Week 3.1

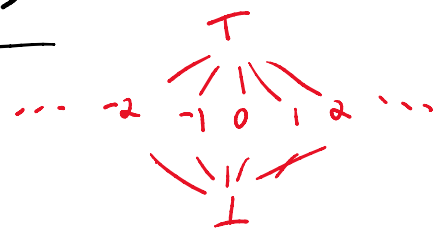
Monday, January 22, 2024 11:09 PM

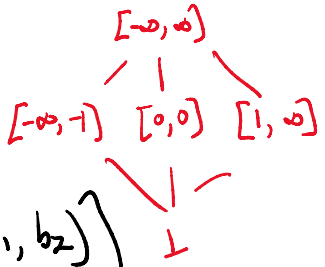
interval analysis

abstract domain:

- $\bot$
- $[a,b]$ s.t. $a \in \mathbb{Z} \cup \{-\infty\}$, $b \in \mathbb{Z} \cup \{\infty\}$, $a \leq b$

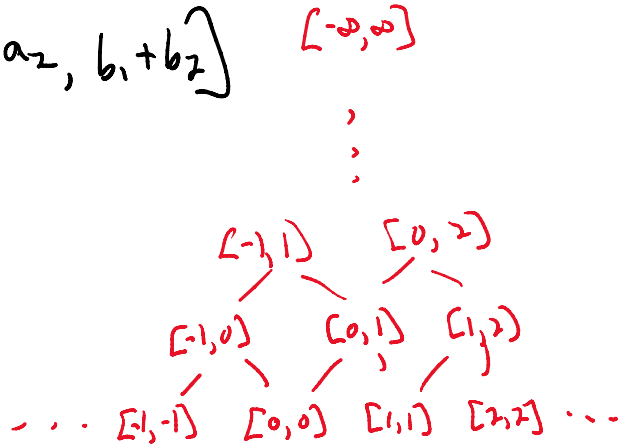
($\top = [-\infty, \infty]$)





$$[a_1, b_1] \sqcup [a_2, b_2] = [\min(a_1, a_2), \max(b_1, b_2)]$$

$$[a_1, b_1] + [a_2, b_2] = [a_1 + a_2, b_1 + b_2]$$



WIDENING

- widening operator ∇
  - join $\sqcup$
- there are many possible widening operators for an abstract domain
- one example widening operator for the interval domain
  - $\bot \nabla X = X \nabla \bot = X$

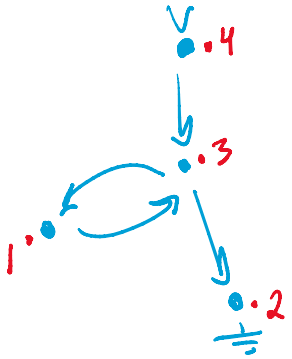
- $\bot \nabla X = X \nabla \bot = X$
- $[a_1, b_1] \nabla [a_2, b_2] = [a_3, b_3]$ s.t.
  - $L\ a_3 = a_1$ if $a_1 \leq a_2$, else $-\infty$
  - $L\ b_3 = b_1$ if $b_1 \geq b_2$, else ∞

- to keep precision, only apply ∇ at loop headers, otherwise use $\sqcup$

---

to detect loop headers in CFG:

- L post-order labeling starting from "entry"



if child node has already been visited but not yet labeled, it is a loop header

---

- first-order DFA: abstracts values
- second-order DFA: abstracts control-flow (execution paths)
  - L taint analysis asks "how did this string

- taint analysis asks "how did this string get to this system call?"

we abstract traces just we abstract values

---

reaching definitions

problem stmt: for each "use" of a variable what "def"s of that variable may "reach" this program point?

- "use": require variable's value
- "def": definition, ie, assignment
- "reach": there is some path from def to use that does not definitely redefine that variable

DFA: abstract domain + abstract semantics + abstract execution

elements of abstract domain: sets of program points representing defs

$\bot = \{\}$

$\top$ = set of all program points

$\sqcup$ = set $\cup$

abstract domain = $\mathcal{P}(PP)$ where PP is the set of program points

ordered by set inclusion

abstract store

map variables to sets of program points

↳ for each type, create a fake variable that stands for all heap objects of that type

for type τ let reachable_types(τ) be the set of types "reachable" via ptr dereference and/or struct field accesses from τ, excluding struct & function types

example:

```
struct foo {
    f1: &bar
    f2: &int
}

struct bar {
    f3: &(int) -> int
    f4: &&int
}

glob: &foo
```

↳ reachable_types(&foo) =

{ &bar, &int, &(int)->int, &&int, int }

{ &bar, &int, &(int)→int, &&int, int }

let PTRS be all pointer-typed globals, parameters, & locals of the function being analyzed.

- for $T \in$ reachable_types(PTRS), create a fake variable representing heap objects of that type
- put all fake variables in 'addr_taken'