Introduction

Welcome to Hype for Types!

- Instructors:
  - Avery Cowan (acowan)
  - Brandon Wu (bjwu)
  - Harrison Grodin (hgrodin)
  - James Gallicchio (jgallicc)
- Attendance
  - In general, you have to come to lecture to pass
  - Let us know if you need to miss a week
- Homework
  - Every lecture will have an associated homework
  - Graded on effort (*not* correctness)
  - If you spend more than an hour, please stop[1]

[1] Unless you're having fun!

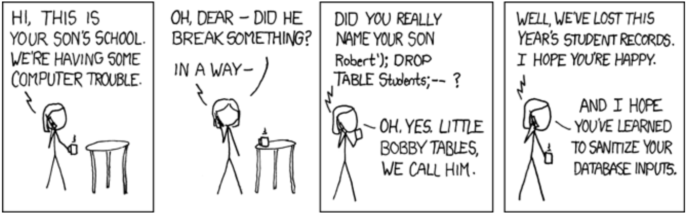
Other Stuff

- Please join the Discord and Gradescope if you haven't
- We assume everyone has 150 level knowledge of functional programming and type systems
  - If you don't have this and feel really lost, send us a message on Discord

Motivation

Programming is Hard

- `1 + "hello"`
- `fun f x = f x`
- `goto not_yet_valid_case;`
- `malloc(sizeof(int)); return;`
- `free(A); free(A);`
- `@requires is_sorted(A)`
- `A[len(A)]`



https://xkcd.com/327/

Types are... hype!

- Rule out a whole class of errors at compile time
- Expressively describe the shape of data
- Could we do more?

Lambda Calculus

Building a tiny language

The *simply-typed lambda calculus* is simple. It only has four features:

- Unit ("empty tuples")
- Booleans
- Tuples
- Functions

Expressions

We represent our expressions using a *grammar*:

$$
\begin{array}{rcll}
e & ::= & x & \text{variable} \\
& | & \langle\rangle & \text{unit} \\
& | & \textbf{false} & \text{false boolean} \\
& | & \textbf{true} & \text{true boolean} \\
& | & \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 & \text{boolean case analysis} \\
& | & \langle e_1, e_2 \rangle & \text{tuple} \\
& | & \textbf{fst}(e) & \text{first tuple element} \\
& | & \textbf{snd}(e) & \text{second tuple element} \\
& | & \lambda x : \tau.\ e & \text{function abstraction (lambda)} \\
& | & e_1\ e_2 & \text{function application}
\end{array}
$$

Types

Similarly, we define our types as follows:

$$\tau ::= \textbf{unit} \mid \textbf{bool} \mid \tau_1 \times \tau_2 \mid \tau_1 \to \tau_2$$

Question

How do we check if $e : \tau$?

Inference Rules

In logic, we use *inference rules* to state how facts follow from other facts.

$$\frac{\text{premise}_1 \quad \text{premise}_2 \quad \ldots}{\text{conclusion}}$$

For example:

$$\frac{\text{you are here} \quad \text{you are hyped}}{\text{you are hyped for types}} \qquad \frac{}{\text{functions are values}}$$

$$\frac{\text{it's raining} \quad x \text{ is outside}}{x \text{ is getting wet}} \qquad \frac{A \text{ ancestor } B \quad B \text{ mother } C}{A \text{ ancestor } C}$$

$$\frac{n \text{ is a number}}{n+1 \text{ is a number}} \qquad \frac{f \text{ total} \quad x \text{ valuable}}{f\ x \text{ valuable}}$$

Typing Rules: First Attempt

Consider the judgement $e : \tau$ ("e has type τ"). Let's try to express some simple typing rules.

$$\frac{}{\langle\rangle : \textbf{unit}} \qquad \frac{}{\textbf{false} : \textbf{bool}} \qquad \frac{}{\textbf{true} : \textbf{bool}} \qquad \frac{e_1 : \textbf{bool} \quad e_2 : \tau \quad e_3 : \tau}{\textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 : \tau}$$

$$\frac{e_1 : \tau_1 \quad e_2 : \tau_2}{\langle e_1, e_2\rangle : \tau_1 \times \tau_2} \qquad \frac{e : \tau_1 \times \tau_2}{\textbf{fst}(e) : \tau_1} \qquad \frac{e : \tau_1 \times \tau_2}{\textbf{snd}(e) : \tau_2}$$

Question

How do we write rules for functions?

Typing Rules: Functions

Let's give it a shot.

$$\frac{e_1 : \tau_1 \to \tau_2 \quad e_2 : \tau_1}{e_1\ e_2 : \tau_2}$$

Looks good so far...

$$\frac{e : \tau_2\ \textcolor{red}{(?)}}{\lambda x : \tau_1.\ e : \tau_1 \to \tau_2}$$

Key Idea

Expressions only have types *given a context*!

Contexts

Intuition

If, given $x : \tau_1$, we know $e : \tau_2$, then $\lambda x : \tau_1.\ e : \tau_1 \to \tau_2$.

Therefore, we need a context (denoted Γ) which associates types with variables.

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2}$$

What types does some variable x have? It depends on the previous code!

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}$$

All the rules!

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau}\ (\textsc{Var}) \qquad \frac{}{\Gamma \vdash \langle\rangle : \textbf{unit}}\ (\textsc{Unit}) \qquad \frac{}{\Gamma \vdash \textbf{false} : \textbf{bool}}\ (\textsc{False})$$

$$\frac{}{\Gamma \vdash \textbf{true} : \textbf{bool}}\ (\textsc{True}) \qquad \frac{\Gamma \vdash e_1 : \textbf{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3 : \tau}\ (\textsc{If})$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}\ (\textsc{Tup}) \qquad \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \textbf{fst}(e) : \tau_1}\ (\textsc{Fst})$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \textbf{snd}(e) : \tau_2}\ (\textsc{Snd}) \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1.\ e : \tau_1 \to \tau_2}\ (\textsc{Abs})$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1\ e_2 : \tau_2}\ (\textsc{App})$$

Example: what's the type?

Let's derive that

$$\cdot \vdash (\lambda x : \textbf{unit}.\ \langle x, \textbf{true} \rangle)\ \langle\rangle : \textbf{unit} \times \textbf{bool}$$

by using the rules.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \textbf{unit} \in \cdot, x : \textbf{unit}}}{\cdot, x : \textbf{unit} \vdash x : \textbf{unit}}\ (\text{VAR}) \quad \dfrac{}{\cdot, x : \textbf{unit} \vdash \textbf{true} : \textbf{bool}}\ (\text{TRUE})}{\cdot, x : \textbf{unit} \vdash \langle x, \textbf{true} \rangle : \textbf{unit} \times \textbf{bool}}\ (\text{TUP})}{\cdot \vdash \lambda x : \textbf{unit}.\ \langle x, \textbf{true} \rangle : \textbf{unit} \to \textbf{unit} \times \textbf{bool}}\ (\text{ABS}) \quad \dfrac{}{\cdot \vdash \langle\rangle : \textbf{unit}}\ (\text{UNIT})}{\cdot \vdash (\lambda x : \textbf{unit}.\ \langle x, \textbf{true} \rangle)\ \langle\rangle : \textbf{unit} \times \textbf{bool}}\ (\text{APP})$$

Homework Foreshadowing

That looks like a trace of a typechecking algorithm!

Get Hype.

The Future is Bright

- How can you use basic algebra to manipulate types?
- How do types and programs relate to logical proofs?
- How can we automatically fold (and unfold) any recursive type?
- How can types allow us to do safe imperative programming?
- Can we make it so that programs that typecheck iff they're correct?