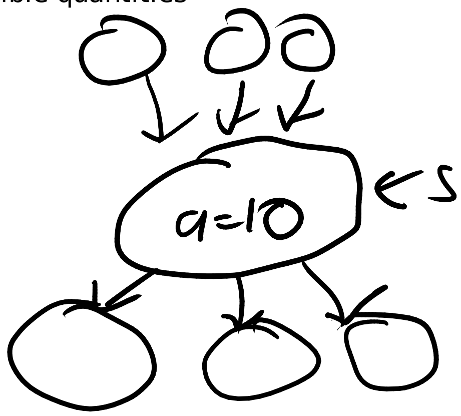
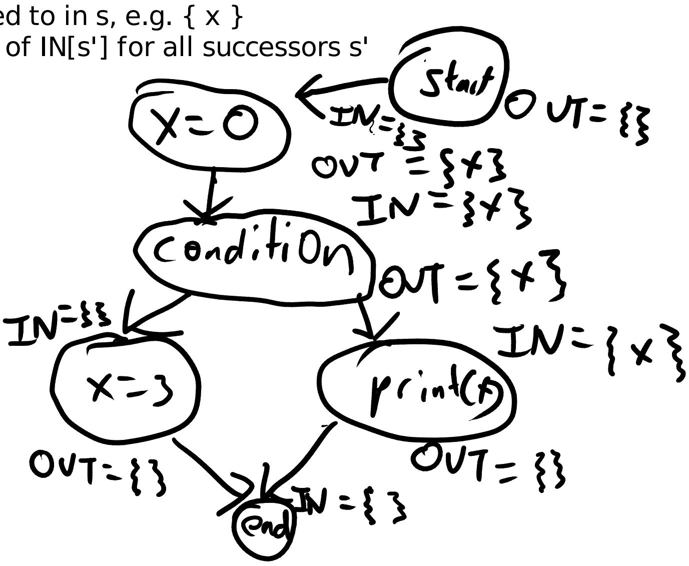
Dataflow Analysis

- some quantity, e.g. "the set of live variables", domain is the set of all possible quantities
- IN[s] and OUT[s] is the quantity before and after each operation s
- direction, e.g. backwards
- transfer functions f_s(x), e.g. IN[s] = f_s(OUT[s]) for backwards analysis
  OUT[s] = f_s(IN[s]) for forwards analysis
- backwards:
  - IN[s] = f_s(OUT[s])
  - OUT[s] = meet(IN[s']) for all successors s' of s
- forwards:
  - OUT[s] = f_s(IN[s])
  - IN[s] = meet(OUT[s']) for all predecessors s' of s

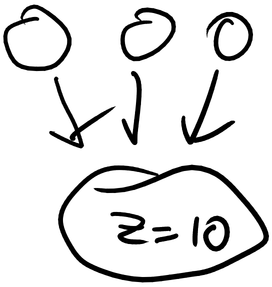


Live variables analysis
- a value in the domain is a set of variables
- backwards analysis
- f_s(x) = gen(s) union (x \ kill(s)) // IN[s] = f_s(OUT[s])
  - gen(s) = any variables used by s, e.g. s is "x = y * z", then gen(s) = { y, z }
  - kill(s) = any variables assigned to in s, e.g. { x }
- meet is union // OUT[s] = union of IN[s'] for all successors s'

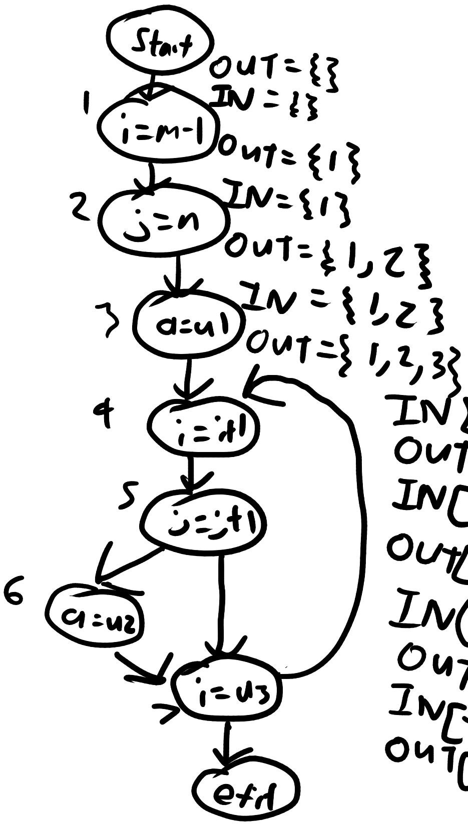
```
x = 0
if (condition) {
    x = 3
} else {
    print(x)
}
```



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



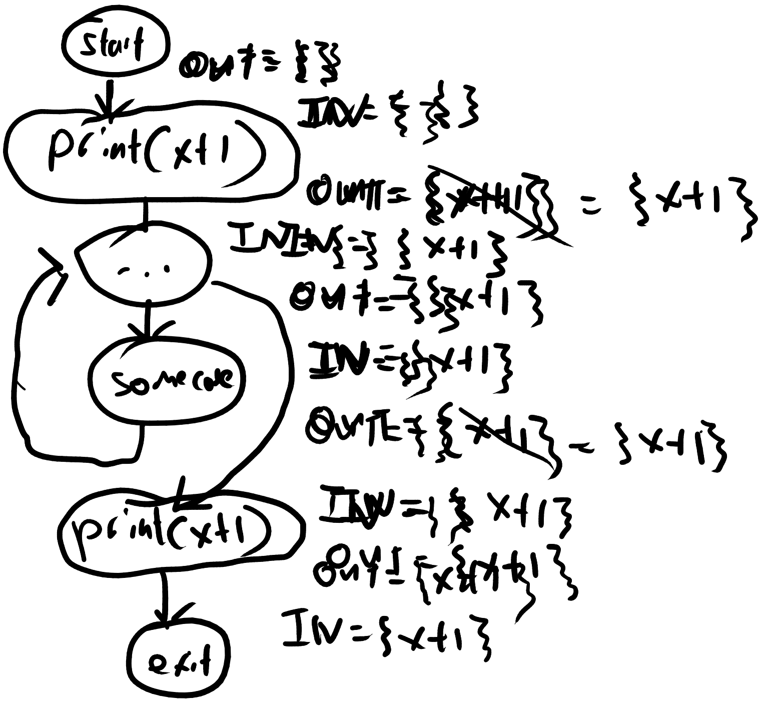
```
1. i = m - 1
2. j = n
3. a = u1
loop {
4.    i = i + 1
5.    j = j - 1
if (condition) {
6.        a = u2
}
7.    i = u3
if (condition) {
        break;
}
}
```



OUT={}
IN={}
OUT={1}
IN={1}
OUT={1,2}
IN={1,2}
OUT={1,2,3}
IN[4] = {1,2,3}
OUT[4] = {4,2,3}
IN[5] = {4,2,3}
OUT[5] = {4,5,3}
IN[6] = {4,5,3}
OUT[6] = {4,5,6}
IN[7] = {4,5,6,3}
OUT[7] = {3,5,6,7}

Available expressions
- a value in our domain is a set of expressions
- forwards
- f_s(x) = gen(s) union (x \ kill(s) // OUT[s] = f_s(IN[s])
    - gen(s) = { the expression of s }
    - kill(s) = { all expressions involving the assignment target of s }
- meet is intersection // IN[s] = intersection OUT[s'] for all predcessors s'
- initialize OUT[entry] = {}
- initialize OUT[all other nodes] = { 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 }

	Forwards	Backwards
Union	Reaching definitions	Live variables
Intersection	Available expressions	Busy expressions