Lecture 5

Introduction to Data Flow Analysis

I. Structure of data flow analysis

II. Example 1: Reaching definition analysis

III. Example 2: Liveness analysis

IV. Framework
What is This?

Course Dependence Graph

Schedule of Lectures Has Been Updated
Review: Expression DAG

Example 1:
- grammar (for bottom-up parsing): 
  \[ E \rightarrow E + T \mid E - T \mid T, \quad T \rightarrow T*F \mid F, \quad F \rightarrow (E) \mid \text{id} \]
- expression: 
  \[ a + a \cdot (b - c) + (b - c) \cdot d \]
Review: Value Numbering

Data structure:
VALUES = Table of
    expression /* [OP, valnum1, valnum2] */
    var /* name of variable currently holding expr */

var2value() /* variable’s current value number */

\[
\begin{align*}
  a &= b + c \\
  t1 &= b + c \\
  a &= t1 \\
  b &= a - d \\
  t2 &= t1 - d \\
  b &= t2 \\
  c &= b + c \\
  t3 &= t2 + c \\
  c &= t3 \\
  d &= a - d \\
  d &= t2
\end{align*}
\]
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

[ALSU 9.2]
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

For each variable x, determine:

- Value of x?
- Which “definition” defines x?
- Is the definition still meaningful (live)?

\[
\begin{align*}
\text{a} &= \text{b} + \text{c} \\
\text{d} &= 7 \\
\text{e} &= \text{b} + \text{c} \\
\text{a} &= 243 \\
\text{e} &= \text{d} + 3 \\
\text{g} &= \text{a}
\end{align*}
\]
Static Program vs. Dynamic Execution

- **Specially**: 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”?

```plaintext
a = 10
if input()
b = a
exit
B1
B2
B3
b = a
a = 11
```
Effects of a Basic Block

• Effect of a statement: \( a = b + c \)
  - **Uses** source variables (b and c)
  - **Kills** an old definition (old definition of a)
  - Defines 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 that is not preceded in the b.b. by a definition of the data item.
  - Any definition of a data item in the b.b. **kills** all other definitions of the same data item.
  - A **locally available definition** = last definition of data item in b.b.

\[
\begin{align*}
  t1 & = r1 + r2 \\
  r2 & = t1 \\
  t2 & = r2 + r1 \\
  r1 & = t2 \\
  t3 & = r1 \times r1 \\
  r2 & = t3 \\
  \text{if } r2 > 100 & \text{ goto L1}
\end{align*}
\]

- **locally exposed uses?** \( r1 \)
- **kills any definitions?** Any other defn of \( t2 \) in program
- **locally available definition?** \( t2 \)
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.
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

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

\text{d2 reaches this point? yes}
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], in[b'] if b and b' are adjacent
- Find a solution to the equations
Effects of a Statement

\[\text{in[B1]}\]

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

\[\text{out[B1]}\]

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

Transfer function of a basic block $B$:
- Composition of transfer functions of statements in $B$
  
  $\text{out}[B] = f_B(in[B]) = f_d_2 f_d_1 f_d_0 (in[B])$
  
  $\text{out}[B] = \text{Gen}[d_2] \cup ((\text{Gen}[d_1] \cup ((\text{Gen}[d_0] \cup (in[B]-\text{Kill}[d_0]))) \cup \text{Kill}[d_1])) - \text{Kill}[d_2])$
  
  $\text{out}[B] = \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] - \text{Kill}[d_1]) - \text{Kill}[d_2]) \cup (in[B] - (\text{Kill}[d_0] \cup \text{Kill}[d_1] \cup \text{Kill}[d_2]))$

  $\text{out}[B] = \text{Gen}[B] \cup (in[B] - \text{Kill}[B])$

- **Gen**[$B$]: locally available definitions (defined locally & reaches end of bb)
- **Kill**[$B$]: set of definitions killed by $B
Reaching Definitions Example

• **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 definitions in $b$ that reach end of $b$
  • **kills** definitions: $\text{in}[b] - \text{Kill}[b]$,
    where $\text{Kill}[b]$=set of defs killed by defs in $b$
  • $\text{out}[b] = \text{Gen}[b] \cup (\text{in}(b) - \text{Kill}[b])$

<table>
<thead>
<tr>
<th>$f$</th>
<th>Gen</th>
<th>Kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{1,2}</td>
<td>{0,2,3,4,5}</td>
</tr>
<tr>
<td>2</td>
<td>{3,4}</td>
<td>{0,1,2,5}</td>
</tr>
<tr>
<td>3</td>
<td>{5,6}</td>
<td>{1,3}</td>
</tr>
</tbody>
</table>

Subtlety: $d0 \& d2$ kill each other, but $d2$ is still generated
Effects of the Edges (acyclic)

• \( \text{out}[b] = f_b(\text{in}[b]) \)
• Join node: a node with multiple predecessors
• \textbf{meet} operator:
  \[
  \text{in}[b] = \text{out}[p_1] \cup \text{out}[p_2] \cup \ldots \cup \text{out}[p_n], \quad \text{where} \quad p_1, \ldots, p_n \text{ are all the predecessors of } b
  \]

\[\text{in}[\text{exit}] = \text{out}[B2] \cup \text{out}[B3]\]
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

\[ \text{in[B2]} = \text{out[B1]} \cup \text{out[B3]} \]
Reaching Definitions: Iterative Algorithm

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

// Boundary condition
out[Entry] = Ø

// Initialization for iterative algorithm
For each basic block B other than Entry
out[B] = Ø

// iterate
While (changes to any out[] occur) {
For each basic block B other than Entry {
in[B] = U(out[p]), for all predecessors p of B
}
}
Reaching Definitions: Worklist Algorithm

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

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

// iterate
While ChangedNodes $\neq \emptyset$
    Remove $i$ from ChangedNodes
    in[$i$] = $\bigcup$ out[$p$], for all predecessors $p$ of $i$
    oldout = out[$i$]
    out[$i$] = $f_i$(in[$i$])  // out[$i$]=gen[$i$] $\bigcup$ (in[$i$]-kill[$i$])
    if (oldout $\neq$ out[$i$]) {
        for all successors $s$ of $i$
            add s to ChangedNodes
    }
IN[b] = U(OUT[pred(b)])

OUT[b] = Gen[b] U (IN(b)-Kill[b])

Init: OUT[b]=∅

Fixed point!
A valid solution to Reaching Definitions?

- Is the solution a fixed point? **yes**  Is it valid? **no**
- Will the worklist algorithm generate this answer? **no**
- What if add control flow edge shown in red? **yes**

**Fixed point:** another iteration of algorithm won’t change in/out values

```
entry
out[entry]={}
in[1]={}
out[1]={}
in[exit]=
out[2]=d1
out[2]=d1
in[3]=d1
out[3]=d1
in[exit]=
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
    
    ```
    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
Live Variables: Effects of a Basic Block (Transfer Function)

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

  - an execution path
  - control flow

```
  def
  def
  use
```

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

---

**example**

```
d4: d = 1
d5: c = a
d6: a = 4
```

```
IN[b] =
{a} \cup (OUT[b] - \{a,c,d\})
```
Flow Graph

- in[b] = f_b(out[b])
- Join node: a node with multiple successors
- meet operator:
  \[ \text{out[b]} = \text{in}[s_1] \cup \text{in}[s_2] \cup ... \cup \text{in}[s_n], \text{where} \]
  \[ s_1, ..., s_n \text{ are all successors of b} \]
Live Variables: 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] = U(in[s]), for all successors s of B
}
}
**Live Variables Example**

\[ \text{INIT: } \text{IN}[b] = \emptyset \]

First Pass:
- **entry**: \( \text{OUT}[\text{entry}] = \{m,n,u1,u2,u3\} \)
- **B1**: \( \text{IN}[B1] = \{m,n,u1,u2,u3\} \)
- **B2**: \( \text{OUT}[B2] = \{i,j,u2,u3\} \)
- **B3**: \( \text{IN}[B3] = \{u2,u3\} \)
- **B4**: \( \text{OUT}[B4] = \{u3\} \)

Second Pass:
- **B1**: \( \text{OUT}[B1] = \{m,n,u1,u2,u3\} \)
- **B2**: \( \text{IN}[B2] = \{i,j,u2,u3\} \)
- **B3**: \( \text{OUT}[B3] = \{j,u2,u3\} \)
- **B4**: \( \text{IN}[B4] = \{j,u2,u3\} \)
- **exit**: \( \text{OUT}[\text{exit}] = \{i,j,u2,u3\} \)

Fixed point!
IV. Framework

<table>
<thead>
<tr>
<th>Domain</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:</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) = \text{Gen}_b \cup (x - \text{Kill}_b) )</td>
<td>( f_b(x) = \text{Use}_b \cup (x - \text{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>

Other Data Flow Analysis problems fit into this general framework, e.g., Available Expressions [ALSU 9.2.6]
Key Questions

• Correctness
  • equations are satisfied, if the analysis algorithm terminates.

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

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

• Speed: how fast is the convergence?
  • how many times will we visit each node?
Today’s Class

I. Structure of data flow analysis

II. Example 1: Reaching definition analysis

III. Example 2: Liveness analysis

IV. Framework

Monday’s Class

• Foundations of Data Flow Analysis
  – ALSU 9.3