

Dependent Types Made Difficult

Mark Hopkins
@antiselfdual
mjhopkins.github.io

**What is the categorical semantics of
dependent type theory?**



PHASE 1 PHASE 2 PHASE 3

Categorical
semantics for
dependent type theory



Profit

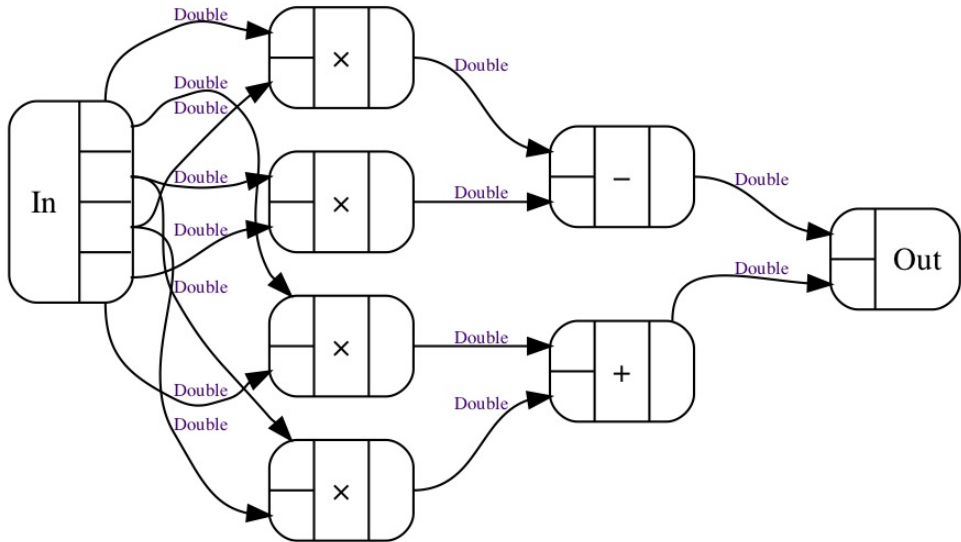


Compiling to Categories (Elliott, 2017)

Take in Haskell source and spit out . . .

Compiling to Categories (Elliott, 2017)

- computation graphs
 - diagrams
 - circuit descriptions (VHDL, Verilog)
- linear maps
- automatic differentiation
- incremental computation
- interval analysis
- Kleisli category for any monad e.g. probabilistic programming
- graphics (GLSL)
- syntax
- products of the above
- tons more . . . see <https://github.com/conal/concat>



How does this work??

The simply-typed lambda calculus is the *internal language* of *Cartesian closed categories*.

which means

We can interpret the lambda calculus in any CCC.

Cartesian closed categories

A Cartesian closed category is a category with function objects.

$$\mathcal{C}(A \times B, C) \cong \mathcal{C}(A, C^B)$$

In Haskell, this isomorphism is called “curry”.

```
curry :: ((a,b) -> c) -> a -> b -> c
```

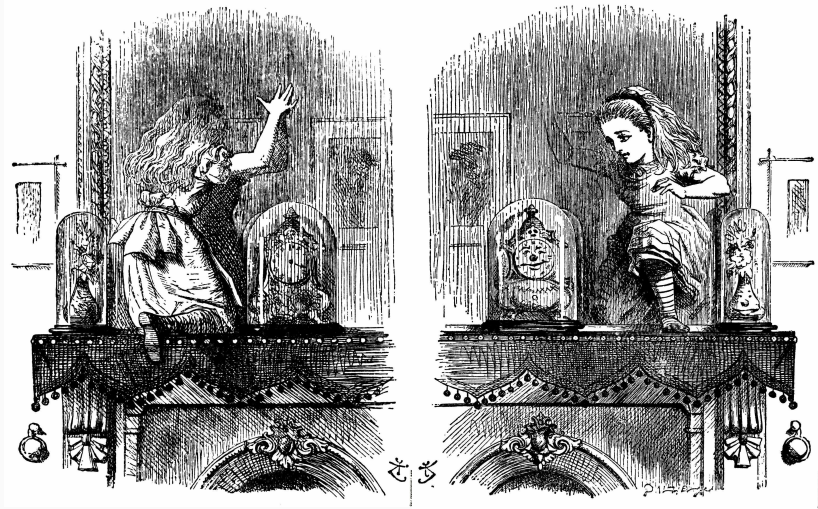
```
curry f a b = f (a, b)
```

```
uncurry :: (a -> b -> c) -> (a, b) -> c
```

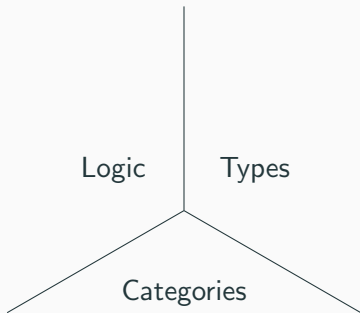
```
uncurry f (a, b) = f a b
```

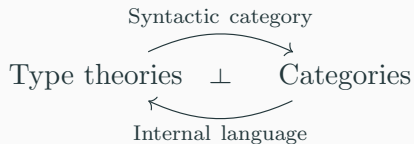
Categorical semantics

Curry-Howard correspondence



Curry-Howard-Lambek correspondence





Advantages

- More easily prove properties about a type theory using CT
- CT proofs using the internal language can be easier
- Can use the internal language to define internal structures
- The CT can illuminate the TT, and vice versa
- The internal language is valid in every model

Adjunctions

Adjunctions

$$F \dashv U : \mathcal{C} \rightarrow \mathcal{D}$$

$$\begin{array}{ccc} & F & \\ \mathcal{C} & \xrightarrow{\quad} & \mathcal{D} \\ & U & \end{array} \quad \begin{array}{c} \perp \\ \leftarrow \end{array}$$

$$\mathcal{D}(Fc, d) \cong \mathcal{C}(c, Ud)$$

$$\frac{Fc \rightarrow d \text{ in } \mathcal{D}}{c \rightarrow Ud \text{ in } \mathcal{C}}$$

Example: free monoid

The free monoid on an alphabet Λ is the set Λ^* of words in Λ .

$$\frac{\Lambda^* \rightarrow m \text{ in } \mathbf{Mon}}{\Lambda \rightarrow m \text{ in } \mathbf{Set}}$$

Why?

A monoid homomorphism has to respect multiplication, so

$$f(abcd \cdots) = f(a)f(b)f(c) \cdots$$

Example: currying

The functor $- \times b$ is left adjoint to $(-)^b$.

$$\frac{a \times b \rightarrow c \text{ in } \mathbf{Type}}{a \rightarrow c^b \text{ in } \mathbf{Type}}$$

`curry :: ((a,b) -> c) -> a -> b -> c`

`curry f a b = f (a, b)`

`uncurry :: (a -> b -> c) -> (a, b) -> c`

`uncurry f (a, b) = f a b`

Example: quantifiers $\exists_y \dashv w_y \dashv \forall_y$

$$w_y : \text{Form}(\bar{x}) \rightarrow \text{Form}(\bar{x}, y)$$

$$\phi(\bar{x}) \mapsto \phi(\bar{x})$$

$$\frac{\bar{x}, y \vdash w_y \phi(\bar{x}) \Rightarrow \psi(\bar{x}, y)}{\bar{x} \vdash \phi(\bar{x}) \Rightarrow \forall_y \psi(\bar{x}, y)}$$

$$\frac{\bar{x} \vdash \exists_y \psi(\bar{x}, y) \Rightarrow \phi(\bar{x})}{\bar{x}, y \vdash \psi(\bar{x}, y) \Rightarrow w_y \phi(\bar{x})}$$

- Categorical semantics for STLC in CCCs
- Lots of CCCs
- GHC Core's System FC can (mostly) be converted to STLC
- GHC plugin, rewrite rules
- Convert Haskell src to VHDL, diagrams, etc

There's more to type theory than STLC!

- polymorphic types
- existential types
- universal types
- type classes
- union and intersection types
- quotient types
- dependent types
- refinement types
- homotopy type theory
- . . .

Dependent types

Dependent types

A dependent type is one that contains free variables.

$$\tau : \text{Type} \quad x : \tau \vdash P(x) : \text{Type}$$

It *depends on* terms.

$n : \text{Nat} \vdash \text{IsEven } n : \text{Type}$

```
data IsEven (n : Nat) : Type where  
  ZEven  : IsEven 0  
  SSEven : IsEven n -> IsEven (n + 2)
```

Dependent types

$$n : \text{Nat}, a : \text{Type} \vdash \text{Vect}_n(a) : \text{Type}$$

```
data Vect (n : Nat) (a : Type) : Type where
  Nil  : Vect 0 a
  (::) : a -> Vect n a -> Vect (n + 1) a
```

Dependent product type

`replicate : (n : Nat) -> a -> Vect n a`

$$\begin{aligned}\text{replicate} &: \prod_{a:\text{Type}} \prod_{n:\mathbb{N}} \prod_{x:a} \text{Vect}_n(a) \\ &= \prod_{a:\text{Type}} \prod_{n:\mathbb{N}} a \rightarrow \text{Vect}_n(a)\end{aligned}$$

$$a \rightarrow b \equiv \prod_{_ : a} b$$

Dependent sum type

`evenLenLists = (l : List Int ** m : Nat ** length l = 2 * m)`

$$\text{evenLenLists} = \sum_{l:\text{List}(a)} \sum_{m:\mathbb{N}} \text{length}(l) = 2m$$

$$(a, b) \equiv \sum_{_ : a} b$$

Dependent types are the internal language of *locally Cartesian closed categories*.

Semantics

Objects: closed types

Arrows: terms in context

also arrows: functions

$$f : a \rightarrow b$$

$$[[f]] : [[a]] \rightarrow [[b]]$$

$\text{IsEven} : \mathbb{N} \rightarrow \mathbf{Set}$

$\text{IsEven } 0 = \{\text{zeroEven}\}$

$\text{IsEven } 1 = \{\}$

$\text{IsEven } 2 = \{\text{twoEven}\}$

$\text{IsEven } 3 = \{\}$

$\text{IsEven } 4 = \{\text{fourEven}\}$

\vdots

CustomerOf : Bank \rightarrow **Set**

CustomerOf ANZ = {Alice, Albert, Anne-Marie}

CustomerOf CBA = {Calvin, Colin}

CustomerOf NAB = {Nathan}

CustomerOf Westpac = {Wendy, Will}

Thinking backwards

$p : \text{IsEven } _ \rightarrow \mathbb{N}$

$p \text{ zeroEven} = 0$

$p \text{ twoEven} = 2$

$p \text{ fourEven} = 4$

\vdots

$p : \text{CustomerOf } _ \rightarrow \text{Bank}$

$p \text{ Alice} = \text{ANZ}$

$p \text{ Albert} = \text{ANZ}$

$p \text{ Anne-Marie} = \text{ANZ}$

$p \text{ Calvin} = \text{CBA}$

$p \text{ Colin} = \text{CBA}$

$p \text{ Nathan} = \text{NAB}$

$p \text{ Wendy} = \text{Westpac}$

$p \text{ Will} = \text{Westpac}$

$$p : \text{IsEven } _ \rightarrow \mathbb{N}$$

$$p \text{ zeroEven} = 0$$

$$p \text{ twoEven} = 2$$

$$p \text{ fourEven} = 4$$

$$\vdots$$

Thinking backwards

$p : \text{CustomerOf } _ \rightarrow \text{Bank}$

$p \text{ Alice} = \text{ANZ}$

$p \text{ Albert} = \text{ANZ}$

$p \text{ Anne-Marie} = \text{ANZ}$

$p \text{ Calvin} = \text{CBA}$

$p \text{ Colin} = \text{CBA}$

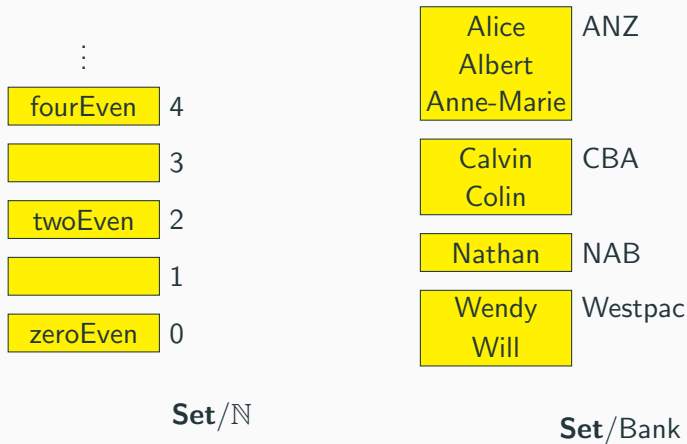
$p \text{ Nathan} = \text{NAB}$

$p \text{ Wendy} = \text{Westpac}$

$p \text{ Will} = \text{Westpac}$

\vdots

Semantics for dependent types



Over-categories

$$X \in \mathcal{C}, \mathcal{C}/X$$

Objects:

$$\begin{array}{c} A \\ \downarrow p \\ X \end{array}$$

fibre $(A)_x = p^{-1}x$ in **Set**

Arrows:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ & \searrow p & \swarrow q \\ & X & \end{array}$$

section $s : X \rightarrow A$
such that $p \circ s = \text{id}$

Over-categories

Alice
Albert
Anne-Marie

ANZ

Calvin
Colin

CBA

Nathan

NAB

Wendy
Will

Westpac

Artarmon
Annandale
Auburn

ANZ

Clovelly
Curl Curl

CBA

Naremburn
Nth Bondi

NAB

Wahroonga
Woollahra

Westpac

$$\text{fibre } (A)_x = p^{-1}x$$

section $s : X \rightarrow A$ such that $p \circ s = \text{id}$

$\text{bankOf} : \text{Branch} \rightarrow \text{Bank}$

$b : \text{Bank} \vdash \text{CustomerOf } b : \text{Type}$

$br : \text{Branch} \vdash \text{CustomerOf}(\text{bankOf } br) : \text{Type}$

Alice Albert	Artarmon
	Annandale
Anne-Marie	Auburn
Calvin Colin	Clovelly
	Curl Curl
	Naremburn
Nathan	Nth Bondi
Will	Wahroonga
Wendy	Woollahra

⋮	
	“hello”
twoEven	“no”
eightEven	“icecream”
	“submarine”
⋮	

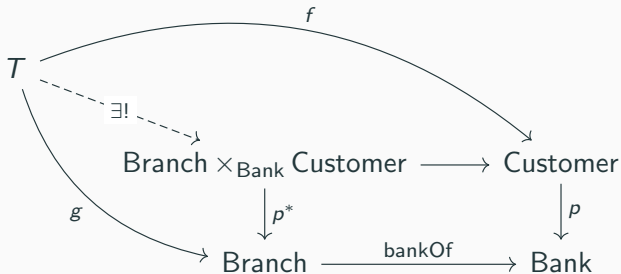
$s : \text{String} \vdash \text{IsEven}(\text{length } s) : \text{Type}$

Substitution = pullback aka change of base

$\text{bankOf} : \text{Branch} \rightarrow \text{Bank}$

$\text{bankOf}^* : \mathbf{Set} / \text{Bank} \rightarrow \mathbf{Set} / \text{Branch}$

Substitution = pullback aka change of base



$$\begin{aligned}\text{Branch} \times_{\text{Bank}} \text{Customer} &= \text{bankOf}^* \text{Customer} \\ &= \{(br, c) \mid \text{bankOf}(br) = p(c)\}\end{aligned}$$

Dependent sum

$$f : X \rightarrow Y$$

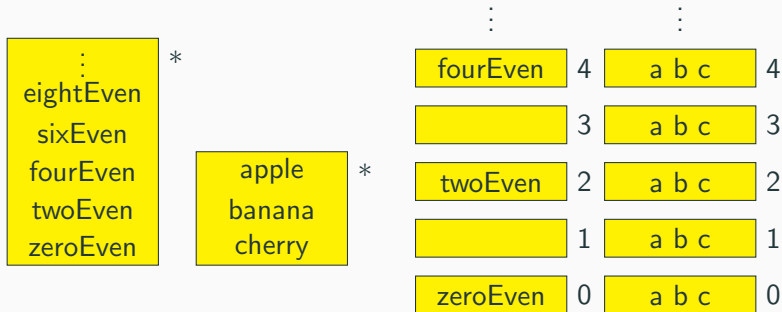
$$\sum_f : \mathbf{Set} / X \rightarrow \mathbf{Set} / Y$$

$$\begin{array}{c} A \\ \downarrow p \\ X \\ \downarrow f \\ Y \end{array}$$

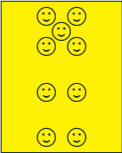









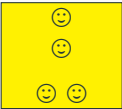







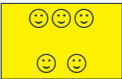





$$\sum_f \dashv f^*$$

$$\sum_{x:X} := \sum_{!_X} \quad !_X : X \rightarrow *$$

$$\mathbf{Set} / * (\sum_f p, q) \cong \mathbf{Set} / X(p, f^* q)$$



Σ_{bankOf} : **Set** / Branch \rightarrow **Set** / Bank

		ANZ			Artarmon
					Annandale
					Auburn
		CBA			Clovelly
		NAB			Naremburn
		Westpac			Wahroonga
					Woollahra

A category is **locally Cartesian closed**
if f^* also has a right adjoint
for all arrows f .

Lemma

In this case all the slice categories \mathcal{C}/X are Cartesian closed.



PHASE 1 PHASE 2 PHASE 3

Categorical
semantics for
dependent type theory



Profit



- Categorical semantics for DTs in LCCCs
- Lots of LCCCs
- Something something Agda, Idris, Lean, ...
- **Profit**

Set is locally Cartesian closed.

Dependent product

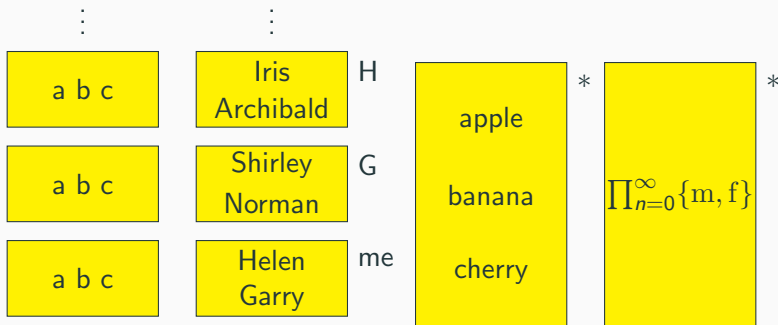
$$f : X \rightarrow Y$$

$$\prod_f : \mathbf{Set} / X \rightarrow \mathbf{Set} / Y$$

$$f^* \dashv \prod_f$$










$$\prod_{x:X} := \prod_{!_X} \quad !_X : X \rightarrow *$$

$$\mathbf{Set} / X(f^* p, q) \cong \mathbf{Set} / * (p, \prod_f q)$$



$$\frac{p : \text{Person} \vdash \text{Fruit} \rightarrow \text{ParentOf}(p)}{\vdash \text{Fruit} \rightarrow \prod_{p:\text{Person}} \text{ParentOf}(p)}$$

$\Pi_{\text{bankOf}} : \mathbf{Set} / \text{Branch} \rightarrow \mathbf{Set} / \text{Bank}$

A A A		Artarmon		
A A A		Annandale		
A A A		Auburn		
C C		Clovelly	Alice Albert Anne-M	...
C C		Curl Curl		
N		Naremburn	Calvin Colin	...
N		Nth Bondi	Nathan	...
W W		Wahroonga	Wendy Will	...
W W		Woollahra		

$$(\Pi_{\text{bankOf}} \text{Staff})_{\text{ANZ}} = \{s : (\text{Branch})_{\text{ANZ}} \rightarrow \text{Staff} \mid \forall br. \text{branchOf}(s(br)) = br\}$$

$c : \text{Customer} \vdash \text{IsEven}(\text{length}(\text{firstName}(c)))$

- substitute $s := \text{firstName}(c)$ into $\text{IsEven}(\text{length}(s))$
- substitute $n := \text{length}(\text{firstName}(c))$ into $\text{IsEven } n$

- display map categories
- contextual categories
- categories with families
- categories with attributes
- Awodey's “natural model”

R. A. G. Seely, *Locally cartesian closed categories and type theory*

Martin Hofman, *Syntax and semantics of dependent types*

Alexandre Buisse, *Categorical models of dependent type theory*

Awodey, *Category Theory*, 7.29 and 9.7

MacLane and Moerdijk, *Sheaves in Geometry and Logic*, IV.7

The nLab, ncatlab.org/nlab

