"I know that we had a big discussion last week about the class and where it is lacking, but to be totally honest I have actually really enjoyed the class up to this point. My only suggestion would be to give example inputs and outputs for all of the functions that are assigned. ... However, I have found the class to be really engaging and interesting and am starting to actually enjoy some of the features that scala has to offer."

- Listen to records on my record player.
- Fed emus, fed kangaroos, and climbed a bridge in Australia.
- Write fiction in my (very limited) spare time.
- Played Division 3 basketball in Milwaukee
- Got a photo I took published in a nature calendar.
- Had a commercial halibut fishing job in Alaska

Homework 5
- Type Equality
  - Add Records/Unions, Recursion to Smalltalk:
  - Type Checker + Interpreter for a realistic language!
Today

Language Extension

- Pairs (unlabeled, binary)  class

- Records (labeled, n-ary)  HW

- Sums (unlabeled, binary)  class

- Unions (labeled, m-ary)  HW

2. Type Equality

In HW4, we implemented interpreter for

Dyn Small

concrete syntax of a program \rightarrow \text{Parser} \rightarrow \text{AST of program} \rightarrow \text{Interpreter (step)} \rightarrow \text{Eval results}

Small

concrete syntax of a program \rightarrow \text{Parser} \rightarrow \text{AST of program} \rightarrow \text{Type Checker} \rightarrow \text{Interpret (try)} \rightarrow \text{AST (crash) interpreter}

Type Safe? Strongly-Typed? - No, don't run!
Type Checkers prevent crashes because of type errors.

= Type Safety

Language Extension

\[
\begin{align*}
\text{Pairs} & \quad e ::= \ldots \mid \text{Unit} \mid (e_1, e_2) \mid e_1 \cdot e_2 \\
\tau ::= \ldots \mid \text{Unit} \mid \tau_1 \times \tau_2 \\

\end{align*}
\]

Note in Scala:

\[
[(\tau_1, \tau_2)]
\]

\[
\begin{align*}
\nu ::= \ldots \mid \text{Unit} \mid (\nu_1, \nu_2)
\end{align*}
\]

pair value
Sums

$$e_1 = \text{injl}_{x_1 + x_2}(e_1) \mid \text{injr}_{x_1 + x_2}(e_2)$$

\(\text{constructor name}\)

\(e_1\) match case \(\text{injl}(x_1) \Rightarrow e_1\)

\(\text{case } \text{injr}(x_2) \Rightarrow e_2\)

\(z := \text{Nothing}\) \(x_1 + x_2\)

\(v := \text{injl}_{x_1 + x_2}(v_1) \mid \text{injr}_{x_1 + x_2}(v_2)\)

\(\text{Right-to-left evaluation}\)

\(e_2 \Rightarrow e_2'\)

\(e_1 \Rightarrow e_1'\)

\((e_1, e_2) \Rightarrow (e_1', e_2')\)

\((e_1, v_2) \Rightarrow (e_1', v_2)\)

\(v := (v_1, v_2)\)

\((v_1, v_2)_{-1} \Rightarrow v_1\)

\(e_1 \Rightarrow e_{15}\)

\(e_1_{-1} \Rightarrow e_{15}.$$
Sum
\[ e_1 \rightarrow e_1 \]

\[ \text{inj}_{l_1+2} (e_1) \rightarrow \text{inj}_{l_1+2} (e_1') \]

+ similar for inj_{r}

\[ \nu \mapsto \cdots \text{inj}_{l_1+2} (\nu) \]

inj_{l_1+2} (\nu) \text{ match case } \text{inj}_{l_1} \Rightarrow e_1, \text{ case } \text{inj}_{l_2} \Rightarrow e_2

\[ \rightarrow [\nu/x] e_1 \]

+ similar for inj_{r}

\[ e \rightarrow e' \]

e match \cdots \rightarrow e' \text{ match } \cdots \]
Type Checking

\[ \Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau \]

\[ \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \]

\[ \text{Int \times Bool} \]

\[ \Gamma \vdash e : \tau_1 \times \tau_2 \]

\[ \Gamma \vdash e_{-1} : \tau \]

+ 0 - 2

\[ \Gamma \vdash \text{inj}_{\tau_1 + \tau_2} (e) : \tau_1 + \tau_2 \]

+ similar for inj_2

\[ \tau = \tau_1 \]

\[ \Gamma \vdash e : \tau_1 + \tau_2 \]

\[ \text{[X]}(e_1 : \tau, e_2 : \tau_2 \rightarrow e) \]

\[ \Gamma \vdash e \text{ match } \text{ case inj}_1(x_1) \rightarrow e_1, \text{ case inj}_2(x_2) \rightarrow e_2 : \tau \]

\[ \text{return type} \]
Type Equality

Structural Equality

\[ \text{Int} \times \text{Boolean} = \text{Int} \times \text{Boolean} \]

Name

[\text{Int} \times \text{Boolean}]_{T1} \neq [\text{Int} \times \text{Boolean}]_{T2}

class C: \text{Int value}; 3

class D: \text{Int value}; 3