Introduction to Dependent Type Theory

Outline

- Refresher on non-dependent type theory
- Motivation for dependent type theory
- Formulation of a very simple type theory

- Me trying to convince you that it's related to programming, with words

- Me trying to convince you that it's related to programming, with an underprepared demo

Non-dependent type theory

- Starting with propositional logic
  - → atoms, connectives, LEM
  - → principled presentation: natural deduction

$\Gamma \vdash A$ "proposition A holds in context Γ"

"context" ↖ Γ $\vdash$ A ↖ proposition

  - ~ list of propositions assumed to hold

  - → technically also $\Gamma \vdash A\ \mathit{prop}$ for "`A` denotes a proposition in context Γ", but often left implicit in treatments of non-dependent type theories

Connective rules

→ "formation" – when does a symbol "represent" a proposition? (omitted)

"introduction" – when do we know that a proposition holds?

"elimination" – what's a "natural" way to use a proposition?

Conjunction

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge I$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge E_1 \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge E_2$$

Disjunction

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee I_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee I_2$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee E$$

True

$$\frac{}{\Gamma \vdash \top}\ \top I$$

False

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot E$$

Connective rules

Implication

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B} \to I$$

$$\frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B} \to E$$

Negation

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A} \neg I$$

$$\frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot} \neg E$$

Odd one out

Excluded middle

$$\frac{}{\Gamma \vdash A \vee \neg A}\ \text{LEM}$$

→ doesn't eliminate anything

→ introduces a disjunction and/or negation, but we already have "natural" rules for those

→ let's ignore it for now

- Flavors of logic without LEM are called "constructive" or "intuitionistic"

- Has the right vibe for everyday programming: *waves hands*

  conjunctions ~ tuples
  disjunctions ~ discriminated unions
  implications ~ functions

  ⇒ can we make this precise?

From propositions to types

- When constructing a proof tree, we can build up a string of "tags" to encode the proof
- Such strings are called programs / terms
- Instead of just "a proposition A holds", we may be more specific and say "a program p proves A"
- We don't regard propositions as contentless true/false statements, but as a collection of programs proving a given proposition

→ judgments $\Gamma \vdash t : A$ "term t has type A in context Γ"

- Γ: context ~ list of variables $x:B$
- t: term
- A: type