Program Analysis
Data Flow Analysis (Part 1)
Big Picture

- Static versus dynamic analysis
- Many ways of formulating and implementing analyses
- This lecture: One popular way of formulating a static analysis: Data flow analysis
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
First example: Available expressions
Basic principles
More examples
Solving data flow problems
Inter-procedural analysis
Sensitivities
Available Expression Analysis

Goal: For each program point, compute which expressions must have already been computed, and not later modified

- Useful, e.g., to avoid re-computing an expression
- Used as part of compiler optimizations
Example

```javascript
var x = a + b;
var y = a * b;
while (y > a + b) {
    a = a - 1;
    x = a + b;
}
```
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
Transfer Functions

- **Transfer function of a statement:**
  How the statement affects the analysis state
  - Here: Analysis state = available expressions

- **Two functions**
  - **gen:** Available expressions generated by a statement
  - **kill:** Available expressions killed by a statement
**gen Function**

**Function** $gen : Stmt \rightarrow \mathcal{P}(Expr)$

- A statement generates an available expression $e$ if
  - it evaluates $e$ and
  - it does not later write any variable used in $e$

- Otherwise, function returns empty set

**Example:**

```plaintext
var x = a * b; generates a * b
```
**kill Function**

Function \( \text{kill} : \text{Stmt} \rightarrow \mathcal{P}(\text{Expr}) \)

- A statement **kills an available expressions** \( e \) if
  - it modifies any of the variables used in \( e \)
- Otherwise, function returns **empty set**

**Example:**

\( a = 23; \) **kills** \( a \times b \)
Example

```javascript
var x = a + b;
var y = a * b;
while (y > a + b) {
    a = a - 1;
    x = a + b;
}
```
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>( \text{hiU}(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:
  
  incoming avail. exprs. minus $\text{kill}(s)$ 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 entry (s) = avail. expr. at entry of s
AE exit (s) = avail. expr. at exit of s

<table>
<thead>
<tr>
<th>i</th>
<th>AE entry (i)</th>
<th>AE exit (i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ø</td>
<td>{a+b}</td>
</tr>
<tr>
<td>2</td>
<td>{a+b}</td>
<td>{a+b, a×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>Ø</td>
</tr>
<tr>
<td>5</td>
<td>Ø</td>
<td>{a+b}</td>
</tr>
</tbody>
</table>

Solution of these equations

see example
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?
Quiz

```
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?

No, because modifying \( x \) kills \( x - y \)

\[ x - y \]