# Introduction and Lambda Calculus

Hype for Types

August 26, 2024

# Introduction

# Welcome to Hype for Types!

- Instructors:
  - Ari Battleman (zbattlem)
  - Kiera O'Flynn (koflynn)
  - Sonya Simkin (ssimkin)
- 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!

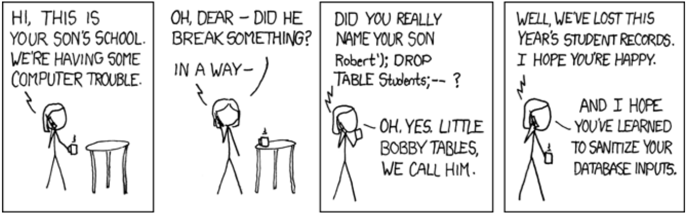
# Other Stuff

- 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, talk to us after class (and a 150 head TA will bring you up to speed)

# Motivation

# Programming is Hard

There are many common classes of mistakes/bugs/errors in code:

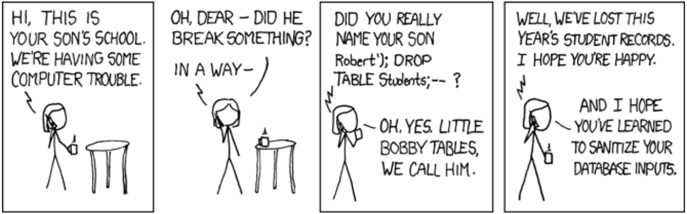



https://xkcd.com/327/

# Programming is Hard

There are many common classes of mistakes/bugs/errors in code:

- `1 + "hello"`

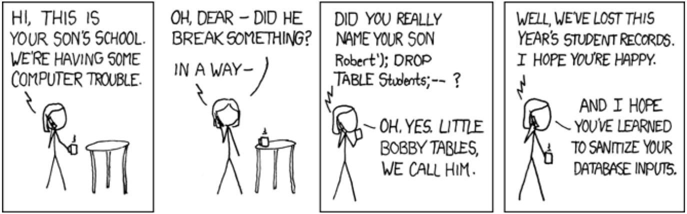



https://xkcd.com/327/

# Programming is Hard

There are many common classes of mistakes/bugs/errors in code:

- `1 + "hello"`
- `fun f x = f x`

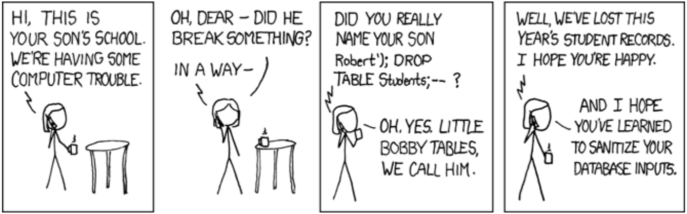



https://xkcd.com/327/

# Programming is Hard

There are many common classes of mistakes/bugs/errors in code:

- `1 + "hello"`
- `fun f x = f x`
- `malloc(sizeof(int)); return;`

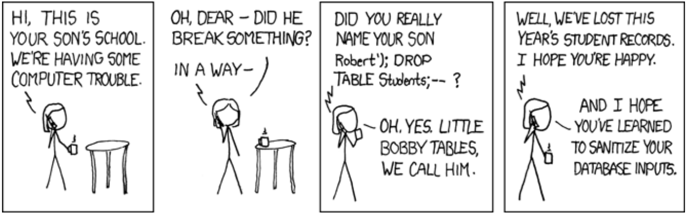



https://xkcd.com/327/

# Programming is Hard

There are many common classes of mistakes/bugs/errors in code:

- `1 + "hello"`
- `fun f x = f x`
- `malloc(sizeof(int)); return;`
- `free(A); free(A);`

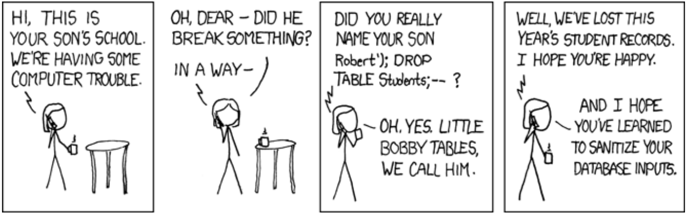



https://xkcd.com/327/

# Programming is Hard

There are many common classes of mistakes/bugs/errors in code:

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

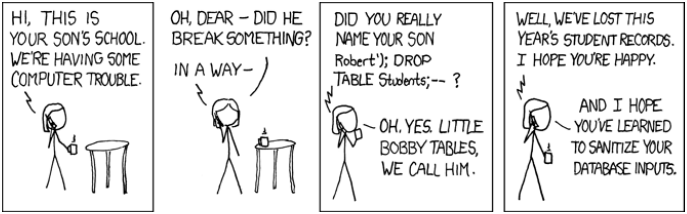



https://xkcd.com/327/

# Programming is Hard

There are many common classes of mistakes/bugs/errors in code:

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




https://xkcd.com/327/

# Types are... hype!

Types are *descriptions* of how some piece of data can be used.

[2] Foreshadowing: "a literary device in which a writer gives an advance hint of what is to come later in the story." *Wikipedia, "Foreshadowing," retrieved 30 Aug 2022*

# Types are... hype!

Types are *descriptions* of how some piece of data can be used.

## Guiding Question

How can we use types to catch errors at compile-time?

[2] Foreshadowing: "a literary device in which a writer gives an advance hint of what is to come later in the story." *Wikipedia, "Foreshadowing," retrieved 30 Aug 2022*

# Types are... hype!

Types are *descriptions* of how some piece of data can be used.

**Guiding Question**

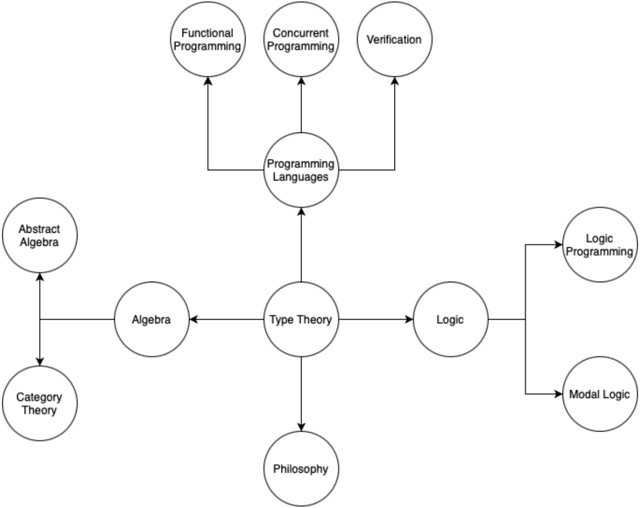
How can we use types to catch errors at compile-time?

**Guiding Question**

Can we use types for more than just bug-catching?[2]

[2] Foreshadowing: "a literary device in which a writer gives an advance hint of what is to come later in the story." *Wikipedia, "Foreshadowing," retrieved 30 Aug 2022*

# Type Theory at Large



# Goal of This Course

- We do not ask students to master the content as in an academic course
- We do not replace any academic courses
- We do not focus on depth, but rather focus on breath

# Goal of This Course

- We do not ask students to master the content as in an academic course
- We do not replace any academic courses
- We do not focus on depth, but rather focus on breath
- We DO expect you to have fun
- We DO hope to spark your interest in PL theory and start pursuing coursework and/or research in adjacent areas after taking this course
- We DO want you to learn about different fascinating aspects of types that you would otherwise take advanced courses and/or read complicated academic papers to understand

# Course Credit

- 3 unit, P/F
- For undergraduate, count towards 360 total units graduation requirement
- For MSCS, count towards 12 units “MSCS elective units”

# Caveat

You will see a lot of weird symbols in this class, please don't be intimated.
We especially love $\lambda$.

# Lambda Calculus

# Building a tiny language

The *simply-typed lambda calculus* is simple. It only has four features[3]:

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

[3] which is a subset of Standard ML

# Building a tiny language

The *simply-typed lambda calculus* is simple. It only has four features[3]:

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

**Goal**

To use STLC as a tool to study how type checker works.

[3] which is a subset of Standard ML

# Expressions

We represent our expressions using a *grammar*:

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

# Types

Similarly, we define our types as follows:

$$
\begin{aligned}
\tau \quad ::= \quad & \textbf{unit} \\
\mid \quad & \textbf{bool} \\
\mid \quad & \tau_1 \times \tau_2 \\
\mid \quad & \tau_1 \rightarrow \tau_2
\end{aligned}
$$

# Types

Similarly, we define our types as follows:

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

**Million-dollar Question**

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

# Inference Rules

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

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

# 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:

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

$$\frac{\text{it's raining} \quad x \text{ is outside}}{x \text{ is getting wet}} \qquad \frac{\text{Socrates is a man} \quad \text{All men are mortal}}{\text{Socrates is mortal}}$$

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

# Typing Rules: First Attempt

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

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

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

# Typing Rules: First Attempt

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

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

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

**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...

# 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...

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

# 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...

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

**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 $\Gamma$) 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.

# 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.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \textbf{unit} \in \cdot, x : \textbf{unit}}}{\cdot, x : \textbf{unit} \vdash x : \textbf{unit}}\ (\textsc{var}) \qquad \dfrac{}{\cdot, x : \textbf{unit} \vdash \textbf{true} : \textbf{bool}}\ (\textsc{true})}{\cdot, x : \textbf{unit} \vdash \langle x, \textbf{true} \rangle : \textbf{unit} \times \textbf{bool}}\ (\textsc{tup})}{\cdot \vdash \lambda x : \textbf{unit}.\ \langle x, \textbf{true} \rangle : \textbf{unit} \to \textbf{unit} \times \textbf{bool}}\ (\textsc{abs}) \qquad \dfrac{}{\cdot \vdash \langle\rangle : \textbf{unit}}\ (\textsc{unit})}{\cdot \vdash (\lambda x : \textbf{unit}.\ \langle x, \textbf{true} \rangle)\ \langle\rangle : \textbf{unit} \times \textbf{bool}}\ (\textsc{app})$$

# 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.

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

## 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?