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

Review: Expression DAG
Example 1:
- grammar (for bottom-up parsing):
  \[ E \rightarrow E + T \mid E - T \mid T \rightarrow T \ast F \mid F \rightarrow (E) \mid id \]
- expression: \[ a + a \ast (b - c) + (b - c) \ast d \]

Parse tree

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 */

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

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

Examples of optimizations:
- Constant propagation
- Common subexpression elimination
- Dead code elimination

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 \((\text{old definition of } a)\)
  - 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.

Example of a basic block:
\[a = b + c\]
\[d = 7\]
\[e = d + 3\]
\[g = a\]

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

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
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: Another Example

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

- \( f_s \): A transfer function of a statement
  - abstracts the execution with respect to the problem of interest
- For a statement \( s \) (e.g., \( d: x = y + z \))
  - out[\( s \)] = transfer function of \( s \) = Gen[\( s \)] U Prop[\( s \)]
  - Gen[\( s \)]: definitions generated: Gen[\( s \)] = \{ \( d \) \}
  - Propagated definitions: Prop[\( s \)] = set of all other defs to \( x \) in the rest of the program
**Effects of a Basic Block**

- 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 $B$:
  - Composition of transfer functions of statements in $B$
  \[ \text{out}[B] = f_B(\text{in}[B]) = f_2 \circ f_1 \circ f_0(\text{in}[B]) \]
  \[ = \text{Gen}[d_2] \cup (\text{Gen}[d_1] \cup (\text{Gen}[d_0] \cup (\text{in}[B] - \text{Kill}[d_0])) - \text{Kill}[d_1])) - \text{Kill}[d_2] \]
  \[ = \text{Gen}[B] \cup (\text{in}[B] - \text{Kill}[B]) \]

- $\text{Gen}[B]$: locally available definitions (defined locally & reaches end of $B$)
- $\text{Kill}[B]$: set of definitions killed by $B$.

---

**Example**

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

- A basic block $b$ that generates definitions $\text{Gen}[b]$,
  - set of definitions in $b$ that reach end of $b$
- kills definitions $\text{Kill}[b]$, set of definitions (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:
  \[ \text{in}(b) = \text{out}(p_1) \cup \text{out}(p_2) \cup ... \cup \text{out}(p_n) \]
  where $p_1, ..., p_n$ are all predecessors of $b$

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

---

**Cyclic Graphs**

- Equations still hold:
  \[ \text{out}(b) = f_b(\text{in}(b)) \]
  \[ \text{in}(b) = \text{out}(p_1) \cup \text{out}(p_2) \cup ... \cup \text{out}(p_n) \]
  where $p_1, ..., p_n$ are all predecessors of $b$

- Find: fixed point solution
Reaching Definitions: Iterative Algorithm

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

// Boundary condition
\( \text{out}[\text{Entry}] = \emptyset \)

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

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

Reaching Definitions: Worklist Algorithm

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

// Initialize
\( \text{out}[\text{Entry}] = \emptyset \) // can set \( \text{out}[\text{Entry}] \) to special def
\( \text{out}[\cdot] = \emptyset \) // if reaching then undefined use
For all nodes \( i \)
\( \text{out}[i] = \emptyset \) // can optimize by \( \text{out}[i] = \text{gen}[i] \)
\( \text{ChangedNodes} = N \)

// iterate
While \( \text{ChangedNodes} \neq \emptyset \) {
    Remove \( i \) from \( \text{ChangedNodes} \)
    \( \text{in}[i] = \bigcup (\text{out}[p]) \), for all predecessors \( p \) of \( i \)
    \( \text{out}[i] = f_i(\text{in}[i]) \) // out\( [i] = \text{gen}[i] \cup (\text{in}[i]-\text{kill}[i]) \)
    \( \text{changed} = \{(\text{oldout} \neq \text{out}[i]) \} \)
    for all successors \( s \) of \( i \)
        add \( s \) to \( \text{ChangedNodes} \)
}

Reaching Definitions Example

\begin{align*}
\text{entry} & \\
B1 & \downarrow \text{d1: } i = m-1 \quad \text{d2: } j = n \quad \text{d3: } a = u1 \\
B2 & \text{d4: } i = i+1 \quad \text{d5: } j = j-1 \\
B3 & \text{d6: } a = u2 \\
B4 & \text{d7: } i = u3 \\
\text{exit} & \\
\end{align*}

First Pass | Second Pass
---|---
\( \text{IN}[B1] \) | 000 00 00 | 000 00 00
\( \text{OUT}[B1] \) | 000 11 11 | 000 11 11
\( \text{IN}[B2] \) | 111 00 00 | 111 00 00
\( \text{OUT}[B2] \) | 001 11 10 | 001 11 10
\( \text{IN}[B3] \) | 000 11 10 | 000 11 10
\( \text{OUT}[B3] \) | 000 11 10 | 000 11 10
\( \text{IN}[B4] \) | 000 11 10 | 000 11 10
\( \text{OUT}[B4] \) | 001 01 11 | 001 01 11
\( \text{IN}[\text{exit}] \) | 001 01 11 | 001 01 11

A legal solution to Reaching Definitions?

\begin{align*}
\text{entry} & \\
\text{out}[\text{entry}] = & \emptyset \\
\text{in}[1] = & i \\
\text{out}[1] = & i \\
\text{in}[2] = & \{d1\} \\
\text{out}[2] = & \{d1\} \\
\text{in}[3] = & \{d1\} \\
\text{out}[3] = & \{d1\} \\
\text{in}[\text{exit}] & \\
\end{align*}

- Will the worklist algorithm generate this answer? No
- What if add control flow edge shown in red? Yes

another iteration of algorithm won't change in/out values
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

Effects of a Basic Block (Transfer Function)

- **Insight:** Trace uses backwards to the definitions
  - an execution path
  - control flow

```
def
IN[b] = \text{Def}[b]
d3: a = 1
d4: b = 1
```
```
def
OUT[b] = \text{Use}[b]
d5: c = a
d6: d = 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
\text{in}[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{in}[B] = \emptyset
  \text{out}[B] = \bigcup \text{in}[s] \text{ for all successors } s \text{ of } B
  \text{in}[B] = f_b(\text{out}[B]) // \text{in}[B]=\text{Use}[B] \cup (\text{out}[B] - \text{Def}[B])
}
```
Live Variables Example

\[
\begin{align*}
\text{entry} & : d_1: i = m-1 \\
& : d_2: j = n \\
& : d_3: a = u_1 \\
B_1 & : d_4: i = i+1 \\
& : d_5: j = j-1 \\
& : d_6: a = u_2 \\
B_2 & : d_7: i = u_3 \\
B_3 & : d_8: j = u_3 \\
B_4 & : d_9: a = u_3 \\
\text{exit} &
\end{align*}
\]

First Pass

\[
\begin{align*}
\text{OUT}[\text{entry}] & : \{m,n,u_1,u_2,u_3\} \\
\text{IN}[B_1] & : \{m,n,u_1,u_2,u_3\}
\end{align*}
\]

\[
\begin{align*}
\text{OUT}[B_1] & : \{i,j,u_2,u_3\} \\
\text{IN}[B_2] & : \{i,j,u_2,u_3\}
\end{align*}
\]

\[
\begin{align*}
\text{OUT}[B_2] & : \{u_2,u_3\} \\
\text{IN}[B_3] & : \{u_2,u_3\}
\end{align*}
\]

\[
\begin{align*}
\text{OUT}[B_3] & : \{u_3\} \\
\text{IN}[B_4] & : \{u_3\}
\end{align*}
\]

\[
\begin{align*}
\text{OUT}[B_4] & : \{\} \\
\end{align*}
\]

Second Pass

\[
\begin{align*}
\text{OUT}[\text{entry}] & : \{m,n,u_1,u_2,u_3\} \\
\text{IN}[B_1] & : \{m,n,u_1,u_2,u_3\}
\end{align*}
\]

\[
\begin{align*}
\text{OUT}[B_1] & : \{i,j,u_2,u_3\} \\
\text{IN}[B_2] & : \{i,j,u_2,u_3\}
\end{align*}
\]

\[
\begin{align*}
\text{OUT}[B_2] & : \{u_2,u_3\} \\
\text{IN}[B_3] & : \{u_2,u_3\}
\end{align*}
\]

\[
\begin{align*}
\text{OUT}[B_3] & : \{u_3\} \\
\text{IN}[B_4] & : \{u_3\}
\end{align*}
\]

\[
\begin{align*}
\text{OUT}[B_4] & : \{i,j,u_2,u_3\}
\end{align*}
\]

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: ( \text{out}[b] = f_{\text{out}}(b) ) | ( \text{in}[b] = \text{out}[\text{pred}(b)] )</td>
</tr>
<tr>
<td>Transfer function</td>
<td>( f_{\text{out}}(x) = \text{Gen}_b \cup (x - \text{Kill}_b) )</td>
</tr>
<tr>
<td>Meet Operation ((\cup))</td>
<td>( \cup )</td>
</tr>
<tr>
<td>Boundary Condition</td>
<td>( \text{out}[\text{entry}] = \emptyset )</td>
</tr>
<tr>
<td>Initial interior points</td>
<td>( \text{out}[b] = \emptyset )</td>
</tr>
</tbody>
</table>

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

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?

Wednesday’s Class

- Foundations of Data Flow Analysis
  - ALSU 9.3