Dataflow Analysis
- some quantity, e.g. "the set of live variables", domain is the set of all possible quantities
- $\text{IN}[s]$ and $\text{OUT}[s]$ is the quantity before and after each operation $s$
- direction, e.g. backwards
- transfer functions $f_s(x)$, e.g. $\text{IN}[s] = f_s(\text{OUT}[s])$ for backwards analysis
  $\text{OUT}[s] = f_s(\text{IN}[s])$ for forwards analysis
- backwards:
  - $\text{IN}[s] = f_s(\text{OUT}[s])$
  - $\text{OUT}[s] = \text{meet}(\text{IN}[s'])$ for all successors $s'$ of $s$
- forwards:
  - $\text{OUT}[s] = f_s(\text{IN}[s])$
  - $\text{IN}[s] = \text{meet}(\text{OUT}[s'])$ for all predecessors $s'$ of $s$
Live variables analysis
- a value in the domain is a set of variables
- backwards analysis
- \( f_s(x) = \text{gen}(s) \cup (x \setminus \text{kill}(s)) \) // \( \text{IN}[s] = f_s(\text{OUT}[s]) \)
  - \( \text{gen}(s) \) = any variables used by \( s \), e.g. \( s = "x = y * z" \), then \( \text{gen}(s) = \{ y, z \} \)
  - \( \text{kill}(s) \) = any variables assigned to in \( s \), e.g. \( \{ x \} \)
- meet is union // \( \text{OUT}[s] = \text{union of IN}[s'] \) for all successors \( s' \)

```plaintext
x = 0
if (condition) {
    x = 3
} else {
    print(x)
}
```
Reaching definitions
- a value in our domain is a set of assignment operations/nodes in the CFG
- forwards analysis
  \[ f_s(x) = \text{gen}(s) \cup (x \setminus \text{kill}(s)) \] // \( \text{OUT}[s] = f_s(\text{IN}[s]) \)
  - \( \text{gen}(s) = \{ s \} \) iff \( s \) is an assignment
  - \( \text{kill}(s) = \{ \text{all other assignments in the CFG with the same target as } s \} \)
- meet is union // \( \text{IN}[s] = \cup \text{OUT}[s'] \) for all predecessors \( s' \)

1. \( i = m - 1 \)
2. \( j = n \)
3. \( a = u_1 \)
loop {
  4. \( i = i + 1 \)
  5. \( j = j - 1 \)
  if (condition) {
    6. \( a = u_2 \)
  }
  7. \( i = u_3 \)
}
Available expressions
- a value in our domain is a set of expressions
- forwards
  - \( f_s(x) = \text{gen}(s) \cup (x \setminus \text{kill}(s)) \)
    - \( \text{gen}(s) = \{ \text{the expression of } s \} \)
    - \( \text{kill}(s) = \{ \text{all expressions involving the assignment target of } s \} \)
- meet is intersection \( \cap \) \( \text{IN}[s] = \cap \text{OUT}[s'] \) for all predecessors \( s' \)
- initialize \( \text{OUT}[\text{entry}] = \{ \} \)
- initialize \( \text{OUT}[\text{all other nodes}] = \{ \text{all expressions} \} \)

```
print(x + 1)
while (...) {
    // some code
}
print(x + 1)
```
Busy expressions
- a value in our domain is a set of expressions
- backwards analysis // IN[s] = f_s(OUT[s])
  - f_s(x) = gen(s) union (x \ kill(s))
    - gen and kill same as available expressions
- meet is intersection // OUT[s] = intersection IN[s'] for all successors s'
- initialize IN[exit] = {}
- initialize IN[everything else] = { all expressions }

<table>
<thead>
<tr>
<th>Forwards</th>
<th>Backwards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union</td>
<td>Reaching definitions</td>
</tr>
<tr>
<td>Intersection</td>
<td>Available expressions</td>
</tr>
<tr>
<td></td>
<td>Busy expressions</td>
</tr>
</tbody>
</table>