Combining Bunched Type Theory with Dependent Types

Mitchell Riley

Wesleyan University

jww. Dan Licata

Wesleyan University

1st February 2022

Introduction

- In Computer Science, type theories are often created first and then categorical semantics are devised for them.
- Today, an example of the backwards direction: we have a categorical structure in mind, and want a type theory.
- Goal: a practical type theory for working with ‘parameterised spectra’ which form a category that is both locally cartesian closed, and also monoidal closed.

Curry-Howard-Lambek Correspondences

The Simply Typed λ-Calculus

$$\frac{}{\Gamma, x : A, \Gamma' \vdash x : A}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B} \qquad \frac{\begin{array}{c}\Gamma \vdash p : A \times B \\ \Gamma, x : A, y : B \vdash c : C\end{array}}{\Gamma \vdash \mathsf{let}\,(x, y) = p\,\mathsf{in}\,c : C}$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda x.b : A \to B} \qquad \frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}$$

These rules (and some omitted equations) present the *free cartesian closed category* on a set of objects.

Eg: Symmetry

Proposition

$\mathsf{sym} : A \times B \to B \times A$

Proof.

To define $\mathsf{sym} : A \times B \to B \times A$, suppose we have $p : A \times B$. Then splitting allows us to assume $p \equiv (x, y)$ and we then have (y, x).

$$\mathsf{sym} :\equiv \lambda p.\mathsf{let}\ (x, y) = p\ \mathsf{in}\ (y, x)$$

$\square$

Interpretation

- Each type A is interpreted as an object $[\![A]\!]$
- Each term $x_1 : A_1, \dots x_n : A_n \vdash b : B$ is interpreted as a morphism
$$[\![b]\!] : [\![A_1]\!] \times \cdots \times [\![A_n]\!] \to [\![B]\!]$$

Curry-Howard-Lambek Correspondence

Type Theory	**Categorical Structure**
Simply Typed Lambda Calculus	Cartesian Closed Categories
Dependent Type Theory with 1, Σ and Extensional $=$	Finitely Complete Categories
... and Π	Locally Cartesian Closed Categories
... and 0, $+$, Prop, Axioms	Elementary Topos
Simply Typed Lambda Calculus	Cartesian Closed Categories
Multiplicative Intuitionistic Linear Logic	Monoidal Closed Categories
Classical Linear Logic	$*$-Autonomous Categories
This Type Theory	LCCC $+$ Monoidal Closed

Dependent Type Theory

Type theory can be made more expressive by allowing types to *depend* on terms: $\Gamma \vdash A$ type.

Example

The set of days in a month depends on which month we are talking about: $x : \mathsf{Month} \vdash \mathsf{DayOf}(x)$ type

Example

Each point of a differentiable manifold has a tangent space: $x : M \vdash T_x M$ type

Dependent Type Theory

The product type can be generalised to *dependent* pairs:

$$\Sigma\text{-}\textsc{form}\ \frac{\Gamma \vdash A\ \mathsf{type} \qquad \Gamma, x : A \vdash B(x)\ \mathsf{type}}{\Gamma \vdash (x : A) \times B(x)\ \mathsf{type}}$$

$$\Sigma\text{-}\textsc{intro}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B(a)}{\Gamma \vdash (a, b) : (x : A) \times B(x)}$$

$$\ldots$$

Example

The dependent pair type $(x : \mathsf{Month}) \times \mathsf{DayOf}(x)$ is type of all days in the year.

The dependent pair type $(x : M) \times T_x M$ is the tangent bundle TM.

Dependent Type Theory

A type of equalities is expressible:

$$=\text{-}\textsc{form}\ \frac{\Gamma \vdash a : A \qquad \Gamma \vdash a' : A}{\Gamma \vdash a = a' \ \mathsf{type}}$$

$$=\text{-}\textsc{intro}\ \frac{\Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_a : a = a}$$

$$\ldots$$

Interpretation

- A type $\Gamma \vdash A$ type is interpreted as an object $[\![A]\!]$ of the slice $\mathcal{C}_{/[\![\Gamma]\!]}$
- The context $\Gamma, x : A$ is interpreted as the object $[\![A]\!]$ of $\mathcal{C}$.
- A term $\Gamma \vdash a : A$ is interpreted as a morphism $[\![a]\!] : \mathsf{id}_{[\![\Gamma]\!]} \to [\![A]\!]$ in $\mathcal{C}_{/[\![\Gamma]\!]}$
- The type $(x : A) \times B$ is interpreted as the composite

$$[\![B]\!] \to [\![A]\!] \to [\![\Gamma]\!]$$

  in $\mathcal{C}_{/[\![\Gamma]\!]}$.
- The type $a = a'$ is interpreted as the diagonal

$$[\![A]\!] \to [\![A]\!] \times_{[\![\Gamma]\!]} [\![A]\!]$$

  pulled back along the map $[\![\Gamma]\!] \to [\![A]\!] \times_{[\![\Gamma]\!]} [\![A]\!]$ induced by $[\![a]\!]$ and $[\![a']\!]$

Curry-Howard-Lambek Correspondence

Type Theory	**Categorical Structure**
Simply Typed Lambda Calculus	Cartesian Closed Categories
Dependent Type Theory with 1, Σ and Extensional $=$	Finitely Complete Categories
... and Π	Locally Cartesian Closed Categories
... and 0, $+$, Prop, Axioms	Elementary Topos
Simply Typed Lambda Calculus	Cartesian Closed Categories
Multiplicative Intuitionistic Linear Logic	Monoidal Closed Categories
Classical Linear Logic	$*$-Autonomous Categories
This Type Theory	LCCC + Monoidal Closed

Dependent Type Theory

Similarly for *dependent* functions:

$$\Pi\text{-FORM}\ \frac{\Gamma \vdash A \ \mathsf{type} \qquad \Gamma, x : A \vdash B(x)\ \mathsf{type}}{\Gamma \vdash (x : A) \to B(x)\ \mathsf{type}}$$

$$\Pi\text{-ELIM}\ \frac{\Gamma \vdash f : (x : A) \to B(x) \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B(a)}$$

$$\dots$$

Example

The dependent function type $(x : \mathsf{Month}) \to \mathsf{DayOf}(x)$ is a choice of one day from each month.

The dependent function type $(x : M) \to T_x M$ is a vector field.
(sort of, one would need to think carefully about continuity)

Symmetry Again

Proposition

$\mathsf{sym}_{X,Y} : X \times Y \to Y \times X$ *is an equivalence.*

('f an equivalence' means that there are g and g' so that pointwise $f \circ g = \mathsf{id}$ and $g' \circ f = \mathsf{id}$.)

Proof.

Its inverse is $\mathsf{sym}_{Y,X}$. To prove

$$\prod_{(p:A\times B)} \mathsf{sym}_{Y,X}(\mathsf{sym}_{X,Y}(p)) = p,$$

use splitting: the goal reduces to $(x, y) = (x, y)$ for which we have reflexivity. $\square$

Interpretation

Weakening is interpreted by a pullback:

$$[\![A]\!]^* : \mathcal{C}_{/[\![\Gamma]\!]} \to \mathcal{C}_{/[\![\Gamma, x:A]\!]}$$

Σ and Π are interpreted in the category as the left- and right-adjoint to weakening.

$$\begin{tikzcd} \mathcal{C}_{/[\![\Gamma]\!]} \arrow[r] & \mathcal{C}_{/[\![\Gamma, x:A]\!]} \arrow[l, bend right, "\Sigma"', "\perp"] \arrow[l, bend left, "\Pi", "\perp"'] \end{tikzcd}$$

Curry-Howard-Lambek Correspondence

Type Theory	Categorical Structure
Simply Typed Lambda Calculus	Cartesian Closed Categories
Dependent Type Theory with 1, Σ and Extensional $=$	Finitely Complete Categories
... and Π	Locally Cartesian Closed Categories
... and 0, $+$, Prop, Axioms	Elementary Topos
Simply Typed Lambda Calculus	Cartesian Closed Categories
Multiplicative Intuitionistic Linear Logic	Monoidal Closed Categories
Classical Linear Logic	$*$-Autonomous Categories
This Type Theory	LCCC + Monoidal Closed

Multiplicative Intuitionistic Linear Logic

$$\frac{}{x : A \vdash x : A}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma' \vdash b : B}{\Gamma, \Gamma' \vdash (a \otimes b) : A \otimes B} \qquad \frac{\begin{array}{c}\Gamma' \vdash p : A \otimes B \\ \Gamma, x : A, y : B \vdash c : C\end{array}}{\Gamma, \Gamma' \vdash \mathsf{let}\,(x \otimes y) = p\,\mathsf{in}\,c : C}$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \partial x.b : A \multimap B} \qquad \frac{\Gamma \vdash f : A \multimap B \qquad \Gamma' \vdash a : A}{\Gamma, \Gamma' \vdash f(a) : B}$$

Eg: Symmetry

Proposition

$\mathsf{sym} : A \otimes B \multimap B \otimes A$

Proof.

Suppose $p : A \otimes B$. Then splitting allows us to assume $p \equiv (x \otimes y)$, and we then have $(y \otimes x)$.

$$\mathsf{sym} :\equiv \partial p.\mathsf{let}\ (x \otimes y) = p\ \mathsf{in}\ (y \otimes x)$$

$\square$

Non-Eg: Diagonal and Projection

We cannot define $\Delta : A \multimap A \otimes A$.

After assuming $x : A$, the term $x \otimes x : A \otimes A$ is not well-formed: only one side of the $\otimes$ is permitted to use x.

We cannot define $\pi_1 : A \otimes B \multimap A$.

After assuming $p : A \otimes B$ and using splitting to obtain $x : A$ and $y : B$, we cannot conclude $x : A$, because $y : B$ is unused.

Interpretation

- Each type A is interpreted as an object $[\![A]\!]$
- Each term $x_1 : A_1, \dots x_n : A_n \vdash b : B$ is interpreted as a morphism

$$[\![b]\!] : [\![A_1]\!] \otimes \cdots \otimes [\![A_n]\!] \to [\![B]\!]$$

Curry-Howard-Lambek Correspondence

Type Theory	**Categorical Structure**
Simply Typed Lambda Calculus	Cartesian Closed Categories
Dependent Type Theory with 1, Σ and Extensional $=$	Finitely Complete Categories
... and Π	Locally Cartesian Closed Categories
... and 0, $+$, Prop, Axioms	Elementary Topos
Simply Typed Lambda Calculus	Cartesian Closed Categories
Multiplicative Intuitionistic Linear Logic	Monoidal Closed Categories
Classical Linear Logic	$*$-Autonomous Categories
This Type Theory	LCCC + Monoidal Closed

Our Setting

The Motivation

Our goal was to use type theory to reason about ‘spectra’, in the sense of stable homotopy theory.

These form a symmetric monoidal closed ∞-category $(\mathrm{Spec}, \mathbb{S}, \otimes, \multimap)$.

Think of the 1-category $(\mathrm{Set}_\bullet, \mathrm{Bool}, \wedge, \to_\bullet)$.

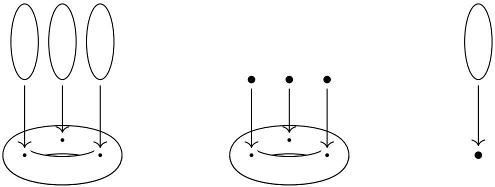
These are models of linear logic.

Families

Definition

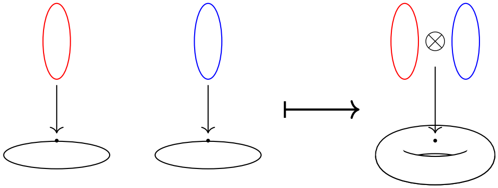
If $\mathcal{C}$ is a category, the category $P\mathcal{C}$ of *parameterised families of* $\mathcal{C}$ has

- Objects given by $(X, \{E_x\}_{x\in X})$ where X is a set and E_x is an object of $\mathcal{C}$ for each $x \in X$.
- Morphisms $(X, \{E_x\}) \to (Y, \{F_y\})$ given by a pair $(f, \{f_x\})$ where $f : X \to Y$ is a function and $f_x : E_x \to F_{f(x)}$ is a morphism of $\mathcal{C}$ for every $x \in X$.



Families

If $\mathcal{C}$ is monoidal closed, then $P\mathcal{C}$ is monoidal closed with the 'external monoidal product' $\overline{\otimes}$.



In favourable conditions, $P\mathcal{C}$ is also locally cartesian closed.
(For example, when $\mathcal{C}$ is LCCC, but in some other unexpected cases too)

Families

$$
\begin{tikzcd}
(P\mathrm{Spec}, \mathbb{S}, \overline{\otimes}, \overline{\multimap}) \arrow[d, ""{name=M}] \\
(\mathcal{S}, 1, \times, \to) \arrow[u, bend left=50, "0", ""{name=L}] \arrow[u, bend right=50, "0"', ""{name=R}]
\arrow[from=L, to=M, phantom, "\dashv"] \arrow[from=M, to=R, phantom, "\dashv"]
\end{tikzcd}
\qquad
\begin{tikzcd}
(P\mathrm{Set}_\bullet, \mathrm{Bool}, \overline{\wedge}, \overline{\to_\bullet}) \arrow[d, ""{name=M}] \\
(\mathrm{Set}, 1, \times, \to) \arrow[u, bend left=50, "0", ""{name=L}] \arrow[u, bend right=50, "0"', ""{name=R}]
\arrow[from=L, to=M, phantom, "\dashv"] \arrow[from=M, to=R, phantom, "\dashv"]
\end{tikzcd}
\qquad
\begin{tikzcd}
(P\overline{M}, I, \overline{\otimes}, \overline{\multimap}) \arrow[d, "\sharp"', ""{name=M}] \\
(\mathrm{Set}, 1, \times, \to) \arrow[u, bend right=50, "q"', ""{name=R}]
\arrow[from=M, to=R, phantom, "\dashv"]
\end{tikzcd}
$$

Linearity and Dependency

Linearity + dependency has been done before, but:

- Indexed type theories (Vákár 2014; Krishnaswami, Pradic, and Benton 2015; Isaev 2021) have semantics in indexed monoidal categories,
- Quantitative type theories (McBride 2016; Atkey 2018; Moon, Eades III, and Orchard 2021; Fu, Kishida, and Selinger 2020) have restricted dependency for Σ and Π,
- Existing dependent 'bunched' type theories (Schöpp 2006; Schöpp and Stark 2004; Cheney 2009; Cheney 2012) require $I = 1$.

Type Theory

The Symmetry Proof We Want

Proposition

$\mathsf{sym} : A \otimes B \simeq B \otimes A$

Proof.

To define $\mathsf{sym} : A \otimes B \to B \otimes A$, suppose we have $p : A \otimes B$. Then $\otimes$-induction allows us to assume $p \equiv x \otimes y$, and we have $y \otimes x$.

$$\mathsf{sym} :\equiv \lambda p.\mathsf{let}\; x \otimes y = p \;\mathsf{in}\; y \otimes x$$

Then to prove $\prod_{(p:A\otimes B)} \mathsf{sym}(\mathsf{sym}(p)) = p$, use $\otimes$-induction again: the goal reduces to $x \otimes y = x \otimes y$ for which we have reflexivity.

$$\mathsf{inv} :\equiv \lambda p.\mathsf{let}\; x \otimes y = p \;\mathsf{in}\; \mathsf{refl}_{x \otimes y}$$

□

Colourful Variables

We need to prevent terms like $\lambda x.x \otimes x : A \to A \otimes A$, so variable use needs to be restricted somehow.

- Every variable x has a *colour* $\mathfrak{c}$.
- The relationships between colours are collected in a *palette*.

Palettes Φ are constructed by

$$1 \qquad \Phi_1 \otimes \Phi_2 \qquad \Phi_1, \Phi_2 \qquad \mathfrak{c} \qquad \mathfrak{c} \prec \Phi$$

Typical palettes:

$$\mathfrak{p} \prec \mathfrak{r} \otimes \mathfrak{b} \qquad \mathfrak{w} \prec (\mathfrak{p} \prec \mathfrak{r} \otimes \mathfrak{b}) \otimes \mathfrak{y} \qquad \mathfrak{p} \prec (\mathfrak{r} \otimes \mathfrak{b}, \mathfrak{r}' \otimes \mathfrak{b}')$$

(Similar to 'bunched' type theory P. W. O'Hearn and Pym 1999; P. O'Hearn 2003)

Using Colourful Variables

Building a term, we need to keep track of the current 'top colour'. Suppose the palette is $\mathfrak{p} \prec \mathfrak{r} \otimes \mathfrak{b}$, and we have variables

$$x^{\mathfrak{r}} : A, y^{\mathfrak{b}} : B, z^{\mathfrak{p}} : C.$$

- The top colour here is $\mathfrak{p}$.
- The only variable that can be used currently is $z : C$. (Using x here would correspond to a projection from one side of a tensor.)
- Ordinary type formers bind variables with the current top colour:

$$(x : A) \times B(x) \qquad (x : A) \to B(x) \qquad (\lambda x.b)$$

- The rules for $\otimes$ will grant us access to the other variables.

Tensor

Say the top colour is $\mathfrak{p}$.

- **Formation:** For any closed (for now) types A and B we can form the type $A \otimes B$.
- **Introduction:** Whenever we can split $\mathfrak{p}$ into two colours red and blue, use red to prove a and blue variables to prove b, then we have $a \otimes b : A \otimes B$.
- **Elimination:** If something holds for a generic tensor pair $x \otimes y$, then it holds for any particular $p : A \otimes B$.

Eg: Symmetry

Proposition

There is a function $\mathsf{sym} : A \otimes B \to B \otimes A$

Proof.

Suppose have $p : A \otimes B$. Then $\otimes$-induction on p gives $x^{\mathfrak{r}} : A$ and $y^{\mathfrak{b}} : B$, where $\mathfrak{p} \prec \mathfrak{r} \otimes \mathfrak{b}$.

Split $\mathfrak{p}$ into $\mathfrak{b}$ and $\mathfrak{r}$. Then we can form $y \ {}_{\mathfrak{b}}{\otimes}_{\mathfrak{r}} \ x : B \otimes A$.

$$\mathsf{sym} :\equiv \lambda p.\mathsf{let}\ x \ {}_{\mathfrak{r}}{\otimes}_{\mathfrak{b}} \ y = p\ \mathsf{in}\ y \ {}_{\mathfrak{b}}{\otimes}_{\mathfrak{r}} \ x$$

$\square$

Non-Eg: Colour Clashes

- We cannot define $\Delta : A \to A \otimes A$ in general.

  Given $a : A$, forming $a \otimes a : A \otimes A$ is not allowed: the two inputs to $\otimes$-intro are not well-formed in separate pieces of the palette.
- We cannot define $e : (A \otimes (A \to B)) \to B$ in general.

  We can destruct a term of $A \otimes (A \to B)$ into $x : A$ and $f : A \to B$, but $f(x)$ is not well formed: neither variable has the top colour, so can't be used.

Eg: Tensors and Ordinary Types

- Once we have access to a variable, we can use it however we like:

$$\lambda p.\mathsf{let}\ x \otimes y = p\ \mathsf{in}\ (x, x) \otimes y : A \otimes B \to (A \times A) \otimes B$$

- Using $\otimes$-elimination does not 'consume' the variable being inspected. If $f : C \otimes C \to \mathbb{N}$ we can do:

$$\lambda p.\mathsf{let}\ z \otimes w = p\ \mathsf{in}\ f(p) + f(w \otimes z) : C \otimes C \to \mathbb{N}$$

Hom

$$\frac{\Gamma \times A \vdash B}{\Gamma \vdash A \to B}$$

$$\frac{\Gamma \otimes A \vdash B}{\Gamma \vdash A \multimap B}$$

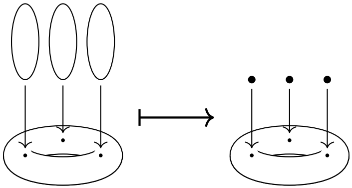
Hom

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

$$\frac{\Gamma \otimes (y : A) \vdash b : B}{\Gamma \vdash \partial y.b : (y : A) \multimap B}$$

Underlying Space

For every type A there is a type $\natural A$ that deletes the $\mathcal{C}$ information.



Marked Variables

Solved by using 'marked variables' $\underline{x} : A$, a second way of using variables.

This lets us add dependency to $\otimes$: we can form

- If A and B are types where all free variables in A and B are marked, then we can form $A \otimes B$.
- Additionally, B can be allowed to use a variable $x : A$ marked, and we can form $(\underline{x} : A) \otimes B$.

(A 'sublocal monoidal closed structure', in the language of Fu, Kishida, and Selinger 2020.)

Conclusion

- The dependency of Σ, $=$ and Π are exactly as in ordinary dependent type theory.
- The dependency of $\otimes$ and $\multimap$ is mediated by $\natural$.
- The two worlds coexist, giving a very expressive type theory!

Thanks!

References I

Robert Atkey (2018). “Syntax and Semantics of Quantitative Type Theory”. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. DOI: 10.1145/3209108.3209189.

James Cheney (2009). “A Simple Nominal Type Theory”. In: *Proceedings of the International Workshop on Logical Frameworks and Metalanguages: Theory and Practice (LFMTP 2008)*. Vol. 228. DOI: 10.1016/j.entcs.2008.12.115.

— (2012). “A dependent nominal type theory”. In: *Logical Methods in Computer Science* 8.1. DOI: 10.2168/LMCS-8(1:8)2012.

Peng Fu, Kohei Kishida, and Peter Selinger (2020). “Linear Dependent Type Theory for Quantum Programming Languages: Extended Abstract”. In: *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*. DOI: 10.1145/3373718.3394765.

References II

Valery Isaev (2021). "Indexed type theories". In: *Mathematical Structures in Computer Science* 31.1. DOI: 10.1017/S0960129520000092.

Neelakantan R. Krishnaswami, Pierre Pradic, and Nick Benton (2015). "Integrating Linear and Dependent Types". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. DOI: 10.1145/2676726.2676969.

Conor McBride (2016). "I Got Plenty o' Nuttin'". In: *A list of successes that can change the world*. Vol. 9600. DOI: 10.1007/978-3-319-30936-1_12.

Benjamin Moon, Harley Eades III, and Dominic Orchard (2021). "Graded Modal Dependent Type Theory". In: *Programming Languages and Systems*. DOI: 10.1007/978-3-030-72019-3_17.

Peter O'Hearn (2003). "On bunched typing". In: *Journal of Functional Programming* 13.4. DOI: 10.1017/S0956796802004495.

Peter W. O'Hearn and David J. Pym (1999). "The Logic of Bunched Implications". In: *Bulletin of Symbolic Logic* 5.2. DOI: 10.2307/421090.

References III

Ulrich Schöpp (2006). “Names and Binding in Type Theory”. PhD thesis. University of Edinburgh. URL: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.561934.

Ulrich Schöpp and Ian Stark (2004). “A Dependent Type Theory with Names and Binding”. In: *Computer Science Logic*. Vol. 3210. DOI: 10.1007/978-3-540-30124-0_20.

Matthjis Vákár (2014). *Syntax and Semantics of Linear Dependent Types*. arXiv: 1405.0033 [cs.AT].