Types and Curry Howard

Watt Seng Joe

September 9, 2020

What more can we do with types?

Formalize mathematics, ie encode theorems and proofs in the computer.

Why?

So that we can use computers to help us prove theorems and verify the correctness of our programs.
This actually forms the foundation for interactive and automated theorem proving.

How?

We want to find a suitable programming language for encoding mathematics.
We'll see that the typed lambda calculus as found in Haskell forms a suitable foundation.

Putting the "formal" in formal proof

Proofs ala structural proof theory

* Proofs as objects with formally defined structure.
* These are built up using inference rules that tell us what we're allowed to conclude given some assumptions.

Inference rules in Natural Deduction

$$\frac{\Gamma_1 \vdash \text{Hypo1} \qquad \Gamma_2 \vdash \text{Hypo2} \qquad \ldots}{\Gamma_1 \cup \Gamma_2 \cup \cdots \vdash \text{Conclusion}}$$

Γ_i = context, ie a finite set of assumptions that we have available to us.

Simplified notation

Often we simplify the notation to drop the Γ_i.

$$\frac{\text{Hypo1} \qquad \text{Hypo2} \qquad \ldots}{\text{Conclusion}}$$

Natural Deduction for propositional logic

Conjunction

$$\frac{A \qquad B}{A \wedge B}\ \wedge\text{-intro}$$

$$\frac{A \wedge B}{A}\ \wedge\text{-elim left}$$

$$\frac{A \wedge B}{B}\ \wedge\text{-elim right}$$

Disjunction

$$\frac{A}{A \vee B}\ \vee\text{-intro left}$$

$$\frac{B}{A \vee B}\ \vee\text{-intro right}$$

$$\frac{A \vee B \qquad A \to C \qquad B \to C}{C}\ \vee\text{-elim}$$

Natural Deduction for propositional logic

Implication

$$\frac{\begin{array}{c}[A] \\ B\end{array}}{A \to B} \to\text{-intro}$$

$$\frac{A \to B \qquad A}{B} \to\text{-elim}$$

Bi-implication

$$\frac{A \to B \qquad B \to A}{A \leftrightarrow B} \leftrightarrow\text{-intro}$$

$$\frac{A \leftrightarrow B}{A \to B} \leftrightarrow\text{-elim left}$$

$$\frac{A \leftrightarrow B}{B \to A} \leftrightarrow\text{-elim right}$$

Curry Howard, aka Propositions as Types

The big idea

$$\overbrace{1+2}^{\textit{Term}} \;:\; \overbrace{\mathsf{Int}}^{\textit{Type}}$$

$$\underbrace{\mathsf{hP}}_{\textit{Proof}} \;:\; \underbrace{P}_{\textit{Proposition}}$$

* Proving a proposition $\Leftrightarrow$ constructing a term of the corresponding type
* Checking a proof for correctness $\Leftrightarrow$ type checking

Some typing rules

Function type
($\cong$ Implication)

* Intro

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash (\lambda x,\ e) : \sigma \to \tau}$$

* Elim

$$\frac{\Gamma \vdash e_1 : \sigma \to \tau \qquad \Gamma \vdash e_2 : \sigma}{\Gamma \vdash e_1\ e_2 : \tau}$$

Product/Pair type
($\cong$ conjunction)

* Intro

$$\frac{\Gamma \vdash a : \sigma \qquad \Gamma \vdash b : \tau}{\Gamma \vdash (a,\ b) : \sigma \wedge \tau}$$

* Elim left

$$\frac{\Gamma \vdash (a,\ b) : \sigma \wedge \tau}{\Gamma \vdash a : \sigma}$$

* Elim right

$$\frac{\Gamma \vdash (a,\ b) : \sigma \wedge \tau}{\Gamma \vdash b : \tau}$$

Some typing rules

Sum/tagged union type ($\cong$ disjunction)

* Intro left

$$\frac{\Gamma \vdash a : \sigma}{\Gamma \vdash \mathsf{inl}\ a : \sigma \vee \tau}$$

* Intro right

$$\frac{\Gamma \vdash b : \tau}{\Gamma \vdash \mathsf{inr}\ b : \sigma \vee \tau}$$

* Elim

$$\frac{\Gamma \vdash s : \sigma \vee \tau \qquad \Gamma, a : \sigma \vdash c : \eta \qquad \Gamma, b : \tau \vdash d : \eta}{\Gamma \vdash (\mathsf{case}\ s\ \mathsf{of}\ (\mathsf{inl}\ a) \Rightarrow \mathsf{c} \mid (\mathsf{inr}\ b) \Rightarrow \mathsf{d}) : \eta}$$

Key point

The pattern for elim must be complete, ie you must consider both possible cases.

Example proofs using Curry Howard

$P \to P$

$\lambda\,(hP : P),\ hP$ (ie identity function)

$P \wedge Q \to Q \wedge P$

$\lambda\,(h : P \wedge Q),\ \text{let}\ (hP,\ hQ) = h\ \text{in}\ (hQ,\ hP)$

$(P \vee Q) \to (Q \vee P)$

$$\lambda\,(h : P \vee Q), \text{case}\ h\ \text{of}$$
$$(\text{inl}\ hP) \Rightarrow \text{inr}\ hP$$
$$(\text{inr}\ hQ) \Rightarrow \text{inl}\ hQ$$

Interpreting Curry & Uncurry

Curry : $(P \wedge Q \to R) \to (P \to Q \to R)$

Uncurry : $(P \to Q \to R) \to (P \wedge Q \to R)$

Propositional logic $\rightarrow$ First order logic

Summary

* We can essentially write proofs of theorems as typed functional programs!!!
* Our proofs are functions such that:
  - Input: Proofs of whatever hypotheses that we have.
  - Output: A proof of the desired conclusion.

How to model $\exists$ and $\forall$?

* Actually, can also encode negation and other rules from propositional logic in the simply typed lambda calculus.
* But we need $\exists$ and $\forall$ to encode real mathematical theorems!
* Dependent types offer an elegant solution to this.