td>
</tr>
<tr>
<td></td>
<td>(\text{out}[b] = \emptyset)</td>
<td>(\text{in}[b] = \emptyset)</td>
</tr>
</tbody>
</table>
Questions

- **Correctness**
  - equations are satisfied, if the program terminates.

- **Precision: how good is the answer?**
  - is the answer ONLY a union of all possible executions?

- **Convergence: will the analysis terminate?**
  - or, will there always be some nodes that change?

- **Speed: how fast is the convergence?**
  - how many times will we visit each node?