Week 7.1

Tuesday, February 20, 2024 4:56 PM

- covered basics of DFA
  - abstract domain + abstract semantics + abstract execution (MOP) (MFP)
  - flow-sensitive
  - interprocedural
  - no ptr info
- math behind DFA
  - poethorian lattices
  - monotone functions
  - fixpoint theorems
- set constraint-based analysis
  - inclusion constraints
  - flow-insensitive
  - constraint gen. & constraint solving
- andersen-style pointer analysis

- go back to DFA, but w/ ptr info
  - reaching defs
  - control analysis

  } program slicing
- extend DFA w/ inter procedural analysis
  - taint analysis
  - context-sensitivity

- ...

L context-sensitivity

- if time:
  - SSA & sparse analysis
  - abstract interpretation
  - ;

---

program slicing

problem: given a specific instruction I, what set of instructions can influence the execution I

L backwards slice

L also "forward slice" (set of instructions that I can influence)

- built on
  - reaching defs (w/ ptr info)
  - control analysis

} program dependence graph (PDG)

↕

slicing

---

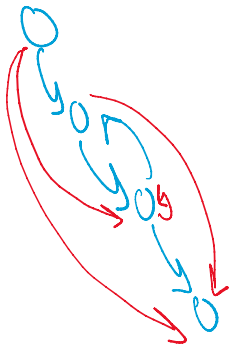
reaching defs w/ ptr info

reaching defs w/ ptr info

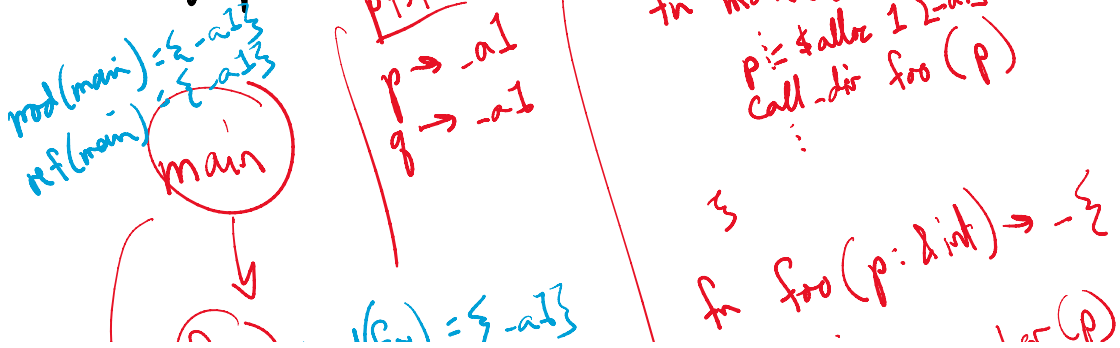
assume: flow-insensitive, interprocedural pointer analysis soln

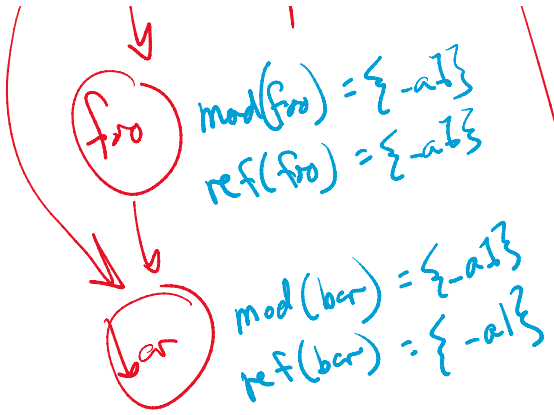
add mod/ref info

mod: modified ref: referenced



1. compute call-graph
   - node for each function
   - edge A→B if function A calls function B
2. compute transitive closure of call graph
3. initialize mod/ref info:
   for each function F, compute
   a) the set of globals & pointed-to objects that F can define (e.g. $store) } mods(F)
   b) the set of globals & pointed-to objects that F can use (e.g. $load) } refs(F)
4. propagate the initial mod/ref info backwards in the transitively closed call graph






```
fn foo(p: a...)
    ⋮
    call_dir bar(p)
    ⋮
}
fn bar(q: &mut ...) -> ...{
    ⋮
    a := $load q
    $store q y2
    ⋮
}
```

improved reaching defs

- no need for fake variables
- all instructions stay the same except

L x = $load y

USE = {y} ∪ pts-to(y)

L $store X <op>

DEF = pts-to(X)

p := $addrof a
$call_dir foo(p)

L [x =] $call-{dir,idr} <id>/fp (args...) then bb

- CALLEES = {id} if $call_dir

```
$call_dir foo(p)
   ⋮
q = &addr_of b
$call_dir foo(q)
```

```
fun foo(r: &int) {
   ⋮
  $store r 42
   ⋮
}
```

- CALLEES = $\begin{cases} \{id\} & \text{if } \$call\_dir \\ pts\text{-}to(fp) & \text{if } \$call\_idr \end{cases}$

- REACHABLE = globals + all objects reachable from globals or arguments (use pts-to soln)

- $WDEF = \left(\bigcup_{c \in CALLEES} mod(c) \right) \cap REACHABLE$

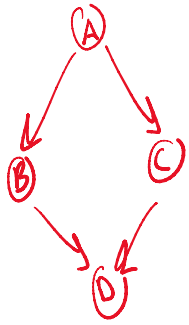
- $USE = \{fp\} \cup \{arg \mid arg \text{ is a variable}\} \cup \left(\left(\bigcup_{c \in CALLEES} ref(c) \right) \cap REACHABLE \right)$

---

control dependence

block X is control-dependent on block Y iff Y is in post-dominance frontier of X

↑ reverse the CFG



PDG

- a node for each instruction

- a node for each instruction
- a data dependence edge A → B if the def at A reaches a use at B
- a control dependence edge A → B if B is control dependent on A
  - if block X is control dependent on block Y, then all instructions in X are control dependent on the terminal of Y