Program Analysis – Lecture 5
Data Flow Analysis (Part 2)

Prof. Dr. Michael Pradel
Software Lab, University of Stuttgart
Winter 2019/2020
Warm-up Quiz

What does the following code print?

```
var a, b;
var x = {};
x[a] = 23;
console.log(x[b]);
```

Nothing    23    undefined    false
Warm-up Quiz

What does the following code print?

```javascript
var a, b;
var x = {};
x[a] = 23;
console.log(x[b]);
```

Nothing  23  undefined  false
What does the following code print?

```javascript
var a, b;
var x = {};
x[a] = 23;
console.log(x[b]);
```

- Have value `undefined`
- Write and then read "undefined" property of `x`

---

23 undefined false
Data Flow Analysis

Basic idea

- Propagate analysis information along the edges of a control flow graph
- Goal: Compute analysis state at each program point
- For each statement, define how it affects the analysis state
- For loops: Iterate until fix-point reached
Available Expressions: Example

```javascript
var x = a + b;
var y = a * b;
while (y > a + b) {
    a = a + 1;
    x = a + b;
}
```
Available Expressions: Example

```javascript
var x = a + b;

var y = a * b;

while (y > a + b) {
    a = a + 1;
    x = a + b;
}
```

Available every time execution reaches this point
Control flow graph

Non-trivial expressions:
- $a + b$
- $a \times b$
- $a - 1$

Transfer function for each statement

<table>
<thead>
<tr>
<th>Statement, s</th>
<th>gen(s)</th>
<th>kill(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>${a+b}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>${a \times b}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>3</td>
<td>${a+b}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td>${a+b, a \times b, a - 1}$</td>
</tr>
<tr>
<td>5</td>
<td>${a+b}$</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Propagating Available Expressions

- Initially, no available expressions
- **Forward analysis**: Propagate available expressions in the direction of control flow
- For each statement $s$, outgoing available expressions are:
  \[ \text{incoming avail. exprs. minus } \text{kill}(s) \text{ plus } \text{gen}(s) \]
- When control flow splits, propagate available expressions both ways
- When control flows merge, intersect the incoming available expressions
Data flow equations

\( AE_{\text{entry}}(s) \) -- avail. expr. at entry of \( s \)
\( AE_{\text{exit}}(s) \) -- ... -- exit ... 

Solution of these equations

<table>
<thead>
<tr>
<th>( s )</th>
<th>( AE_{\text{entry}}(s) )</th>
<th>( AE_{\text{exit}}(s) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( \emptyset )</td>
<td>{ a+b }</td>
</tr>
<tr>
<td>2</td>
<td>{ a+b }</td>
<td>{ a+b, a \times b }</td>
</tr>
<tr>
<td>3</td>
<td>{ a+b }</td>
<td>{ a+b }</td>
</tr>
<tr>
<td>4</td>
<td>{ a+b }</td>
<td>\emptyset</td>
</tr>
<tr>
<td>5</td>
<td>\emptyset</td>
<td>{ a+b }</td>
</tr>
</tbody>
</table>

\[ AE_{\text{entry}}(1) = \emptyset \]
\[ AE_{\text{entry}}(2) = AE_{\text{exit}}(1) \]
\[ AE_{\text{entry}}(3) = AE_{\text{exit}}(2) \cap AE_{\text{exit}}(5) \]
\[ AE_{\text{entry}}(4) = AE_{\text{exit}}(3) \]
\[ AE_{\text{entry}}(5) = AE_{\text{exit}}(4) \]
\[ AE_{\text{exit}}(1) = AE_{\text{entry}}(1) \cup \{ a+b \} \]
\[ AE_{\text{exit}}(2) = AE_{\text{entry}}(2) \cup \{ a \times b \} \]
\[ AE_{\text{exit}}(3) = AE_{\text{entry}}(3) \cup \{ a+b \} \]
\[ AE_{\text{exit}}(4) = AE_{\text{entry}}(4) \setminus \{ a+b, a \times b, a+1 \} \]
\[ AE_{\text{exit}}(5) = AE_{\text{entry}}(5) \cup \{ a+b \} \]
var m = x - y;
if (random()) {
    while (m > 0) {
        x = y + 1;
    }
} else {
    n = x - y;
}
}
z = x - y;

Is \(x-y\) an available expression when entering this statement?
Outline

- First example: Available expressions
- Basic principles
- More examples
- Solving data flow problems
- Inter-procedural analysis
- Sensitivities
Defining a Data Flow Analysis

Any data flow analysis:
Defined by six properties

- Domain
- Direction
- Transfer function
- Meet operator
- Boundary condition
- Initial values
Domain

- Analysis associates some information with every program point
- “Information” means elements of a set
- Domain of the analysis: All possible elements the set may have
- E.g., for available expressions analysis: Domain is set of non-trivial expressions
Direction

- Analysis *propagates* information along the *control flow graph*

- **Forward analysis:** Normal *flow of control*

- **Backward analysis:** *Invert all edges*
  - Reasons about executions in reverse

- **E.g., available expression analysis:** Forward
Transfer Function

- Defines how a **statement affects the propagated information**

\[
DF_{exit}(s) = \text{some function of } DF_{entry}(s)
\]

- E.g., for available expression analysis:

\[
AE_{exit}(s) = (AE_{entry} \setminus \text{kill}(s)) \cup \text{gen}(s)
\]
Meet Operator

- What if two statements $s_1, s_2$ flow to a statement $s$?
  - Forward analysis: Execution branches merge
  - Backward analysis: Branching point

- Meet operator defines how to combine the incoming information
  - Union: $DF_{entry}(s) = DF_{exit}(s_1) \cup DF_{exit}(s_2)$
  - Intersection: $DF_{entry}(s) = DF_{exit}(s_1) \cap DF_{exit}(s_2)$
Meet Operator

- What if two statements $s_1, s_2$ flow to a statement $s$?
  - Forward analysis: Execution branches merge
  - Backward analysis: Branching point

- Meet operator defines how to combine the incoming information
  - Union: $DF_{entry}(s) = DF_{exit}(s_1) \cup DF_{exit}(s_2)$
  - Intersection: $DF_{entry}(s) = DF_{exit}(s_1) \cap DF_{exit}(s_2)$

E.g., available expressions analysis
Boundary Condition

■ What information to start with at the first CFG node?
  □ Forward analysis: First node is entry node
  □ Backward analysis: First node is exit node

■ Common choices
  □ Empty set
  □ Entire domain
Boundary Condition

■ What information to start with at the first CFG node?

□ Forward analysis: First node is entry node
□ Backward analysis: First node is exit node

■ Common choices

□ Empty set
□ Entire domain

E.g., available expressions analysis
Initial Values

- What is the **information to start with at intermediate nodes?**

- **Common choices**
  - Empty set
  - Entire domain
Initial Values

- What is the **information to start with at intermediate nodes**?

- Common choices
  - Empty set
  - Entire domain

E.g., available expressions analysis
Defining a Data Flow Analysis

Any data flow analysis: Defined by six properties

- Domain
- Direction
- Transfer function
- Meet operator
- Boundary condition
- Initial values
Defining a Data Flow Analysis

Any data flow analysis: Defined by six properties

- Domain
- Direction
- Transfer function
- Meet operator
- Boundary condition
- Initial values

- Non-trivial expressions
- Forward
- \( AE_{exit}(s) = (AE_{entry} \setminus \text{kill}(s)) \cup \text{gen}(s) \)
- Intersection (\( \cap \) )
- \( AE_{entry}(\text{entryNode}) = \emptyset \)
- \( \emptyset \)

Example: Available expressions
Outline

- First example: Available expressions
- Basic principles
- More examples
- Solving data flow problems
- Inter-procedural analysis
- Sensitivities
Reaching Definitions Analysis

Goal: For each program point, compute which assignments may have been made and may not have been overwritten

- Useful in various program analyses
- E.g., to compute a data flow graph
Example

```javascript
var x = 5;
var y = 1;
while (x > 1) {
    y = x * y;
    x = x - 1;
}
```
Example

```javascript
var x = 5;
var y = 1;
while (x > 1) {
    y = x * y;
    x = x - 1;
}
```

Definition reaches entry of this statement
Example

```javascript
var x = 5;
var y = 1;
while (x > 1) {
  y = x * y;
  x = x - 1;
}
```

All definitions reach the entry of this statement.
Example

```javascript
var x = 5;
var y = 1;
while (x > 1) {
    y = x * y;
    x = x - 1;
}
```

Three definitions reach entry of this statement
Defining the Analysis

- **Domain**: Definitions (i.e., assignments) in the code
- **Direction**: Forward
- **Meet operator**: Union

- Because we care about definitions that *may* reach a program point
Transfer Function

Defined via \textit{gen} and \textit{kill} functions (again)

- Returns set of \textbf{pairs} \((v, s)\) of variables and stmts
- \((v, s)\) means a definition of \(v\) at \(s\)

\textbf{Function} \textit{gen}(\(s\))

- If \(s\) is assignment to \(v\): \((v, s)\)
- Otherwise: Empty set

\textbf{Function} \textit{kill}(\(s\))

- If \(s\) is assignment to \(v\): \((v, s')\) for all \(s'\) that define \(v\)
- Otherwise: Empty set
Defining the Analysis (2)

- Special "statement" for undefined variables: ?

- Transfer function:
  \[ RD_{exit}(s) = (RD_{entry}(s) \setminus kill(s)) \cup gen(S) \]

- Boundary condition: Entry node starts with all variables undefined
  - \[ RD_{entry}(entryNode) = \{(v, ?) | v \in Vars\} \]

- Initially, all nodes have no reaching definitions
Example: Reaching Definitions

\[ \begin{aligned}
&\text{entry} \\
&\quad x = 5 \\
&\quad y = 1 \\
&\quad \text{if } x > 7 \\
&\quad \quad \text{then } y = x \times y \\
&\quad \quad \text{return } x + 1
\end{aligned} \]

\[ \begin{array}{c|c|c}
& \text{gen}(s) & \text{hiU}(s) \\
1 & \{ (x, 1) \} & \{ (x, 1), (x, 5), (x, \_? ) \} \\
2 & \{ (y, 2) \} & \{ (y, 2), (y, 4), (y, \_? ) \} \\
3 & \emptyset & \emptyset \\
4 & \{ (y, 4) \} & \{ (y, 2, (y, 4), (y, \_? ) \} \\
5 & \{ (x, 5) \} & \{ (x, 1), (x, 5), (x, \_? ) \} \\
\end{array} \]

\[ \Sigma = 16 \]
Data flow equation

\[ \text{RD}_{\text{entry}}(1) = \{x, y\} \]

\[ \text{RD}_{\text{exit}}(2) = \text{RD}_{\text{exit}}(1) \]

\[ \text{RD}_{\text{exit}}(3) = \text{RD}_{\text{exit}}(2) \cup \text{RD}_{\text{exit}}(5) \]

\[ \text{RD}_{\text{exit}}(4) = \text{RD}_{\text{exit}}(3) \]

\[ \text{RD}_{\text{exit}}(5) = \text{RD}_{\text{exit}}(4) \]

\[ \text{RD}_{\text{exit}}(1) = (\text{RD}_{\text{entry}}(1) \setminus \{x, y\}) \cup \{x, y\} \]

\[ \text{RD}_{\text{exit}}(2) = \ldots \quad \text{(similar)} \]

\[ \text{RD}_{\text{exit}}(3) = \text{RD}_{\text{entry}}(3) \]

\[ \text{RD}_{\text{exit}}(4) = \ldots \quad \text{(similar)} \]

\[ \text{RD}_{\text{exit}}(5) = \ldots \quad \text{(similar)} \]
Outline

- First example: Available expressions
- Basic principles
- More examples
- Solving data flow problems
- Inter-procedural analysis
- Sensitivities
Goal: For each program point, find expressions that must be very busy

- "Very busy": On all future paths, the expression will be used before any of the variables in it are redefined.

- Useful for program optimizations, e.g., hoisting.
  - Hoisting an expression: Pre-compute it, e.g., before entering a block, for later use.
Example

```java
if (a > b) {
    x = b - a;
    y = a - b;
} else {
    y = b - a;
    x = a - b;
}
```
Example

```java
if (a > b) {
    x = b - a;
    y = a - b;
} else {
    y = b - a;
    x = a - b;
}
```

*a - b* and *b - a* are very busy here
Defining the Analysis

- **Domain**: All non-trivial expressions occurring in the code
- **Direction**: Backward
- **Meet operator**: Intersection

  - Because we care about very busy expressions that *must* be used
Transfer Function

Defined via \textit{gen} and \textit{kill} functions (again)

- Returns set of expressions
- Backward analysis, i.e., returned expressions are very busy expressions at entry of statement

\textbf{Function} \textit{gen}(s)

- All expressions $e$ that appear in $s$

\textbf{Function} \textit{kill}(s)

- If $s$ assigns to $x$, all expressions in which $x$ occurs
- Otherwise: Empty set
Defining the Analysis (2)

- **Transfer function:**
  \[ V_{B_{entry}}(s) = (V_{B_{exit}}(s) \setminus kill(s)) \cup gen(S) \]

- **Boundary condition:** Final node starts with no very busy expressions
  - \[ V_{B_{exit}}(finalNode) = \emptyset \]

- **Initially, all nodes have no very busy expressions**
Example: Very busy expression analysis

\[
\begin{align*}
&\text{entry} \\
&\quad \downarrow \\
&\quad a > b \\
&\downarrow \\
&x = b - a \\
&\downarrow \\
y = b - a \\
&\downarrow \\
x = a - b \\
&\downarrow \\
&\text{exit}
\end{align*}
\]

\[
\begin{array}{c|c|c}
& \text{gen(s)} & \text{lin(s)} \\
\hline
1 & \emptyset & \emptyset \\
2 & \{b-a\} & \emptyset \\
3 & \{a-b\} & \emptyset \\
4 & \{b-a\} & \emptyset \\
5 & \{a-b\} & \emptyset \\
\end{array}
\]

\[
\begin{array}{c|c|c}
& \text{VBentry(s)} & \text{VBexit(s)} \\
\hline
1 & \{a-b, b-a\} & \{a-b, b-a\} \\
2 & \{a-b, b-a\} & \{a-b\} \\
3 & \{a-b\} & \emptyset \\
4 & \{a-b, b-a\} & \{a-b\} \\
5 & \{a-b\} & \emptyset \\
\end{array}
\]