HW3 Grades
Median 69.5/85 81%
Avg 63.3/85 74.5%
Max 84/85
Stdev 17.5

for (x < list) yield f(x)
listMap (list) (f)

\( U : \text{List}[A] \quad f : A \Rightarrow B \)
\( O^2) \text{ for } (i < \text{List.range}(0, u \cdot \text{length})) \text{ yield } f(U(i)) \)
\( \sim \text{listMap}(U)(f) \quad O(n) \)

How was HW4?
Type Checking

true + 3 \rightarrow \text{type error}

Dynamic Type Checking
Python, JavaScript, Ruby

Static Type Checking
before execution - rule out bad programs

\((x: \text{Int}) \Rightarrow x + 1)\) ok

\((x: \text{Int}) \Rightarrow '1')\) ok

\((x: \text{Int}) \Rightarrow 1) (\text{true})\)

No by what we would like
\(
\Rightarrow \text{statically}
\)
Yes by dynamic typing
A type system consists of

1) a language of types—a classification of values

→ Why? Know what operations are valid/can be performed on different "types of values"

\[ e_1 \times e_2 \] should only be applied on integers (Int)

2) typing rules: rules that govern whether a program respects types—assigning types to expressions—type checking algorithm

A type error is an expression that violates those prescribed rules.
val \( x : T = (y : U) \)

when should this be a type error?

1. Never an Error?
   \[ U = T \]

2. Maybe an Error?
   \[ U \subseteq T \text{ subtyping} \]

3. Always an Error?
   "not similar"

Set interpretation of types

\[ \text{(base)} \quad \text{Int} \]

\[ 0, 1, 2 - 1 \]

\[ T \]
typing judgement

- relate an expression and type

\[ e : \tau \]

Informally, if \( e \rightarrow^* v \)
then \( v \in \text{set described by } \tau \)

\[ e ::= x | \texttt{nil} | \texttt{bool} | (\texttt{\_} : \tau ) \rightarrow e_1 | - e_1 | ! e_1 \]

what about variables?

what is the type of \( x \)?

we need to say what we want
its type to be.

we need a \underline{type environment}

\[ L \text{ mapping from variables to types} \]
\( \Gamma \) type environment

is a finite map from variables to types

\( \Gamma'(x) \) to lookup the type of \( x \)

\( \Gamma[\cdot \mapsto z] \) returns a type env that extends \( \Gamma' \) with \( x \) mapping to \( z \)

- empty environment

\( \Gamma[\cdot \mapsto z](x) = z \)

\( \Gamma[\cdot \mapsto z](y) = \Gamma(y) \) if \( x \neq y \)

\[ \Gamma' = \Gamma[\cdot \mapsto z] \]

\[ (\circ [\cdot \mapsto \text{Int}])(y \mapsto \text{Int}) \]

\[ \Gamma, x : z \]

\[ \Gamma[\cdot \mapsto z] \]

\[ \uparrow \text{same} \]
Typing Judgment

\[ \Gamma \vdash e : \tau \]

In type environment \( \Gamma \), expression \( e \) has type \( \tau \)

Define this judgment

Type inference: Given \( \Gamma, e \) as input, produce a \( \tau \)

Type checking: Given \( \Gamma, e, \tau \) as input, say ok/not ok

\[ \Gamma \vdash x : \Gamma'(x) \]
\[ \Gamma \vdash n : \text{Int} \]
\[ \Gamma \vdash b : \text{Boolean} \]
\[ \Gamma \vdash () : \text{Unit} \]
\[ \Gamma \vdash e_1 : \text{Int} \]
\[ \Gamma \vdash -e_1 : \text{Int} \]
\[ \Gamma \vdash e_1 : \text{Boolean} \]
\[ \Gamma \vdash \bot e_1 : \text{Boolean} \]
\[ \Gamma \vdash e_1 : \text{Int}, \Gamma \vdash e_2 : \text{Int} \]
\[ \Gamma \vdash e_1 \cdot e_2 : \text{Int} \]
\( \Gamma \vdash e_1 : \text{Boolean} \quad \Gamma \vdash e_2 : \text{Boolean} \)

\( \Gamma \vdash e_1 \& \& e_2 : \text{Boolean} \)

false \& \& 3 \rightarrow \text{false}

ok in dynamic type checking

type error in static type checking

Don't know before execution whether \( e_1 \) evaluates to true or false

So we must be conservative

\( \Gamma \vdash e_1 : \text{Boolean} \quad \Gamma \vdash e_2 : \tau_2 \quad \Gamma \vdash e_3 : \tau_3 \quad \tau_3 \leq \tau_2 \)

\( \Gamma \vdash \text{if} \ (e_1) \ e_2 \ \text{else} \ e_3 : \tau_2 \)

\( \text{C/Java} \ e_1 ? \ e_2 : e_3 \)
\[\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \]
\[\Gamma \vdash e_1(e_2) : \tau'\]

\[(x : \text{Int}) \Rightarrow 1 \quad \text{(true)}
\quad \text{Static type error}\]

\[\Gamma \vdash [x \mapsto \tau]e_1 : \tau'\]
\[\Gamma \vdash (x : \tau) \Rightarrow e_1 : \tau \Rightarrow \tau'\]

Derivation that witnesses a typing judgment:

\[\begin{align*}
[x \mapsto \tau] & x : \tau \\
& \quad \text{letrec } x : \tau = e_1, \text{ in } e_2
\end{align*}\]