Dependent types

Ali Caglayan

University of Bath

February 18, 2019

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 formation 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:

- Formation 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 terms of the type compute)

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

Product types

Formation

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

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

Formation

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

Constructors

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

Eliminators

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

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. (Dependent Curry-Howard)
- Generalises polymorphism, GADT, inductive types etc.

Pi types

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

Introduction

$$\frac{\Gamma \vdash A \ \text{Text} \qquad \Gamma, x : A \vdash B \ \text{Type}}{\Gamma \vdash \prod_{(x:A)} B \ \text{Type}}$$

Constructors

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda(x : A).b : \prod_{(x:A)} B}$$

Eliminators and Computation rules

$$\frac{\Gamma \vdash f : \prod_{(x:A)} B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B[a/x]}$$

$$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x : A).b)(a) \equiv b[a/x] : B[a/x]}$$

Sigma types

When making the type of dates, we could write

$$\text{Days} \times \text{Months} \times \text{Years}$$

However this would give us completely nonsense dates such as 31/02/2019.
Some times product types are not enough. In this case we need **sigma types**.

$$\sum_{(y:\text{Years})} \sum_{(m:\text{Months})} \text{Days}(y, m)$$

A term of which would look like $(2019; (01; 31))$ (the order doesn't really matter). We might call such a term a dependent pair. Note: Days is a family of types, i.e. a type depending on the given year and month.

- Sigma types generalise product types.

(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.

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

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

Models of dependent types (all equivalent)

- Display map categories (objects are contexts, pick your "display maps") (lots of examples but harder to work with)
- Comprehension categories [Jacobs 1993] (category equipped with a nice fibration giving category of types in a context)
- Category with attributes [Cartmell 1978] (comprehension with discrete fibration giving set of types in a context)
- Contextual categories [Cartmell 1986, Streicher 1991] (CwA with "length function", awkward but Streicher uses them to "prove" initiality of dependent type theory.
- Category with families [Dybjer 1995] (construction giving a set of terms for a given type)
- Natural models [Awodey 2018] (essentially CwF but phrased in terms of nice universal properties making proofs easier)

However we have a problem...

Fixing associativity

Comprehension categories, for example, don't give strict substiution (which is modelled by pullbacks). Only split fibrations give strict composition of pullbacks. There are 3 ways to split comprehension categories that I know of:

Splitting comprehension categories

- Right ajoint splitting [Bénabou, Hoffman]
- Left adjoint splitting [Lumsdaine, Warren]
- Splitting via universe [Voevodsky]

As of 2019 these have all been written out explicitly.

Models where this is strict tend have very few natural examples.
Models where this isn't tend to have to be later "split".
Awodey's natural models have strictly composing pullbacks, and seem to be relatively easy to work with.