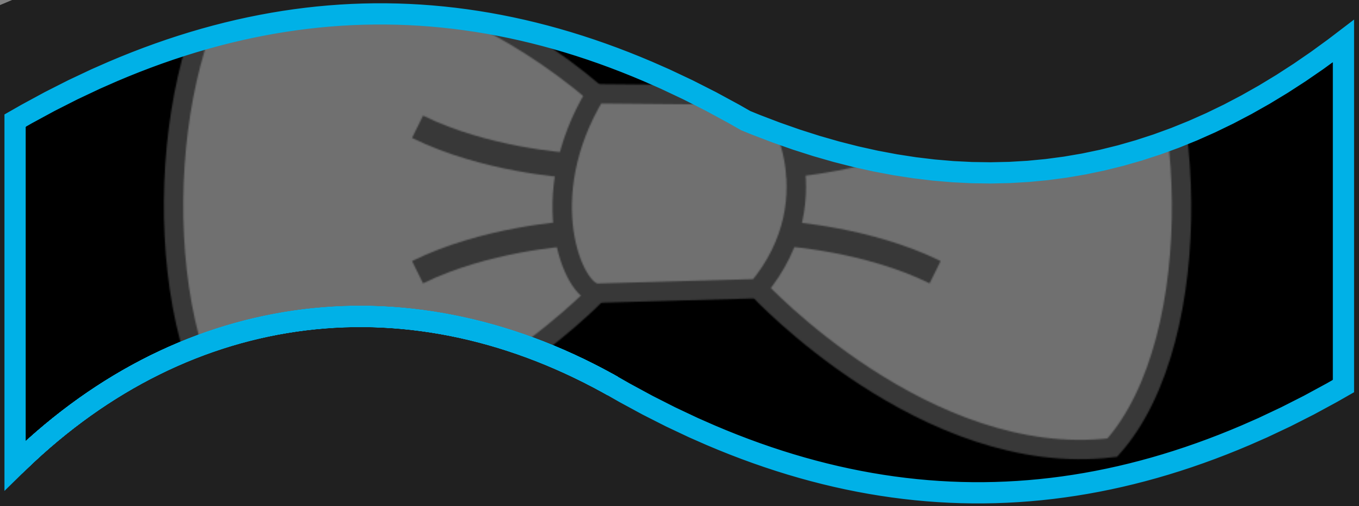
Intro to
Homotopy Type Theory



Formalities & Informalities

Intro to Homotopy Type Theory, No. 2

Formalities & Informalities

Dress Coding

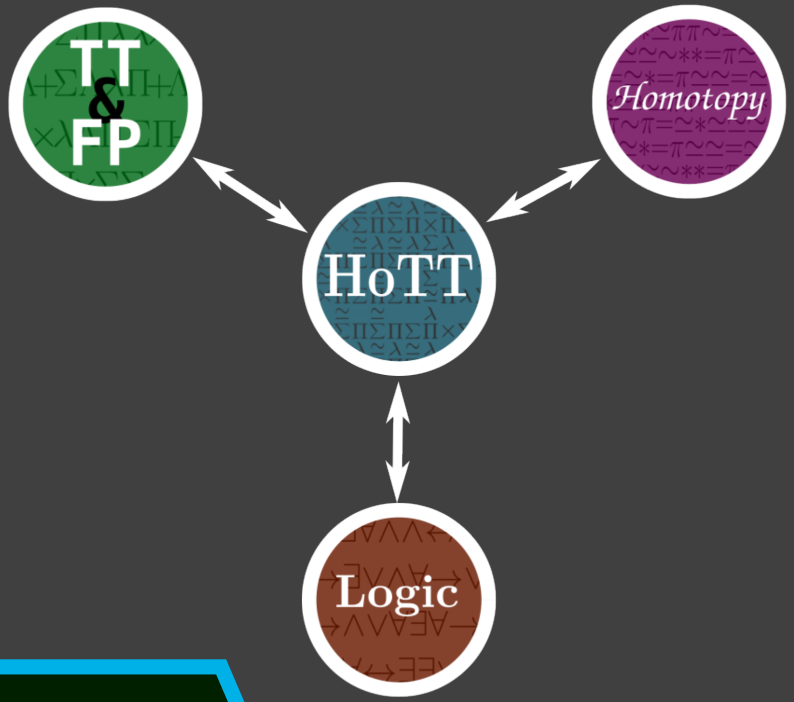

Business
Casual


BLACK
TIE


White
Tie

0 HoTT Workflows

Why HoTT?
What does HoTT mean?



How do we do HoTT?

- Written for humans, in sentences and paragraphs
- Primary way of doing mathematics
- Key innovation of the HoTT/UF project: developing informal type theory
- Informal $\neq$ unrigorous

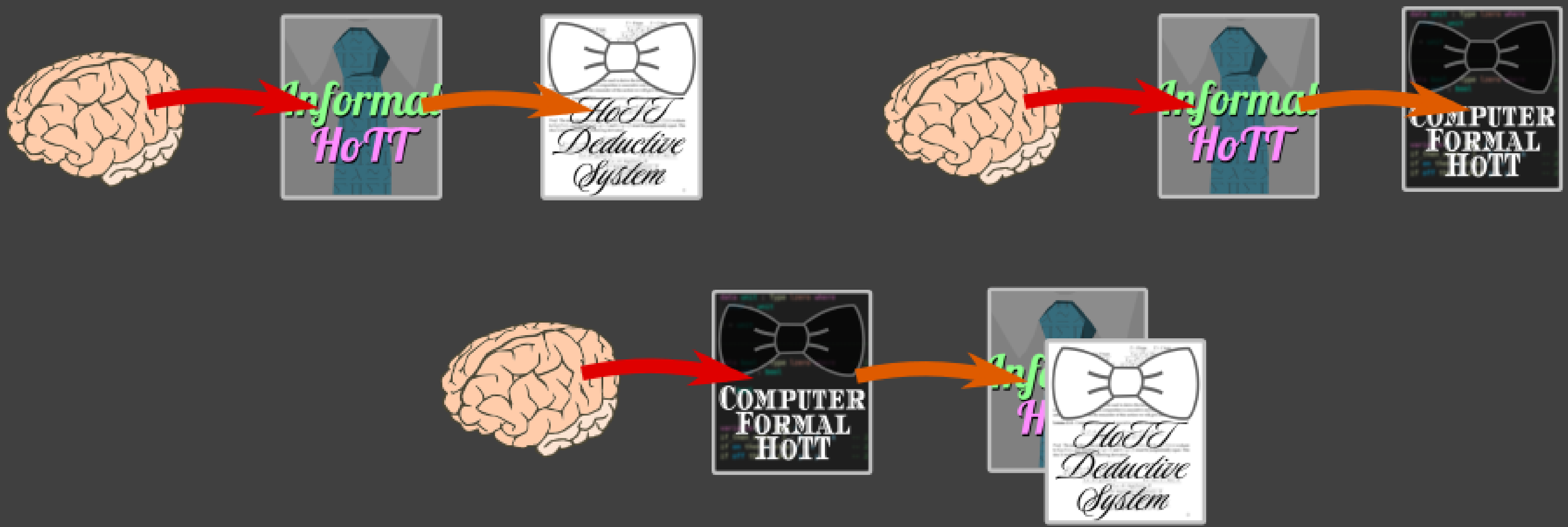
- Written in a computer proof assistant (e.g. Agda, Coq, Lean)
- Correctness can be checked automatically
- Central motivation for HoTT: informal theory is *amenable* to formalization in a computer proof assistant

- Written in the form of inference rules, e.g.

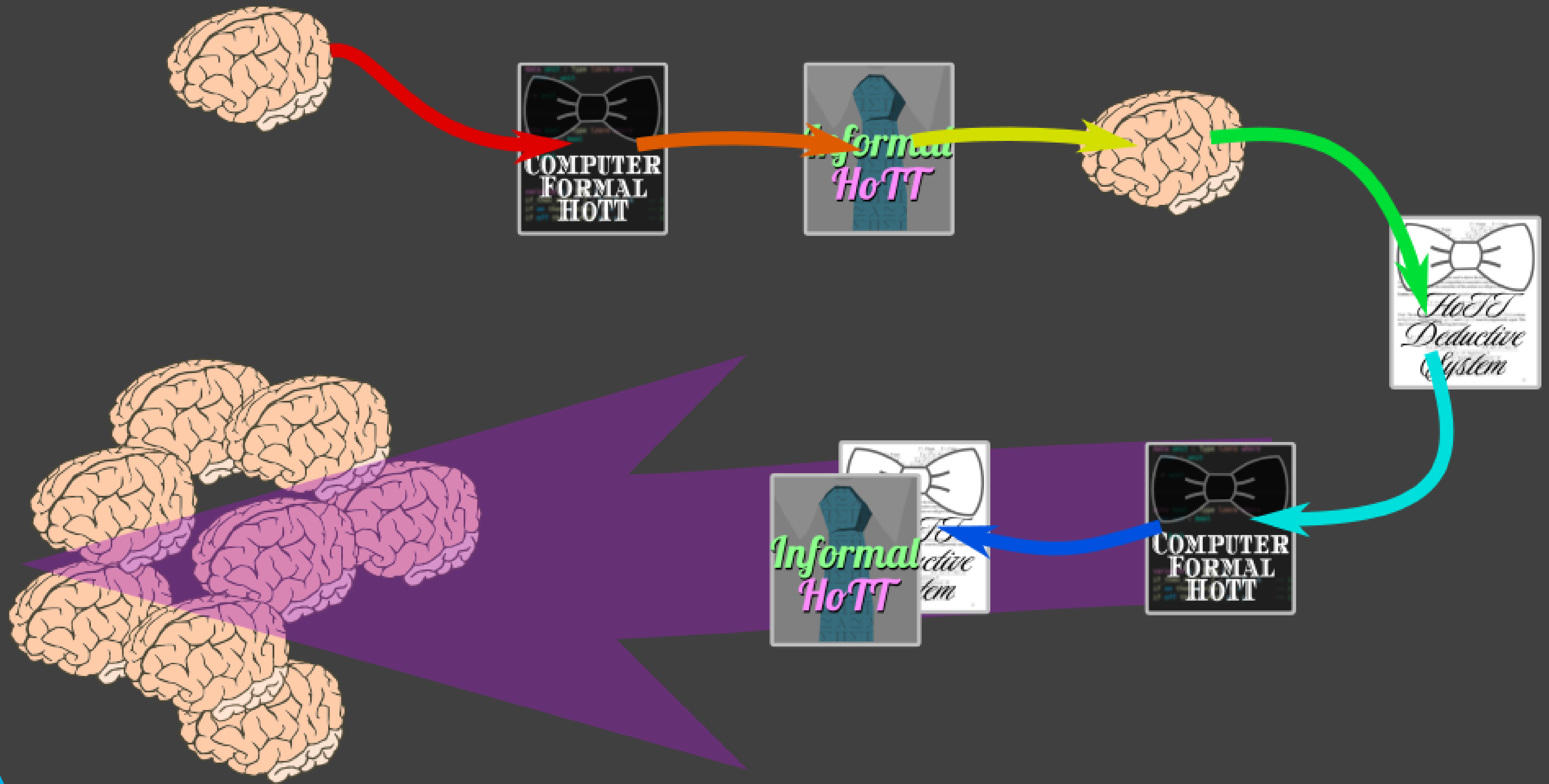
$$\frac{\Gamma \vdash f : A \to B \quad \Gamma \vdash x : A}{\Gamma \vdash f(x) : B}$$

- Unwieldy as a formalization system, but often a convenient for
  - precisely stating rules
  - reasoning about metatheory
  - figuring out how to formalize in a computer

HoTT Workflow Suggestions



HoTT Workflow Suggestions



How do we *do* HoTT?

The formalization we'll use

$\downarrow$ Links in description! $\downarrow$

1 Declare-It-Yourself

Declaring new terms in Agda

$\mathbf{1}$ is a type with exactly one term, $\star : \mathbf{1}$

Programming	Homotopy	Logic
Type of zero-tuples	Contractible (single-point) space	Uniquely-witnessed true proposition

01-simpleTT.agda

```
data unit : Type lzero where
  star : unit

𝟙 = unit
```

`Type` is a *hierarchy of universes*, parametrized by a type `Level`. `Level` is basically the natural numbers: `lzero` is level 0, `lsuc` is the successor operation, and $\sqcup$ is the maximum operator. So there are as many `Type` levels as there are natural numbers.

We have an infinite hierarchy to avoid Russell-paradox-esque problems with having a *type of types*. For each ℓ : `Level`,

```
(Type ℓ) : Type (lsuc ℓ)
```

so no type is a term of itself.

The `Type` universes are cumulative: if `A : Type` ℓ, then `A` can also be viewed as an element of `Type` ℓ' for every ℓ':`Level` higher than ℓ. So if we say `A : Type lzero`, this means that `A` is a type at *every* level.

00-preamble.agda

```
module 00-preamble where

open import Agda.Primitive using (Level;
  lzero; lsuc; _⊔_) public

variable ℓ : Level

Type : (ℓ : Level) → Set (lsuc ℓ)
Type ℓ = Set ℓ
```

Judgments and Inference Rules

$$\frac{\mathcal{J}_1 \quad \mathcal{J}_2 \quad \dots \quad \mathcal{J}_n}{\mathcal{C}}$$

- "If $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_n$ hold, $\mathcal{C}$ follows"
- $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_n$, and $\mathcal{C}$ are **judgments**

$$A \text{ type} \qquad\qquad a : A$$

- Can be stacked atop each other to make *deduction trees*

Formation

$$\frac{}{\mathbf{1}\ \mathsf{type}}$$

Introduction

$$\frac{}{\star : \mathbf{1}}$$

Type definition checklist

- Formation Rule
  - Assert the existence of the type
- Introduction Rule(s)
  - Specify how to give terms of the type
- Elimination Rule
- Computation Rule(s)
- Coherence Rule(s)

The Formation Rule and Introduction Rule are achieved in Computer Formal HoTT (e.g. in Agda) by the type delcaration

01-simpleTT.agda

```
-- 𝟙-Formation & 𝟙-Introduction
data unit : Type lzero where
  star : unit
```

This declares the type into existence (Formation) and declares how to build terms of the type (Introduction)

Context-Dependence

A **context** is a finite list of typed variable names

$$x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$$

We use letters like Γ and Δ to denote arbitrary contexts.

A **judgment-in-context** has the form

$$\Gamma \vdash \mathcal{J}$$

where $\mathcal{J}$ might contain variables from Γ.

Formation

$$\frac{}{\Gamma \vdash \mathbf{1}\ \mathsf{type}}$$

Introduction

$$\frac{}{\Gamma \vdash \star : \mathbf{1}}$$

2 Judgmental Equality

$$3 + 3 \doteq 6$$

$$T_1 \doteq T_2$$

$$t_1 \doteq t_2 : T$$

T_1 and T_2 are judgmentally equal types

t_1 and t_2 are judgmentally equal terms of type T

$$\Gamma \vdash T_1 \doteq T_2$$

In context Γ, T_1 and T_2 are judgmentally equal types

$$\Gamma \vdash t_1 \doteq t_2 : T$$

In context Γ, t_1 and t_2 are judgmentally equal terms of type T

Confluence

"HoTT is a programming language"

Confluence

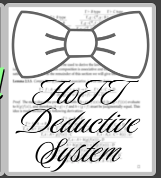
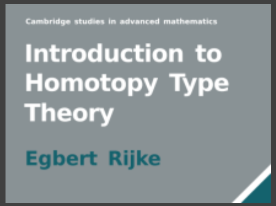
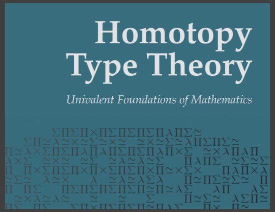
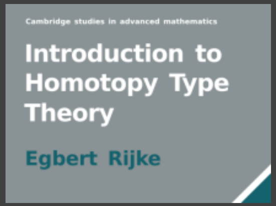
Confluence

$$\begin{tikzcd}
\mathtt{6+6} \arrow[dr] & & \mathtt{7+5} \arrow[dl] \\
& \mathtt{12} &
\end{tikzcd}$$

Computing with Agda

Computational Judgmental Equality

- Compute to the same thing
  - $6+6 \doteq 7+5$
- Equal by definition
  - $\texttt{helloWord} \doteq \texttt{"Hello"}$

   Intro to Homotopy Type Theory	 	  Intro to Homotopy Type Theory
$\doteq$	$\equiv$	$=$

Judgmental Equality is an equivalence relation

$$\frac{A \text{ type}}{A \doteq A} \qquad \frac{a : A}{a \doteq a : A}$$

$$\frac{A \doteq B}{B \doteq A} \qquad \frac{a \doteq a' : A}{a' \doteq a : A}$$

$$\frac{A \doteq B \quad \Gamma \vdash B \doteq C}{A \doteq C} \qquad \frac{a \doteq a' : A \quad a' \doteq a'' : A}{a \doteq a'' : A}$$

Judgmental Equality is an equivalence relation

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash A \doteq A} \qquad \frac{\Gamma \vdash a : A}{\Gamma \vdash a \doteq a : A}$$

$$\frac{\Gamma \vdash A \doteq B}{\Gamma \vdash B \doteq A} \qquad \frac{\Gamma \vdash a \doteq a' : A}{\Gamma \vdash a' \doteq a : A}$$

$$\frac{\Gamma \vdash A \doteq B \quad \Gamma \vdash B \doteq C}{\Gamma \vdash A \doteq C} \qquad \frac{\Gamma \vdash a \doteq a' : A \quad \Gamma \vdash a' \doteq a'' : A}{\Gamma \vdash a \doteq a'' : A}$$

3 Not-So-Casual Friday

We define the type of *days of the week* to be a type **day**, equipped with exactly seven terms

$$\mathbf{Sunday}, \mathbf{Monday}, \ldots, \mathbf{Saturday} \quad : \quad \mathbf{day}$$

$$\frac{}{\textbf{day type}} \qquad (\textbf{day}\text{-Formation})$$

$$\frac{}{\textbf{Sunday} : \textbf{day}} \quad \frac{}{\textbf{Monday} : \textbf{day}} \qquad (\textbf{day}\text{-Introduction})$$

$$\frac{}{\textbf{Tuesday} : \textbf{day}} \quad \frac{}{\textbf{Wednesday} : \textbf{day}}$$

$$\frac{}{\textbf{Thursday} : \textbf{day}} \quad \frac{}{\textbf{Friday} : \textbf{day}}$$

$$\frac{}{\textbf{Saturday} : \textbf{day}}$$

For each $\mathbf{d} : \mathbf{day}$, we have

$$\mathbf{next(d)} : \mathbf{day} \qquad \text{and} \qquad \mathbf{prev(d)} : \mathbf{day}$$

representing the next and previous day, respectively. For instance,

$$\mathbf{next(Tuesday)} \doteq \mathbf{Wednesday} \doteq \mathbf{prev(Thursday)}$$

$$\frac{\mathsf{d} : \mathsf{day}}{\mathsf{next}(\mathsf{d}) : \mathsf{day}} \qquad \frac{\mathsf{d} : \mathsf{day}}{\mathsf{prev}(\mathsf{d}) : \mathsf{day}}$$

$$\frac{}{\mathsf{next}(\mathsf{Sunday}) \doteq \mathsf{Monday} : \mathsf{day}} \quad \frac{}{\mathsf{next}(\mathsf{Monday}) \doteq \mathsf{Tuesday} : \mathsf{day}} \quad \frac{}{\mathsf{next}(\mathsf{Tuesday}) \doteq \mathsf{Wednesday} : \mathsf{day}}$$

$$\frac{}{\mathsf{next}(\mathsf{Wednesday}) \doteq \mathsf{Thursday} : \mathsf{day}} \quad \frac{}{\mathsf{next}(\mathsf{Thursday}) \doteq \mathsf{Friday} : \mathsf{day}} \quad \frac{}{\mathsf{next}(\mathsf{Friday}) \doteq \mathsf{Saturday} : \mathsf{day}}$$

$$\frac{}{\mathsf{next}(\mathsf{Saturday}) \doteq \mathsf{Sunday} : \mathsf{day}}$$

$$\frac{\mathsf{next}(\mathsf{d}_1) \doteq \mathsf{d}_2 : \mathsf{day}}{\mathsf{prev}(\mathsf{d}_2) \doteq \mathsf{d}_1 : \mathsf{day}}$$

example-daysOfWeek.agda

```
data day : Type lzero where
  Sunday Monday Tuesday Wednesday Thursday
   Friday Saturday : day

next : day → day
next Sunday = Monday
next Monday = Tuesday
next Tuesday = Wednesday
next Wednesday = Thursday
next Thursday = Friday
next Friday = Saturday
```

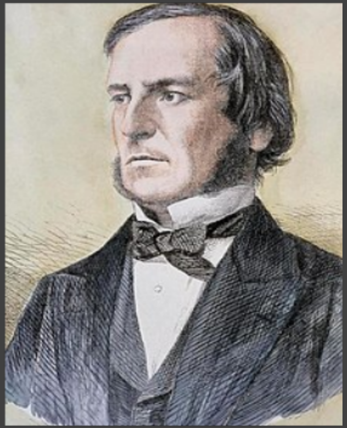
$$
\begin{aligned}
&\mathsf{next}(\mathsf{next}(\mathsf{next}(\mathsf{Tuesday})))\\
&\doteq \mathsf{next}(\mathsf{next}(\mathsf{Wednesday}))\\
&\doteq \mathsf{next}(\mathsf{Thursday})\\
&\doteq \mathsf{Friday}\\
&\doteq \mathsf{prev}(\mathsf{Saturday})
\end{aligned}
$$

Indeed. . .

Summary: How to do HoTT

1. Think about what structure/behavior you want to describe rigorously/mathematically
2. Write it up informally (or formally in a computer proof assistant, or as inference rules)
3. Try (un)formalizing it into other styles of HoTT, to better understand it & to check your work
4. Share!

Next Time



Designed, written, and performed by

Jacob Neumann

Intro-HoTT.video

@Intro_HoTT

@Intro_HoTT

Intro to
Homotopy Type Theory

Next video:

Coming soon!