Intro to Dataflow Analysis

Static type checking
+ prevents run-time bugs
+ leads to optimized code

Essence: use types to approximate run-time behaviors

There are properties that cannot be (easily) captured/approximated by a type system

Example: Definite assignment

JLS: local vars must be written before being read

```
public static int f(List<Integer> l){
  int x;
  try {
    x = l.get(0);
  } catch(IndexOutOfBoundsException e) {
    System.err.println("list is empty");
  }
  return x;
}
```

Program analyses

Live variables analysis

Reachable statements analysis (A4)

Array bounds analysis

Null pointer analysis

Escape analysis

Exception analysis

Termination analysis

and many more...

Live variable analysis

useful for reg allocation

var is live := its value is read later

```
b = a+3;   {a, e}
c = b*b;   {b, e}
d = c+e;   {c, e}
return d;  {d}
           live vars = ∅
```

2 vars live at any point ⇓ 2 regs

a, b, c, d never live together ⇓ assign ecx to a, b, c, d

```
int x = y;   is x alive?
f();
return x;
```

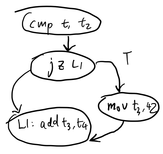
Undecidable !

Control flow graphs

We will design dataflow analyses that compute facts (e.g., live vars) over a control-flow graph.

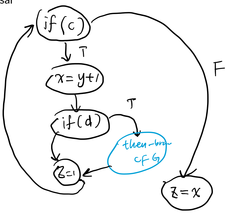
Can build CFG for IR/assembly programs

```
    cmp t1, t2
    jz L1
    mov t3, 42
L1: add t3, t4
```



Can build CFG for source programs, as an AST traversal

```
while (c) {
  x = y+1
  if(d) { x = y+z; }
  z=1
}
z = x
```



LVA on a simplified, IR-like language

(not exactly the IR we are going to use; more proper intro later)

	use[n]	def[n]
x ← e	vars(e)	{x}
[e1] ← e2	vars(e1) ∪ vars(e2)	∅
if e	vars(e)	∅
start	∅	∅
return e	vars(e)	∅

Use/def

use[n] := set of vars read by n

def[n] := ——— written ———

MOVE(x, e)

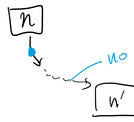
MOVE(MEM(e1), e2)

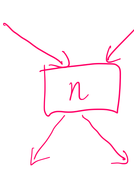
CJump(e, L1, L2)

LABEL(f)

RETURN(e)

use[a ← b+c] = {b, c}
def[a ← b+c] = {a}
use[[a] ← b+c] = {a, b, c}
def[[a] ← b+c] = ∅





$$in[n] = use[n] \cup (out[n] \setminus def[n])$$

out[n] = facts true on all out-edges

= vars may be live after n executes

$$out[n] = \bigcup_{n \lessdot n'} in[n']$$