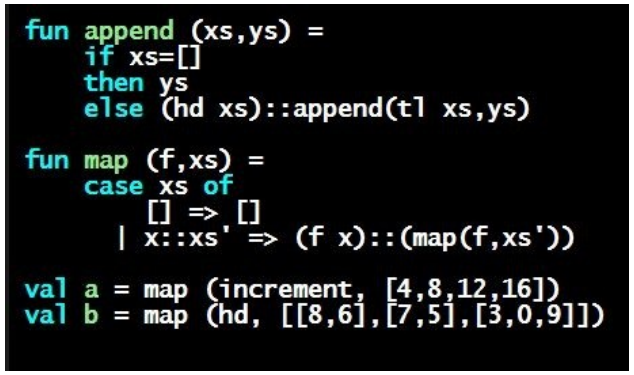

Programming Languages

Dan Grossman

2013

What is Static Checking?

Static checking

- *Static checking* is anything done to reject a program *after* it (successfully) parses but *before* it runs

- ***Part of a PL's definition: what static checking is performed***
  - A "helpful tool" could do more checking

- Common way to define a PL's static checking is via a *type system*
  - *Approach* is to give each variable, expression, etc. a type
  - *Purposes* include preventing misuse of primitives (e.g., `4/"hi"`), enforcing abstraction, and avoiding dynamic checking
    - Dynamic means at run-time

- Dynamically-typed languages do (almost) no static checking
  - Line is not absolute

Example: ML, what types prevent

In ML, type-checking ensures a program (when run) will **never** have:

- A primitive operation used on a value of the wrong type
  - Arithmetic on a non-number
  - `e1 e2` where `e1` does not evaluate to a function
  - A non-boolean between `if` and `then`
- A variable not defined in the environment
- A pattern-match with a redundant pattern
- Code outside a module call a function not in the module's signature
- …

(First two are "standard" for type systems, but different languages' type systems ensure different things)

Example: ML, what types allow

In ML, type-checking does ***not*** prevent any of these errors

- Instead, detected at run-time

- Calling functions such that exceptions occur, e.g., **`hd []`**
- An array-bounds error
- Division-by-zero

In general, no type system prevents logic / algorithmic errors:

- Reversing the branches of a conditional
- Calling **`f`** instead of **`g`**

(Without a program specification, type-checker can't "read minds")

Purpose is to prevent something

Have discussed facts about *what* the ML type system does and does not prevent

- Separate from *how* (e.g., one type for each variable) though previously studied many of ML's typing rules

Language design includes deciding *what* is checked and *how*

Hard part is making sure the type system "achieves its purpose"

- That "the how" accomplishes "the what"
- More precise definition next

A question of eagerness

"Catching a bug before it matters"
is in inherent tension with
"Don't report a bug that might not matter"

Static checking / dynamic checking are two points on a continuum

Silly example: Suppose we just want to prevent evaluating `3 / 0`

- Keystroke time: disallow it in the editor
- Compile time: disallow it if seen in code
- Link time: disallow it if seen in code that may be called to evaluate **`main`**
- Run time: disallow it right when we get to the division
- Later: Instead of doing the division, return `+inf.0` instead
  - Just like `3.0 / 0.0` does in every (?) PL (it's useful!)