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

```
\begin{align*}
a &= b + c \\
d &= 7
\end{align*}
```

```
\begin{align*}
e &= b + c \\
a &= 243 \\
e &= d + 3 \\
g &= a
\end{align*}
```

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”?

```
B1  a = 10

B2  if input() → exit
B3  b = a
    a = 11
```
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 \( \rightarrow \) 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.

\[
\begin{align*}
t1 &= r1 + r2 \\
r2 &= t1 \\
t2 &= r2 + r1 \\
r1 &= t2 \\
t3 &= r1 \times r1 \\
r2 &= t3 \\
if \ r2 > 100 \ goto \ L1
\end{align*}
\]
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 = $\#\text{defs}$
Reaching Definitions: Another Example

```
d0: a = x

L1: if input() GOTO L2

L2: ...
```

```
d1: b = a
d2: a = y
GOTO L1
```
• 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

\[ \text{in}[^{B0}] \]

\[
\begin{align*}
\text{d0: } & y = 3 & f_{d0} \\
\text{d1: } & x = 10 & f_{d1} \\
\text{d2: } & y = 11 & f_{d2}
\end{align*}
\]

\[ \text{out}[^{B0}] \]

• \( f_s \): A transfer function of a statement
  – abstracts the execution with respect to the problem of interest

• For a statement \( s \) (d: \( x = y + z \))
  \[
  \text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s]-\text{Kill}[s])
  \]
  – \textbf{Gen}[s]: definitions \textit{generated}: \text{Gen}[s] = \{d\}
  – \textbf{Propagated} definitions: in[s] - 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

\[
in[B0] \quad \begin{align*}
\text{d0: } & y = 3 & f_{d0} \\
\text{d1: } & x = 10 & f_{d1} \\
\text{d2: } & y = 11 & f_{d2}
\end{align*}
\]

\[
\text{out}[B0] = f_B = f_{d2} \cdot f_{d1} \cdot f_{d1}
\]

- Transfer function of a statement \( s \):
  - \( \text{out}[s] = f_s(\text{in}[s]) = \text{Gen}[s] \cup (\text{in}[s] \setminus \text{Kill}[s]) \)
- Transfer function of a basic block \( B \):
  - Composition of transfer functions of statements in \( B \)
  - \( \text{out}[B] = f_B(\text{in}[B]) = f_{d2} f_{d1} f_{d0}(\text{in}[B]) \)
    \[
    = \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] \cup (\text{in}[B] \setminus \text{Kill}[d_0]) \setminus \text{Kill}[d_1]) \setminus \text{Kill}[d_2])
    \]
  - \( \text{Gen}[B] \): locally exposed definitions (available at end of bb)
  - \( \text{Kill}[B] \): set of definitions killed by \( B \)
Example

\[ \begin{align*}
B_0: & \quad d0: \quad y = 3 \\
& \quad d1: \quad x = 10 \\
& \quad d2: \quad y = 11 \\
& \quad \text{if } e \\
B_1: & \quad d3: \quad x = 1 \\
& \quad d4: \quad y = 2 \\
B_2: & \quad d5: \quad z = x \\
& \quad d6: \quad x = 4
\end{align*} \]

- 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 \( b \)
  - 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
- \textbf{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**

- Equations still hold
  - $\text{out}[b] = f_b(\text{in}[b])$
  - $\text{in}[b] = \text{out}[p_1] \cup \text{out}[p_2] \cup ... \cup \text{out}[p_n], p_1, ..., p_n$ 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 \( \text{out}[\cdot] \) occur) {
    For each basic block \( B \) other than \( \text{Entry} \) {
        \[
        \text{in}[B] = \bigcup (\text{out}[p]), \text{ for all predecessors } p \text{ of } B
        \]
        \[
        \text{out}[B] = f_B(\text{in}[B]) \quad \text{// } \text{out}[B] = \text{gen}[B] \cup (\text{in}[B] - \text{kill}[B])
        \]
    }
}
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 ChangedNodes
        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

d1: i = m-1

d2: j = n

d3: a = u1

B1

d4: i = i+1

d5: j = j-1

B2

d6: a = u2

B3

d7: i = u3

B4

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
    ```c
    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)

- **Insight:** Trace uses backwards to the definitions

  an execution path

  control flow

  example

  ![Diagram](image)

  \[ \text{IN}[b] = f_b(\text{OUT}[b]) \]

  \[ \text{d3: } a = 1 \]

  \[ \text{d4: } b = 1 \]

- **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

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

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

// Boundary condition
\[
in[\text{Exit}] = \emptyset
\]

// Initialization for iterative algorithm
For each basic block $B$ other than $\text{Exit}$
\[
in[B] = \emptyset
\]

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

entry

B1
- 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
## IV. 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(in[b])</td>
<td>in[b] = f_b(out[b])</td>
</tr>
<tr>
<td></td>
<td>in[b] = ( \land ) out[pred(b)]</td>
<td>out[b] = ( \land ) in[succ(b)]</td>
</tr>
<tr>
<td>Transfer function</td>
<td>( f_b(x) = Gen_b \cup (x - Kill_b) )</td>
<td>( f_b(x) = Use_b \cup (x - Def_b) )</td>
</tr>
<tr>
<td>Meet Operation (( \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>
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?

```
entry
  out[entry]={}
  in[1]={}
  out[1]={}
  in[2]={d1}
  out[2]={d1}
  in[3]={d1}
  out[3]={d1}
  in[exit]
```

- 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?