Type Soundness

Types and evaluation

- Why is a type system useful?

  → It can rule out ill-formed programs before we run them

- What information can a type system give us?

  → The type of data the program should produce as a result

- What is the relationship between:

$$\Gamma \vdash \mathbf{e} : \tau$$

and

`interp-expr : e -> v`

→ **v** should be consistent with τ

- We'd like types to tell us something useful about the behavior of our program at run-time

Type Soundness

If

$\varnothing \vdash$ `e : τ` and

`(interp-expr e) = v`

then

if `τ = number` then `v` is a number

if `τ = (τ₁ -> τ₂)` then `v` is `'procedure`

Type Soundness

- With type soundness, our types accurately predict the kind of data we'll get when we run our program
    - Guaranteed

- Without type soundness, may get bogus predictions
    - So can't rely on it
    - Invitation for bugs, security vulnerabilities, yikes

- Formal property, can be proven mathematically
    - Starting from typing rules
    - Bugs may creep in as you go from rules to code!

Type Soundness

Not all type systems used in practice are sound!

- Standard ML: proven sound
- Haskell: subsets have been proven sound
  - Whole type system proven sound at one point
  - But constantly evolves, so may be out of date
- Rust: proven sound, at least a subset (IIRC)
- Java: has soundness holes, but mostly hangs together
  - But soundness holes are enough for security holes!
- C: lol, what's soundness