Dataflow Analysis

Mayur Naik
CIS 573 – Fall 2019
What Is Dataflow Analysis?

- Static analysis reasoning about flow of data in program
- Different kinds of data: constants, variables, expressions
- Used by bug-finding tools and compilers
The WHILE Language

x = 5;
y = 1;
while (x != 1) {
    y = x * y;
x = x - 1
}

(statement) S ::= x = a | S1 ; S2 | if (b) { S1 } else { S2 } | while (b) { S1 }

(arithmetic expression) a ::= x | n | a1 * a2 | a1 - a2

(boolean expression) b ::= true | !b | b1 && b2 | a1 != a2

(integer variable) x

(integer constant) n
Control-Flow Graphs

```
x = 5;
y = 1;
while (x != 1) {
    y = x * y;
    x = x - 1
}
```

```
entry

x = 5

y = 1

(x != 1) ?

true

y = x * y

false

x = x - 1

exit
```
QUIZ: Control-Flow Graphs

- **entry**
- **x = 5**
- **x != 0**
  - **false**
    - **exit**
  - **true**
    - **y = x**
    - **x = x - 1**
    - **y != 0?**
      - **false**
      - **y = y - 1**
      - **true**

QUIZ: Control-Flow Graphs

```plaintext
x = 5;
while (x != 0) {
    y = x;
    x = x - 1;
    while (y != 0) {
        y = y - 1
    }
}
```
Soundness, Completeness, Termination

- Impossible for analysis to achieve all three together
- Dataflow analysis sacrifices completeness
- Sound: Will report all facts that could occur in actual runs
- Incomplete: May report additional facts that can’t occur in actual runs
Abstracting Control-Flow Conditions

- Abstracts away control-flow conditions with non-deterministic choice (*).
- Non-deterministic choice => assumes condition can evaluate to true or false.
- Considers all paths possible in actual runs (sound), and maybe paths that are never possible (incomplete).
Applications of Dataflow Analysis

Reaching Definitions Analysis
• Find usage of uninitialized variables

Available Expressions Analysis
• Avoid recomputing expressions

Very Busy Expressions Analysis
• Reduce code size

Live Variables Analysis
• Allocate registers efficiently
Reaching Definitions Analysis

Goal: Determine, for each program point, which assignments have been made and not overwritten, when execution reaches that point along some path.

- “Assignment” == “Definition”
QUIZ: Reaching Definitions Analysis

1. The assignment \( y = 1 \) reaches P1
2. The assignment \( y = 1 \) reaches P2
3. The assignment \( y = x \times y \) reaches P1
QUIZ: Reaching Definitions Analysis

1. The assignment $y = 1$ reaches P1

2. The assignment $y = 1$ reaches P2

3. The assignment $y = x \times y$ reaches P1
Result of Dataflow Analysis (Informally)

- Set of facts at each program point
- For reaching definitions analysis, fact is a pair of the form:
  - Examples: \(<x, 2>, <y, 5>\)
Result of Dataflow Analysis (Formally)

- Give distinct label $n$ to each node
- $\text{IN}[n] = \text{set of facts at entry of node } n$
- $\text{OUT}[n] = \text{set of facts at exit of node } n$
- Dataflow analysis computes $\text{IN}[n]$ and $\text{OUT}[n]$ for each node
- Repeat two operations until $\text{IN}[n]$ and $\text{OUT}[n]$ stop changing
  - Called “saturated” or “fixed point”
Reaching Definitions Analysis: Operation #1

\[ \text{IN}[n] = \bigcup_{n' \in \text{predecessors}(n)} \text{OUT}[n'] \]

\[ \text{IN}[n] = \text{OUT}[n1] \cup \text{OUT}[n2] \cup \text{OUT}[n3] \]
Reaching Definitions Analysis: Operation #2

\[
\text{OUT}[n] = (\text{IN}[n] - \text{KILL}[n]) \cup \text{GEN}[n]
\]

\text{OUT}[n]
\text{IN}[n]

\begin{align*}
n: & \quad \text{b ?} \quad \text{GEN}[n] = \emptyset \quad \text{KILL}[n] = \emptyset \\
n: & \quad x = a \quad \text{GEN}[n] = \{ <x, n> \} \\
& \quad \text{KILL}[n] = \{ <x, m> : m \neq n \} 
\end{align*}
Overall Algorithm: Chaotic Iteration

for (each node n):
    \( \text{IN}[n] = \text{OUT}[n] = \emptyset \)

\( \text{OUT}[\text{entry}] = \{ <v, ?> : v \text{ is a program variable} \} \)

repeat:
    for (each node n):
        \( \text{IN}[n] = \bigcup \text{OUT}[n'] \)  
        \( n' \in \text{predecessors}(n) \)

        \( \text{OUT}[n] = (\text{IN}[n] - \text{KILL}[n]) \cup \text{GEN}[n] \)

until \( \text{IN}[n] \) and \( \text{OUT}[n] \) stop changing for all n
# Reaching Definitions Analysis Example

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>{&lt;x,?&gt;,&lt;y,?&gt;}</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>7</td>
<td>∅</td>
<td>--</td>
</tr>
</tbody>
</table>

1: entry
2: \( x = y \)
3: \( y = 1 \)
4: \( (x \neq 1) \) ?
5: \( y = x \cdot y \)
6: \( x = x - 1 \)
7: exit

true
false
# Reaching Definitions Analysis Example

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>{&lt;x,?&gt;,&lt;y,?&gt;}</td>
</tr>
<tr>
<td>2</td>
<td>{&lt;x,?&gt;,&lt;y,?&gt;}</td>
<td>{&lt;x,2&gt;,&lt;y,?&gt;}</td>
</tr>
<tr>
<td>3</td>
<td>{&lt;x,2&gt;,&lt;y,?&gt;}</td>
<td>{&lt;x,2&gt;,&lt;y,3&gt;}</td>
</tr>
<tr>
<td>4</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>7</td>
<td>∅</td>
<td>--</td>
</tr>
</tbody>
</table>

1: entry
2: x = y
3: y = 1
4: (x != 1) ?
5: y = x * y
6: x = x - 1
7: exit
# QUIZ: Reaching Definitions Analysis

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>{&lt;x,?&gt;,&lt;y,?&gt;}</td>
</tr>
<tr>
<td>2</td>
<td>{&lt;x,?&gt;,&lt;y,?&gt;}</td>
<td>{&lt;x,2&gt;,&lt;y,?&gt;}</td>
</tr>
<tr>
<td>3</td>
<td>{&lt;x,2&gt;,&lt;y,?&gt;}</td>
<td>{&lt;x,2&gt;,&lt;y,3&gt;}</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>--</td>
</tr>
</tbody>
</table>

```plaintext
x = y
y = 1
(x != 1) ?
y = x * y
x = x - 1
```

Flowchart:
- **Entry**
- **x = y**
- **y = 1**
- **(x != 1)?**
  - **true**
  - **y = x * y**
  - **false**
    - **x = x - 1**
    - **exit**
QUIZ: Reaching Definitions Analysis

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>{&lt;x,?&gt;,&lt;y,?&gt;}</td>
</tr>
<tr>
<td>2</td>
<td>{&lt;x,?&gt;,&lt;y,?&gt;}</td>
<td>{&lt;x,2&gt;,&lt;y,?&gt;}</td>
</tr>
<tr>
<td>3</td>
<td>{&lt;x,2&gt;,&lt;y,?&gt;}</td>
<td>{&lt;x,2&gt;,&lt;y,3&gt;}</td>
</tr>
<tr>
<td>4</td>
<td>{&lt;x,2&gt;,&lt;y,3&gt;,&lt;y,5&gt;,&lt;x,6&gt;}</td>
<td>{&lt;x,2&gt;,&lt;y,3&gt;,&lt;y,5&gt;,&lt;x,6&gt;}</td>
</tr>
<tr>
<td>5</td>
<td>{&lt;x,2&gt;,&lt;y,3&gt;,&lt;y,5&gt;,&lt;x,6&gt;}</td>
<td>{&lt;x,2&gt;,&lt;y,5&gt;,&lt;x,6&gt;}</td>
</tr>
<tr>
<td>6</td>
<td>{&lt;x,2&gt;,&lt;y,5&gt;,&lt;x,6&gt;}</td>
<td>{&lt;y,5&gt;,&lt;x,6&gt;}</td>
</tr>
<tr>
<td>7</td>
<td>{&lt;x,2&gt;,&lt;y,3&gt;,&lt;y,5&gt;,&lt;x,6&gt;}</td>
<td>--</td>
</tr>
</tbody>
</table>

1: entry
2: \( x = y \)
3: \( y = 1 \)
4: \( (x \neq 1) \) ?
5: \( y = x \times y \)
6: \( x = x - 1 \)
7: exit

1: \( x = y \)
2: \( y = 1 \)
3: \( (x \neq 1) \) ?
4: \( y = x \times y \)
5: \( x = x - 1 \)
6: \( x = y \)
7: exit
Does It Always Terminate?

Chaotic Iteration algorithm always terminates

- The two operations of reaching definitions analysis are monotonic
  => IN and OUT sets never shrink, only grow
- Largest they can be is set of all definitions in program, which is finite
  => IN and OUT cannot grow forever

=> IN and OUT will stop changing after some iteration
Goal: Determine very busy expressions at the exit from the point.

An expression is **very busy** if, no matter what path is taken, the expression is used before any of the variables occurring in it are redefined.
Very Busy Expressions Analysis:

Operation #1

\[ \text{OUT}[n] = \bigcap_{n' \in \text{successors}(n)} \text{IN}[n'] \]

\[ \text{OUT}[n] = \text{IN}[n1] \cap \text{IN}[n2] \cap \text{IN}[n3] \]
Very Busy Expressions Analysis: Operation #2

\[ \text{IN}[n] = (\text{OUT}[n] - \text{KILL}[n]) \cup \text{GEN}[n] \]

n: \text{b ?} \quad \text{GEN}[n] = \emptyset \quad \text{KILL}[n] = \emptyset

n: \text{x = a} \quad \text{GEN}[n] = \{ a \} \quad \text{KILL}[n] = \{ \text{expr e : e contains x} \}
Overall Algorithm: Chaotic Iteration

for (each node n)
    IN[n] = OUT[n] = set of all exprs in program
IN[exit] = ∅
repeat:
    for (each node n)
        \( \text{OUT}[n] = \bigcap_{n' \in \text{successors}(n)} \text{IN}[n'] \)
        \( \text{IN}[n] = (\text{OUT}[n] - \text{KILL}[n]) \cup \text{GEN}[n] \)
until IN[n] and OUT[n] stop changing for all n
**Very Busy Expressions Analysis Example**

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>{ b-a, a-b }</td>
</tr>
<tr>
<td>2</td>
<td>{ b-a, a-b }</td>
<td>{ b-a, a-b }</td>
</tr>
<tr>
<td>3</td>
<td>{ b-a, a-b }</td>
<td>{ b-a, a-b }</td>
</tr>
<tr>
<td>4</td>
<td>{ b-a, a-b }</td>
<td>{ b-a, a-b }</td>
</tr>
<tr>
<td>5</td>
<td>{ b-a, a-b }</td>
<td>{ b-a, a-b }</td>
</tr>
<tr>
<td>6</td>
<td>{ b-a, a-b }</td>
<td>{ b-a, a-b }</td>
</tr>
<tr>
<td>7</td>
<td>{ b-a, a-b }</td>
<td>{ b-a, a-b }</td>
</tr>
<tr>
<td>8</td>
<td>\ø</td>
<td>--</td>
</tr>
</tbody>
</table>

Diagram:

1: `entry`
2: `(a!=b)` ?
3: `x=b-a`
4: `y=b-a`
5: `a=0`
6: `y=a-b`
7: `x=a-b`
8: `exit`
Very Busy Expressions Analysis Example

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>{ \text{b-a, a-b} }</td>
</tr>
<tr>
<td>2</td>
<td>{ \text{b-a, a-b} }</td>
<td>{ \text{b-a, a-b} }</td>
</tr>
<tr>
<td>3</td>
<td>{ \text{b-a, a-b} }</td>
<td>{ \text{b-a, a-b} }</td>
</tr>
<tr>
<td>4</td>
<td>{ \text{b-a, a-b} }</td>
<td>{ \text{b-a, a-b} }</td>
</tr>
<tr>
<td>5</td>
<td>{ \text{b-a, a-b} }</td>
<td>{ \text{b-a, a-b} }</td>
</tr>
<tr>
<td>6</td>
<td>{ \text{a-b} }</td>
<td>\emptyset</td>
</tr>
<tr>
<td>7</td>
<td>{ \text{a-b} }</td>
<td>\emptyset</td>
</tr>
<tr>
<td>8</td>
<td>\emptyset</td>
<td>--</td>
</tr>
</tbody>
</table>

![Diagram](image)
QUIZ: Very Busy Expressions Analysis

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
<td>{ a-b }</td>
</tr>
<tr>
<td>6</td>
<td>{ a-b }</td>
<td>∅</td>
</tr>
<tr>
<td>7</td>
<td>{ a-b }</td>
<td>∅</td>
</tr>
<tr>
<td>8</td>
<td>∅</td>
<td>--</td>
</tr>
</tbody>
</table>

```
1: entry
2: (a!=b) ?
1. true 3: x=b-a
2. false 4: y=b-a
3.      5: a=0
4.      7: x=a-b
5.      6: y=a-b
7.      8: exit
8.      
```
QUIZ: Very Busy Expressions Analysis

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>{ b-a }</td>
</tr>
<tr>
<td>2</td>
<td>{ b-a }</td>
<td>{ b-a }</td>
</tr>
<tr>
<td>3</td>
<td>{ b-a, a-b }</td>
<td>{ a-b }</td>
</tr>
<tr>
<td>4</td>
<td>{ b-a }</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
<td>{ a-b }</td>
</tr>
<tr>
<td>6</td>
<td>{ a-b }</td>
<td>∅</td>
</tr>
<tr>
<td>7</td>
<td>{ a-b }</td>
<td>∅</td>
</tr>
<tr>
<td>8</td>
<td>∅</td>
<td>--</td>
</tr>
</tbody>
</table>

1: entry
2: (a!=b) ?
3: x=b-a
4: y=b-a
5: a=0
6: y=a-b
7: x=a-b
8: exit

x=b
y=a
x=a

true
false

y=b-a
a=0
x=a-b
exit

true
false
Goal: Determine, for each program point, which expressions must already have been computed, and not later modified, on all paths to the program point.
# Available Expressions Analysis

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>2</td>
<td>{a-b, a*b, a-1}</td>
<td>{a-b, a*b, a-1}</td>
</tr>
<tr>
<td>3</td>
<td>{a-b, a*b, a-1}</td>
<td>{a-b, a*b, a-1}</td>
</tr>
<tr>
<td>4</td>
<td>{a-b, a*b, a-1}</td>
<td>{a-b, a*b, a-1}</td>
</tr>
<tr>
<td>5</td>
<td>{a-b, a*b, a-1}</td>
<td>{a-b, a*b, a-1}</td>
</tr>
<tr>
<td>6</td>
<td>{a-b, a*b, a-1}</td>
<td>{a-b, a*b, a-1}</td>
</tr>
<tr>
<td>7</td>
<td>{a-b, a*b, a-1}</td>
<td>--</td>
</tr>
</tbody>
</table>

1: **entry**
2: $x = a-b$
3: $y = a*b$
4: $(y \neq a-b)?$ 
   - true
   - false
5: $a = a-1$
6: $x = a-b$
7: **exit**
Available Expressions Analysis

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
<td>{a-b}</td>
</tr>
<tr>
<td>3</td>
<td>{a-b}</td>
<td>{a-b, a*b}</td>
</tr>
<tr>
<td>4</td>
<td>{a-b, a*b}</td>
<td>{a-b, a*b}</td>
</tr>
<tr>
<td>5</td>
<td>{a-b, a*b}</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>{a-b}</td>
</tr>
<tr>
<td>7</td>
<td>{a-b, a*b}</td>
<td>--</td>
</tr>
</tbody>
</table>

1: entry
2: \( x = a-b \)
3: \( y = a*b \)
4: (\( y \neq a-b \))?
5: \( a = a-1 \)
6: \( x = a-b \)
7: exit
Available Expressions Analysis

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
<td>{a-b}</td>
</tr>
<tr>
<td>3</td>
<td>{a-b}</td>
<td>{a-b, a*b}</td>
</tr>
<tr>
<td>4</td>
<td>{a-b}</td>
<td>{a-b}</td>
</tr>
<tr>
<td>5</td>
<td>{a-b}</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>{a-b}</td>
</tr>
<tr>
<td>7</td>
<td>{a-b}</td>
<td>--</td>
</tr>
</tbody>
</table>

1: entry
2: \( x = a-b \)
3: \( y = a*b \)
4: \( (y \neq a-b) \)?
5: \( a = a-1 \)
6: \( x = a-b \)
7: exit

Flow diagram:
- Start at entry
- \( x = a-b \) (2)
- \( y = a*b \) (3)
- \( y = a-b \) or \( y = a-b \) (4)
- \( a = a-1 \) (5)
- \( x = a-b \) (6)
- Exit (7)
**Goal:** Determine for each program point which variables could be *live* at the point’s exit.

A variable is *live* if there is a path to a use of the variable that doesn’t redefine the variable.
## Live Variables Analysis

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>3</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>4</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>5</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>6</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>7</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>8</td>
<td>∅</td>
<td>--</td>
</tr>
</tbody>
</table>

```
x = 2
y = 4
(y != x) ?
exit
```

```
x = z
z = y*y
```

```
entry
```

```
y = 4
```

```
x = 2
```

```
(y != x) ?
true
false
```

```
z = y
```

```
z = y*y
```

```
x = z
```

```
exit
```
Live Variables Analysis

<table>
<thead>
<tr>
<th>n</th>
<th>IN[n]</th>
<th>OUT[n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>--</td>
<td>∅</td>
</tr>
<tr>
<td>2</td>
<td>∅</td>
<td>{y}</td>
</tr>
<tr>
<td>3</td>
<td>{y}</td>
<td>{x, y}</td>
</tr>
<tr>
<td>4</td>
<td>{x, y}</td>
<td>{y}</td>
</tr>
<tr>
<td>5</td>
<td>{y}</td>
<td>{z}</td>
</tr>
<tr>
<td>6</td>
<td>{y}</td>
<td>{z}</td>
</tr>
<tr>
<td>7</td>
<td>{z}</td>
<td>∅</td>
</tr>
<tr>
<td>8</td>
<td>∅</td>
<td>--</td>
</tr>
</tbody>
</table>
Overall Pattern of Dataflow Analysis

\[ [n] = ( [n] - \text{KILL}[n]) \cup \text{GEN}[n] \]

\[ [n] = \text{IN or OUT} \]

\[ n' \subseteq \text{predecessors or successors} \]

\[ = \cup \text{(may)} \lor \cap \text{(must)} \]
Reaching Definitions Analysis

\[ \text{OUT}[n] = (\text{IN}[n] - \text{KILL}[n]) \cup \text{GEN}[n] \]

\[ \text{IN}[n] = U \text{OUT}[n'] \]

\[ n' \subseteq \text{preds}(n) \]

= IN or OUT

= U (may) or \cap (must)

= predecessors or successors
\[ \text{IN}[n] = \left( \text{OUT}[n] - \text{KILL}[n] \right) \cup \text{GEN}[n] \]

\[ \text{OUT}[n] = \bigcap \text{IN}[n'] \]

\[ n' \subseteq \text{succs}(n) \]

\[ = \text{IN or OUT} \]

\[ = \cup \text{(may) or } \cap \text{ (must)} \]

\[ = \text{predecessors or successors} \]
QUIZ: Available Expressions Analysis

\[ n = (n - \text{KILL}[n]) \cup \text{GEN}[n] \]

\[ n = [n'] \]

\( n' \subseteq \) (n)

\( n' = \text{IN or OUT} \)

\( = \text{U (may) or } \cap \text{ (must)} \)

\( = \text{predecessors or successors} \)
QUIZ: Available Expressions Analysis

\[
\text{OUT}[n] = (\text{IN}[n] - \text{KILL}[n]) \cup \text{GEN}[n]
\]

\[
\text{IN}[n] = \cap \text{OUT}[n']
\]

\[
n' \subseteq \text{preds}(n)
\]

\[
= \text{IN or OUT}
\]

\[
= \cup (\text{may}) \text{ or } \cap (\text{must})
\]

\[
= \text{predecessors or successors}
\]
QUIZ: Live Variables Analysis

\[
\begin{align*}
\text{[n]} &= (\text{[n]} - \text{KILL[n]}) \cup \text{GEN[n]} \\
\text{[n']} &= \text{[n']} \\
n' &\subseteq (n)
\end{align*}
\]

= IN or OUT

= U (may) or \cap (must)

= predecessors or successors
 QUIZ: Live Variables Analysis

$\text{IN}[n] = (\text{OUT}[n] - \text{KILL}[n]) \cup \text{GEN}[n]$ 

$\text{OUT}[n] = \text{IN}[n'] \cup \text{succs}(n)$ 

$n' \in \text{succs}(n)$ 

= IN or OUT  = U (may) or \cap (must)  = predecessors or successors
### QUIZ: Classifying Dataflow Analyses

Match each analysis with its characteristics.

<table>
<thead>
<tr>
<th></th>
<th>May</th>
<th>Must</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Backward</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Very Busy Expressions**
- **Reaching Definitions**
- **Live Variables**
- **Available Expressions**
**QUIZ: Classifying Dataflow Analyses**

Match each analysis with its characteristics.

<table>
<thead>
<tr>
<th></th>
<th>May</th>
<th>Must</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward</td>
<td>Reaching Definitions</td>
<td>Available Expressions</td>
</tr>
<tr>
<td>Backward</td>
<td>Live Variables</td>
<td>Very Busy Expressions</td>
</tr>
</tbody>
</table>
What Have We Learned?

- What is dataflow analysis
- Reasoning about flow of data using control-flow graphs
- Specifying dataflow analyses using local rules
- Chaotic iteration algorithm to compute global properties
- Four classical dataflow analyses
- Classification: forward vs. backward, may vs. must