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

What is 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

What is Data Flow Analysis? (Cont.)

- 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"?
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 preceded 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.

\[
\begin{align*}
  t_1 &= r_1 + r_2 \\
  t_2 &= t_1 \\
  t_3 &= t_2 + r_1 \\
  r_1 &= t_3 \\
  r_2 &= t_3 \\
  \text{if } r_2 > 100 \text{ goto } L1
\end{align*}
\]

II. Reaching Definitions

- Every assignment is a definition
  - A definition \( d \) reaches a point \( p \) if there exists 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

Reaching Definitions: Another Example

\[
\begin{align*}
  d0: & \quad a = x \\
  L1: & \quad \text{if input}() \ \text{GOTO } L2 \\
  d1: & \quad b = a \\
  d2: & \quad a = y \\
  \text{GOTO } L1
\end{align*}
\]

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
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 \( \text{in}[b] \) 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 \( \text{Kill}[s] \)-set of all other defs to \( x \) in the rest of program

Effects of a Basic Block

- \( \text{out}[B] = f_B(\text{in}[B]) = \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B]) \)
  - \( \text{Gen}[B] \): locally exposed definitions (available at end of bb)
  - \( \text{Kill}[B] \): set of definitions killed by \( B \)

Example

- A transfer function \( f_b \) of a basic block \( B \):
  \( \text{OUT}[B] = f_b(\text{IN}[B]) \)
  incoming reaching definitions \( \rightarrow \) outgoing reaching definitions
- A basic block \( B \)
  - generates definitions: \( \text{Gen}[B] \)
    - set of locally available definitions in \( B \)
  - kills 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:
  \( \text{in}[b] = \text{out}[p_1] \cup \text{out}[p_2] \cup \ldots \cup \text{out}[p_n] \), where
  \( p_1, \ldots, p_n \) are all predecessors of \( b \)

### Cyclic Graphs

\[
\begin{align*}
\text{out} & : 1 \to 2 \to 1 \\
\text{in} & : 1 \to 2 \to 3 \\
\text{out} & : 2 \to 1 \to 3 \\
\text{in} & : 3 \to 2 \to 3 \\
\text{out} & : 3 \to 1 \to 2 \\
\end{align*}
\]

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

---

### Reaching Definitions: Iterative Algorithm

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

1. **Boundary condition**
   - \( \text{out}[\text{Entry}] = \emptyset \)

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

3. **Iterate**
   - While (Changes to any \( \text{out}[\cdot] \) occur)
     - For each basic block \( B \) other than \( \text{Entry} \)
       - \( \text{in}[B] = \bigcup \{ \text{out}[p] \}, \) for all predecessors \( p \) of \( B \)
       - \( \text{out}[B] = f_B(\text{in}[B]) \)  // \( \text{out}[B] = \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B]) \)

---

### Reaching Definitions: Worklist Algorithm

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

1. **Initialize**
   - \( \text{out}[\text{Entry}] = \emptyset \)  // can set out[Entry] to special def
   - \( \text{out}[i] = \emptyset \)  // if reaching then undefined use
   - \( \text{ChangesNodes} = N \)

2. **Iterate**
   - While \( \text{ChangesNodes} \neq \emptyset \)
     - Remove \( i \) from \( \text{ChangesNodes} \)
     - \( \text{in}[i] = \bigcup \{ \text{out}[p] \}, \) for all predecessors \( p \) of \( i \)
     - \( \text{out}[i] = f_i(\text{in}[i]) \)  // \( \text{out}[i] = \text{gen}[i] \bigcup (\text{in}[i] - \text{kill}[i]) \)
     - If \( \text{oldout} \neq \text{out}[i] \)
       - For all successors \( s \) of \( i \)
         - Add \( s \) to \( \text{ChangesNodes} \)

---
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
    
    ```
    for i = 0 to n
    \text{...}
    \text{...}
    i \text{...}
    \text{...}
    ```

- 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)

- Insight: Trace uses backwards to the definitions
  - an execution path
    - control flow
    - example
      
      ```
      \text{def} \quad \text{IN}[b] = f_b(\text{OUT}[b])
      \text{def} \quad b \quad f_b
      \text{use} \quad \text{OUT}[b] \quad d3: a = 1
      \text{def} \quad d4: b = 1
      \text{def} \quad d5: c = a
      \text{def} \quad d6: a = 4
      ```

- 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] \cdot \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])
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 s of B
in[B] = f_B(out[B]) // in[B]=Use[B] ∪(out[B]-Def[B])
}

Example

![Example Diagram with basic blocks B1 to B4 and operations d1 to d7]

IV. Framework

<table>
<thead>
<tr>
<th>Reaching Definitions</th>
<th>Live Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Sets of definitions</td>
</tr>
<tr>
<td>Direction</td>
<td>forward: out[b] = f_B(in[b]) in[b] = ∨ out[pred(b)]</td>
</tr>
<tr>
<td>Transfer function</td>
<td>f_B(x) = Gen_B(x) ∪ (x - Kill_B)</td>
</tr>
<tr>
<td>Meet Operation (-)</td>
<td>⊔</td>
</tr>
<tr>
<td>Boundary Condition</td>
<td>out[entry] = ∅</td>
</tr>
<tr>
<td>Initial interior points</td>
<td>out[b] = ∅</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?

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?