Simply typed lambda calculus

- Simply typed lambda calculus is a formal system.
- We start with some "atomic" types
- We can make new types out of "type constructors" known as introduction rules
- Usually we only have function types, but we can have more...

Extending typed lambda calculus

When adding a new type we must write down rules to define how it will behave. Usually these are sorted into 4 kinds of rules:

- Introduction rules (how to make the type)
- Constructors (how to make terms of the type)
- Eliminators (how to break terms of the type)
- Computation rules (how a function coming out of the type computes)

Note: Computation rules can usually be derived from the other rules, and therefore can be omitted.

Simply typed lambda calculus

- Simply typed lambda calculus is a formal system.
- We start with some "atomic" types
- We can make new types out of "type constructors" known as introduction rules
- Usually we only have function types, but we can have more...

When adding a new type we must write down rules to define how it will behave. Usually these are sorted into 4 kinds of rules:

- Introduction rules (how to make the type)
- Constructors (how to make terms of the type)
- Eliminators (how to break terms of the type)
- Computation rules (how a function coming out of the type computes)

Note: Computation rules can usually be derived from the other rules, and therefore can be omitted.

Product types

Introduction

$$\frac{\Gamma \vdash A \ \text{Type} \qquad \Gamma \vdash B \ \text{Type}}{\Gamma \vdash A \times B \ \text{Type}}$$

Constructors

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B}$$

Eliminators

$$\frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \mathsf{fst}(t) : A} \qquad \frac{\Gamma \vdash t : A \times B}{\Gamma \vdash \mathsf{snd}(t) : B}$$

Computation rules

$$(\mathsf{fst}(t), \mathsf{snd}(t)) \equiv t$$

Sum types

Introduction

$$\frac{\Gamma \vdash A \text{ Type} \qquad \Gamma \vdash B \text{ Type}}{\Gamma \vdash A + B \text{ Type}}$$

Constructors

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash \mathrm{inl}(a) : A + B} \qquad \frac{\Gamma \vdash b : B}{\Gamma \vdash \mathrm{inr}(b) : A + B}$$

Eliminators

$$\frac{\Gamma \vdash f : A \to C \qquad \Gamma \vdash g : B \to C}{\Gamma \vdash \mathrm{ind}_{A+B}(f, g) : A + B \to C}$$

Motivation

- Programming languages
  - Haskell
  - ML
  - basically every other typed functional programming language

Mathematical motivation - Curry-Howard correspondance

There is a correspondance between propositional logic and type theory. Types are propositions, and terms are proofs.

Propositional logic	**Type theory**
proposition A	A Type
proof of A	term of A
and $A \wedge B$	product type $A \times B$
or $A \vee B$	sum type $A + B$
implies $A \implies B$	function type $A \to B$
true	unit type $\mathbf{1}$
false	empty type $\mathbf{0}$
not A	$A \to \mathbf{0}$

This is the begining of using type theory to encode mathematics. This is how proof assistants work.

What are dependent types?

- Functions allow terms to depend on other terms
- Polymorphism allows types to depend on other types
- Terms already depend on types
- Dependent types allow types to depend on terms

What problems can dependent types solve?

- Encoding hard to encode data types such as lists (or vectors) of fixed length.
- It is equivalent to first-order logic in some suitable sense.

Pi types

What if the target of a function type could change depending on the input?

Sigma types

Some times product types are not enough. Especially when we need a family.

(Dependent) Curry-Howard

Propositional logic	**Type theory**
$\forall a \in A, P(a)$	pi type $\prod_{(a:A)} P(a)$
$\exists a \in A, P(a)$	sigma type $\sum_{(a:A)} P(a)$
proposition A	A Type
proof of A	term of A
and $A \wedge B$	product type $A \times B$
or $A \vee B$	sum type $A + B$
implies $A \implies B$	function type $A \to B$
true	unit type $\mathbf{1}$
false	empty type $\mathbf{0}$
not A	$A \to \mathbf{0}$

How can we model type theories?

Answer:

Categorical semantics.

This allows us to use category theory to reason about the metatheory of our type theory.

But theres more...

When modelling "type theories" in mathematics it was found that there is really a two way correspondance.

$$\text{Type theory} \rightleftharpoons \text{Category theory}$$

Type theory can be used to reason about a category. Lots of people have investigated this, notably Topos theorists.