

# Introduction

# Welcome to Hype for Types!

- Instructors:
  - ▶ Avery Cowan (acowan)
  - ▶ Aditi Gupta (aditig)
  - ▶ Harrison Grodin (hgrodin)
  - ▶ Matthew McQuaid (mmcquaid)
- Attendance
  - ▶ In general, you have to come to lecture to pass
  - ▶ If you can't attend because of timezone issues, let us know
  - ▶ Just email us 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<sup>1</sup>

---

<sup>1</sup>Unless you're having fun!

# Other Stuff

- Please join the Slack 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 Slack

# 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*:

$e ::= x$	variable
$\langle \rangle$	unit
<b>false</b>	false boolean
<b>true</b>	true boolean
<b>if</b> $e_1$ <b>then</b> $e_2$ <b>else</b> $e_3$	boolean case analysis
$\langle e_1, e_2 \rangle$	tuple
<b>fst</b> ( $e$ )	first tuple element
<b>snd</b> ( $e$ )	second tuple element
$\lambda x : \tau. e$	function abstraction (lambda)
$e_1 e_2$	function application

# Types

Similarly, we define our types as follows:

$$\begin{array}{l} \tau ::= \mathbf{unit} \\ \quad | \mathbf{bool} \\ \quad | \tau_1 \times \tau_2 \\ \quad | \tau_1 \rightarrow \tau_2 \end{array}$$

# Types

Similarly, we define our types as follows:

$$\begin{array}{l} \tau ::= \mathbf{unit} \\ \quad | \mathbf{bool} \\ \quad | \tau_1 \times \tau_2 \\ \quad | \tau_1 \rightarrow \tau_2 \end{array}$$

## Question

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

# Inference Rules

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

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

# Inference Rules

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

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

For example:

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

$$\frac{}{\text{functions are values}}$$

$$\frac{\text{it's raining} \quad x \text{ is outside}}{x \text{ is getting wet}}$$

$$\frac{A \text{ ancestor } B \quad B \text{ mother } C}{A \text{ ancestor } C}$$

$$\frac{n \text{ is a number}}{n + 1 \text{ is a number}}$$

$$\frac{f \text{ total} \quad x \text{ valuable}}{f \ x \text{ valuable}}$$

# Typing Rules: First Attempt

Consider the judgement  $e : \tau$  (“ $e$  has type  $\tau$ ”). Let’s try to express some simple typing rules.

$$\frac{}{\langle \rangle : \mathbf{unit}}$$

$$\frac{}{\mathbf{false} : \mathbf{bool}}$$

$$\frac{}{\mathbf{true} : \mathbf{bool}}$$

$$\frac{e_1 : \mathbf{bool} \quad e_2 : \tau \quad e_3 : \tau}{\mathbf{if } e_1 \mathbf{ then } e_2 \mathbf{ else } e_3 : \tau}$$

$$\frac{e_1 : \tau_1 \quad e_2 : \tau_2}{\langle e_1, e_2 \rangle : \tau_1 \times \tau_2}$$

$$\frac{e : \tau_1 \times \tau_2}{\mathbf{fst}(e) : \tau_1}$$

$$\frac{e : \tau_1 \times \tau_2}{\mathbf{snd}(e) : \tau_2}$$

# Typing Rules: First Attempt

Consider the judgement  $e : \tau$  (“ $e$  has type  $\tau$ ”). Let’s try to express some simple typing rules.

$$\frac{}{\langle \rangle : \mathbf{unit}}$$

$$\frac{}{\mathbf{false} : \mathbf{bool}}$$

$$\frac{}{\mathbf{true} : \mathbf{bool}}$$

$$\frac{e_1 : \mathbf{bool} \quad e_2 : \tau \quad e_3 : \tau}{\mathbf{if } e_1 \mathbf{ then } e_2 \mathbf{ else } e_3 : \tau}$$

$$\frac{e_1 : \tau_1 \quad e_2 : \tau_2}{\langle e_1, e_2 \rangle : \tau_1 \times \tau_2}$$

$$\frac{e : \tau_1 \times \tau_2}{\mathbf{fst}(e) : \tau_1}$$

$$\frac{e : \tau_1 \times \tau_2}{\mathbf{snd}(e) : \tau_2}$$

## Question

How do we write rules for functions?

# Typing Rules: Functions

Let's give it a shot.

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

Looks good so far...



# Typing Rules: Functions

Let's give it a shot.

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

Looks good so far...

$$\frac{e : \tau_2 \text{ (?)}}{\lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2}$$

# Typing Rules: Functions

Let's give it a shot.

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

Looks good so far...

$$\frac{e : \tau_2 \text{ (?)}}{\lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2}$$

## Key Idea

Expressions only have types *given a context!*

# Contexts

## Intuition

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

Therefore, we need a context (denoted  $\Gamma$ ) which associates types with variables.

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

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!

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} \text{ (VAR)}$$

$$\frac{}{\Gamma \vdash \langle \rangle : \mathbf{unit}} \text{ (UNIT)}$$

$$\frac{}{\Gamma \vdash \mathbf{false} : \mathbf{bool}} \text{ (FALSE)}$$

$$\frac{}{\Gamma \vdash \mathbf{true} : \mathbf{bool}} \text{ (TRUE)}$$

$$\frac{\Gamma \vdash e_1 : \mathbf{bool} \quad \Gamma \vdash e_2 : \tau \quad \Gamma \vdash e_3 : \tau}{\Gamma \vdash \mathbf{if } e_1 \mathbf{ then } e_2 \mathbf{ else } e_3 : \tau} \text{ (IF)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \text{ (TUP)}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathbf{fst}(e) : \tau_1} \text{ (FST)}$$

$$\frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \mathbf{snd}(e) : \tau_2} \text{ (SND)}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2} \text{ (ABS)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2} \text{ (APP)}$$

## Example: what's the type?

Let's derive that

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

by using the rules.

## Example: what's the type?

Let's derive that

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

by using the rules.

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

## Example: what's the type?

Let's derive that

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

by using the rules.

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

### 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?