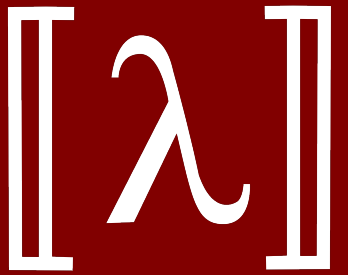


Lambda Calculus

as The Internal Language of Cartesian Closed Categories

Jacob Neumann

80-413/713 - November 2020

0 CCCs: An Algebraic Perspective

Terminal Objects

A **terminal object** is an object 1, such that for every object A of $\mathbb{C}$, there is a morphism $!_A : A \to 1$

$$A \xrightarrow{!_A} 1$$

such that

$$\text{for all } f : A \to 1, \quad f = !_A$$

Products

For each A, B, their **product** is an object $A \times B$ with an operation

$$\langle -, - \rangle : \mathsf{Hom}(Z, A) \times \mathsf{Hom}(Z, B) \to \mathsf{Hom}(Z, A \times B)$$

and maps $p_1 : A \times B \to A$ and $p_2 : A \times B \to B$ such that, for all f, g, h,

$$p_1 \circ \langle f, g \rangle = f$$

$$p_2 \circ \langle f, g \rangle = g$$

$$\langle p_1 \circ h, p_2 \circ h \rangle = h$$

Exponentials

For all B, C, their **exponential** is an object C^B equipped with an operation

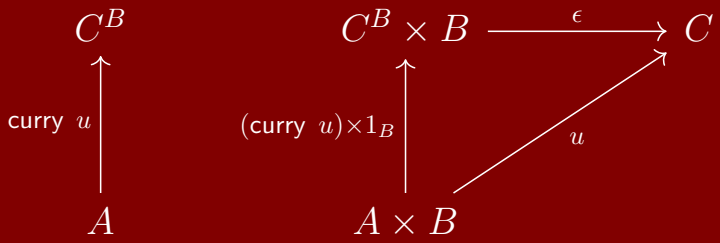
$$\text{curry} : \text{Hom}(A \times B, C) \to \text{Hom}(A, C^B)$$

and a map $\epsilon : C^B \times B \to C$ such that for all $u : A \times B \to C$ and all $v : A \to C^B$,

$$\epsilon \circ ((\text{curry } u) \times 1_B) = u$$

$$\text{curry}(\epsilon \circ (v \times 1_B)) = v$$

Recall $v \times 1_B : A \times B \to C^B \times B$ is $\langle v \circ p_1, p_2 \rangle$.



Note: the operation $v \mapsto \epsilon \circ (v \times 1_B)$ is a kind of "uncurrying".

Adjunction Definition

This definition is equivalent to the definition of exponentials as right adjoints:

$$(-) \times B \dashv (-)^B.$$

The required isomorphism is given by

$$\text{curry} : \mathsf{Hom}(A \times B, C) \to \mathsf{Hom}(A, C^B)$$

$$\text{uncurry} : \mathsf{Hom}(A, C^B) \to \mathsf{Hom}(A \times B, C)$$

where $\text{uncurry}(v) = \epsilon \circ \langle v \circ p_1, p_2 \rangle = \epsilon \circ (v \times 1_B)$.

ϵ is the counit of the adjunction.

Recall that every adjunction $L \dashv R$ gives rise to a counit natural transform $\epsilon : L \circ R \to 1_{\mathrm{dom}(R)}$

In the case of $(-) \times B \dashv (-)^B$, this means

$$\epsilon_C : C^B \times B \to C$$

natural in C. This is the "evaluation" morphism for C^B.

Summary

Here, A, B, C, Z range over all objects of $\mathbb{C}$

Structure	**Required Data**	**Laws**
Terminal Object	$!_A : A \to 1$	For all $k : A \to 1$, $k = !_A$
Binary Products	$p_1 : A \times B \to A$ $p_2 : A \times B \to B$ $\langle -, - \rangle : \mathsf{Hom}(Z, A) \times \mathsf{Hom}(Z, B) \to \mathsf{Hom}(Z, A \times B)$	For all $f : Z \to A$, $g : Z \to B$, $p_1 \circ \langle f, g \rangle = f$ $p_2 \circ \langle f, g \rangle = g$ For all $h : Z \to A \times B$, $h = \langle p_1 \circ h, p_2 \circ h \rangle$
Exponentials	$\epsilon_C : C^B \times B \to C$ $\mathsf{curry} : \mathsf{Hom}(A \times B, C) \to \mathsf{Hom}(A, C^B)$	For all $u : A \times B \to C$, $v : A \to C^B$, $\epsilon \circ ((\mathsf{curry}\ u) \times 1_B) = u$ $\mathsf{curry}(\epsilon \circ (v \times 1_B)) = v$

1 The Simply-Typed Lambda Calculus

History & Motivation of the Lambda Calculus

The lambda calculus

- Developed in 1930s by Alonzo Church as a mathematical model of computation (prior to the existence of electronic computers). Provably equivalent to other such notions (e.g. Turing machines)
- Has "untyped" and "typed" variants
- Theoretical basis of functional programming and type theory

Lambda Abstraction

Lambda Calculus studies functions between **types**, which can be thought of like sets. We denote "x is an element of T" as $x : T$ instead of $x \in T$.

To define a function from T_1 to T_2, it suffices to specify a rule assigning to each $x : T_1$ some expression $e : T_2$, where the value of e, in general, can depend on x.

$$(\lambda x.e) : T_1 \to T_2$$

This is the *function* which "accepts" $x : T_1$ and "returns" $e : T_2$.

General Setup

- We'll build up a system of **types** T
- We'll define **contexts**, which consist of several "typed variable declarations"
- *In* a context Γ, we'll form **terms** of given types
- In a context Γ, we'll prove **equalities** between terms

Types

To get started, we fix some set $\mathbf{Ty}_0$ of **basic types**, and recursively generate the set of all types $\mathbf{Ty}$ from it.

$$
\begin{aligned}
T, T' ::= {} & T_0 && (T_0 \in \mathbf{Ty}_0) \\
\mid {} & \mathbf{1} && \text{(Unit type)} \\
\mid {} & T \times T' && \text{(Product types)} \\
\mid {} & T \to T' && \text{(Arrow types)}
\end{aligned}
$$

Contexts

For some variable name x and some $T \in \mathbf{Ty}$, we can form the **type judgment** $x : T$, read "x is of type T".

A **context** Γ is a finite list of typing judgments $x_1 : T_1, \ldots, x_n : T_n$,

$$\begin{aligned} \Gamma ::= & && \text{(Empty context)} \\ & \mid \Gamma, x : T && \text{(Context Extension)} \end{aligned}$$

(We usually make some syntactic requirements on contexts, e.g. that all the variable names are distinct)

General Setup

- ✓ We'll build up a system of **types** T
- ✓ We'll define **contexts**, which consist of several "typed variable declarations"
- *In* a context Γ, we'll form **terms** of given types
- In a context Γ, we'll prove **equalities** between terms

Term-Building Activities

Now recursively build up *terms-in-context* of these types. We write

$$\Gamma \vdash t : T$$

to indicate that t is a term of type T in context Γ.

- We may also have some basic terms $t_0 \in \mathbf{Tm}_0$, each of a specified basic type. If $t_0 \in \mathbf{Tm}_0$ and is specified to be of type T_0, then

$$\Gamma \vdash t_0 : T_0 \qquad \text{for any } \Gamma$$

- $\star$ is always a term of type $\mathbf{1}$:

$$\Gamma \vdash \star : \mathbf{1} \qquad \text{for any } \Gamma$$

Term-Building

- If $\Gamma \vdash t_1 : T_1$ and $\Gamma \vdash t_2 : T_2$,

$$\Gamma \vdash (t_1, t_2) : T_1 \times T_2$$

- If $\Gamma \vdash p : T_1 \times T_2$,

$$\Gamma \vdash \mathsf{fst}(p) : T_1 \qquad \text{and} \qquad \Gamma \vdash \mathsf{snd}(p) : T_2$$

- If $\Gamma, x : T_1 \vdash e : T_2$,

$$\Gamma \vdash (\lambda x.e) : T_1 \to T_2$$

- If $\Gamma \vdash f : T_1 \to T_2$ and $\Gamma \vdash v : T_1$,

$$\Gamma \vdash (f\ v) : T_2$$

Rules of Lambda Calculus

$$\frac{}{\Gamma, x : T \vdash x : T}(\text{Var.})$$

$$\frac{\Gamma \vdash x_2 : T_2}{\Gamma, x_1 : T_1 \vdash x_2 : T_2}(\text{Weak.})$$

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t = t} \quad \frac{\Gamma \vdash t_1 = t_2}{\Gamma \vdash t_2 = t_1} \quad \frac{\Gamma \vdash t_1 = t_2 \quad \Gamma \vdash t_2 = t_3}{\Gamma \vdash t_1 = t_3}$$

Unit & Product Rules

$$\frac{\Gamma \vdash w : \mathbf{1}}{\Gamma \vdash w = \star}$$

$$\frac{\Gamma \vdash p : T_1 \times T_2}{\Gamma \vdash (\mathsf{fst}(p), \mathsf{snd}(p)) = p}$$

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \mathsf{fst}(t_1, t_2) = t_1} \qquad \frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \mathsf{snd}(t_1, t_2) = t_2}$$

Beta & Eta

$$\frac{\Gamma \vdash (\lambda x.e) : T_1 \to T_2 \qquad \Gamma \vdash v : T_1}{\Gamma \vdash (\lambda x.e)(v) = e[v/x]}(\beta)$$

$$\frac{\Gamma \vdash f : T_1 \to T_2}{\Gamma \vdash (\lambda x.f\ x) = f}(\eta)$$

Example

Claim 1 For any types T_1, T_2, T_3 and any context Γ, there exist terms in context Γ

$$\Gamma \vdash \mathsf{split} : (T_1 \to T_2 \times T_3) \to (T_1 \to T_2) \times (T_1 \to T_3)$$
$$\Gamma \vdash \mathsf{pair} : (T_1 \to T_2) \times (T_1 \to T_3) \to (T_1 \to T_2 \times T_3)$$

such that

$$\Gamma, f : T_1 \to T_2 \times T_3 \qquad \vdash \mathsf{pair}(\mathsf{split}(f)) = f$$
$$\Gamma, g_1 : T_1 \to T_2, g_2 : T_1 \to T_3 \vdash \mathsf{split}(\mathsf{pair}(g_1, g_2)) = (g_1, g_2)$$

Proof (sketch)

$$\mathsf{split} :\equiv \lambda f.(\lambda x.\mathsf{fst}(f\ x), \lambda x.\mathsf{snd}(f\ x))$$

$$\mathsf{pair} :\equiv \lambda p.\lambda x.(\mathsf{fst}(p)(x), \mathsf{snd}(p)(x))$$

2 Categorical Semantics

High-level idea

The preceding syntactic structure of types T, contexts Γ, and terms-in-context $\Gamma \vdash x : T$ is the **language of lambda calculus**, $\mathcal{L}$.

"Interpret" $\mathcal{L}$ inside a cartesian closed category $\mathbb{C}$, i.e. we want some mapping[1]

$$[\![-]\!] : \mathcal{L} \to \mathbb{C}$$

In particular, $[\![T]\!]$ and $[\![\Gamma]\!]$ will be objects of $\mathbb{C}$, and $[\![\Gamma \vdash x : T]\!]$ will be a morphism $[\![\Gamma]\!] \to [\![T]\!]$. Goal:

$$\Gamma \vdash t_1 = t_2 \text{ is provable} \quad \Longrightarrow \quad [\![\Gamma \vdash t_1]\!] = [\![\Gamma \vdash t_2]\!]$$

[1] $[\![-]\!]$ is actually a functor, if we view $\mathcal{L}$ as a category in an appropriate way.

Interpreting Types

$$
\begin{aligned}
T, T' ::= {} & T_0 && (T_0 \in \mathbf{Ty}_0) \\
\mid {} & \mathbf{1} && \text{(Unit type)} \\
\mid {} & T \times T' && \text{(Product types)} \\
\mid {} & T \to T' && \text{(Arrow types)}
\end{aligned}
$$

Define the interpretation of types, $[\![-]\!] : \mathbf{Ty} \to \mathrm{Ob}(\mathbb{C})$.

- We must supply $[\![T_0]\!] \in \mathrm{Ob}(\mathbb{C})$ for each $T_0 \in \mathbf{Ty}_0$
- $[\![\mathbf{1}]\!] = 1$, the terminal object
- $[\![T_1 \times T_2]\!] = [\![T_1]\!] \times [\![T_2]\!]$
- $[\![T_1 \to T_2]\!] = [\![T_2]\!]^{[\![T_1]\!]}$

Interpreting Contexts

Recall that $\mathbf{Ctx}$, the set of all contexts, is defined recursively by

$$\begin{aligned} \Gamma ::= & && \text{(Empty context)} \\ & \mid \Gamma, x : T && \text{(Context Extension)} \end{aligned}$$

So define $[\![-]\!] : \mathbf{Ctx} \to \mathrm{Ob}(\mathbb{C})$

- Interpret the empty context as 1, the terminal object
- $[\![\Gamma, x : T]\!] = [\![\Gamma]\!] \times [\![T]\!]$

Interpreting Terms

To complete our interpretation of the lambda calculus, we must interpret terms-in-context as morphisms of $\mathbb{C}$. Specifically,

$$[\![\Gamma \vdash t : T]\!] : [\![\Gamma]\!] \to [\![T]\!].$$

- If $t_0 \in \mathbf{Tm}_0$ is a basic term of type T_0, we must pick $[\![\vdash t_0 : T_0]\!] : 1 \to [\![T_0]\!]$
- For any Γ, $[\![\Gamma \vdash \star : \mathbf{1}]\!] = !_{[\![\Gamma]\!]}$.

$$[\![\Gamma]\!] \xrightarrow{[\![\Gamma \vdash \star : \mathbf{1}]\!]} [\![\mathbf{1}]\!] = 1 \qquad \frac{\Gamma \vdash w : \mathbf{1}}{\Gamma \vdash w = \star}$$

Structural Rules

Variable Rule:

$$\frac{}{\Gamma, x : T \vdash x : T}$$

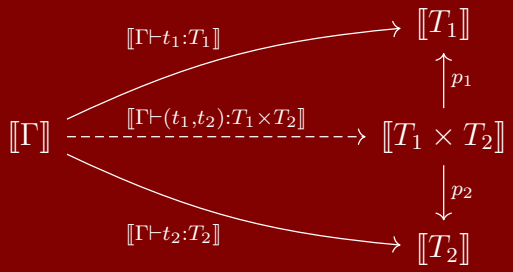
Weakening:

$$\frac{\Gamma \vdash x_2 : T_2}{\Gamma, x_1 : T_1 \vdash x_2 : T_2}$$

$$[\![\Gamma]\!] \times [\![T]\!] \xrightarrow{p_2} [\![T]\!]$$

$$[\![\Gamma]\!] \times [\![T_1]\!] \xrightarrow{[\![\Gamma \vdash x_2 : T_2]\!] \times !_{[\![T_1]\!]}} [\![T_2]\!] \times 1 = [\![T_2]\!]$$

Product Terms



- $[\![\Gamma \vdash (t_1, t_2) : T_1 \times T_2]\!] = \langle [\![\Gamma \vdash t_1 : T_1]\!], [\![\Gamma \vdash t_2 : T_2]\!] \rangle$
- $[\![\Gamma \vdash \mathsf{fst}(p) : T_1]\!] = p_1 \circ [\![\Gamma \vdash p : T_1 \times T_2]\!]$

Product Rules

$$\frac{\Gamma \vdash p : T_1 \times T_2}{\Gamma \vdash (\mathsf{fst}(p), \mathsf{snd}(p)) = p}$$

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \mathsf{fst}(t_1, t_2) = t_1} \qquad \frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \mathsf{snd}(t_1, t_2) = t_2}$$

Arrows are Exponentials

- Abstraction: want to fulfill

$$\frac{\Gamma, x : T_1 \vdash e : T_2}{\Gamma \vdash (\lambda x.e) : T_1 \to T_2}$$

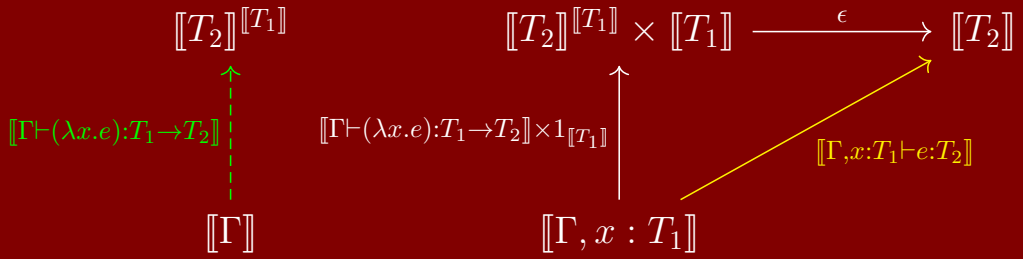
  i.e. given $[\![\Gamma]\!] \times [\![T_1]\!] \to [\![T_2]\!]$, define a morphism $[\![\Gamma]\!] \to [\![T_2]\!]^{[\![T_1]\!]}$

- Application: want to fulfill

$$\frac{\Gamma \vdash f : T_1 \to T_2 \qquad \Gamma \vdash v : T_1}{\Gamma \vdash (f\ v) : T_2}$$

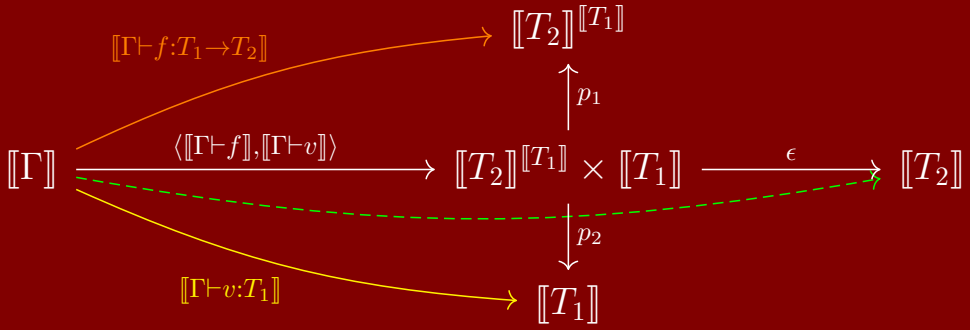
  i.e. given $[\![\Gamma]\!] \to [\![T_2]\!]^{[\![T_1]\!]}$ and $[\![\Gamma]\!] \to [\![T_1]\!]$, define a morphism $[\![\Gamma]\!] \to [\![T_2]\!]$

λ-Abstraction



$$[\![\Gamma \vdash (\lambda x.e) : T_1 \to T_2]\!] := \text{curry } [\![\Gamma, x : T_1 \vdash e : T_2]\!]$$

Application



$$[\![\Gamma \vdash (f\ v) : T_2]\!] := \epsilon \circ \langle [\![\Gamma \vdash f : T_1 \to T_2]\!], [\![\Gamma \vdash v : T_1]\!] \rangle$$

Beta

If we define substitution in the appropriate way, the β rule

$$\frac{\Gamma \vdash (\lambda x.e) : T_1 \to T_2 \qquad \Gamma \vdash v : T_1}{\Gamma \vdash (\lambda x.e)(v) = e[v/x]}(\beta)$$

is valid for every interpretation of lambda calculus in any CCC, i.e. $[\![\Gamma \vdash (\lambda x.e)(v)]\!] = [\![\Gamma \vdash e[v/x]]\!]$. This usually involves proving a "substitution lemma".

Eta

The η rule

$$\frac{\Gamma \vdash f : T_1 \to T_2}{\Gamma \vdash (\lambda x.f\ x) = f}(\eta)$$

is valid for all interpretations: $[\![\Gamma \vdash \lambda x.f\ x]\!] = [\![\Gamma \vdash f]\!]$.This is proved using the universal property of exponentials.

3 Results

Soundness

Thm. 1 (Soundness) For any CCC $\mathbb{C}$ and any interpretation $[\![-]\!] : \mathcal{L} \to \mathbb{C}$ and any terms $t_1, t_2 : T$ in context Γ, if we can deduce

$$\Gamma \vdash t_1 = t_2$$

using the rules of lambda calculus, then

$$[\![\Gamma \vdash t_1 : T]\!] = [\![\Gamma \vdash t_2 : T]\!].$$

Proof by induction on the deduction of $\Gamma \vdash t_1 = t_2$.

Isomorphism Corollary

Cor. 1.1 If T_1 and T_2 are types and in any context Γ there exists terms

$$\Gamma \vdash F : T_1 \to T_2$$

$$\Gamma \vdash G : T_2 \to T_1$$

such that

$$\Gamma, x : T_1 \vdash G(F(x)) = x \qquad \Gamma, y : T_2 \vdash F(G(y)) = y$$

then

$$[\![T_1]\!] \cong [\![T_2]\!].$$

Application

Cor 1.2 If $\mathbb{C}$ is a CCC, $[\![-]\!] : \mathcal{L} \to \mathbb{C}$, then for any types T_1, T_2, T_3,

$$([\![T_2]\!] \times [\![T_3]\!])^{[\![T_1]\!]} \cong [\![T_2]\!]^{[\![T_1]\!]} \times [\![T_3]\!]^{[\![T_1]\!]}$$

Proof: By **Claim 1** and **Cor 1.1**.

Language of a CCC

For any CCC $\mathbb{C}$, we can define the **language of** $\mathbb{C}$, $\mathcal{L}(\mathbb{C})$, to be the lambda calculus language

- whose basic types are the objects of $\mathbb{C}$
- whose basic terms of type X are the morphisms $1 \to X$ in $\mathbb{C}$.

Define a canonical $[\![-]\!] : \mathcal{L}(\mathbb{C}) \to \mathbb{C}$ by interpreting each basic type and term as itself.

Thm. 2 For any objects X, Y, Z of a cartesian closed category, there is an isomorphism

$$(Y \times Z)^X \cong Y^X \times Z^X$$

Further Directions

- Completeness: Prove that any equality which is valid in every CCC is provable in the lambda calculus.
- Duality: Show that the syntax of lambda calculus is – in a sense – dual to its model theory
- Add fancier stuff:
  - Initial object
  - Coproducts
  - Natural numbers object, and other inductive types
  - Dependent types and higher inductive types
- Do an analogous development for other formal systems and classes of categories (categorical logic!)

Thank you!