The Curry-Howard Isomorphism

Student: Chuangjie Xu (1061493)
Supervisor: Achim Jung
February 1, 2011

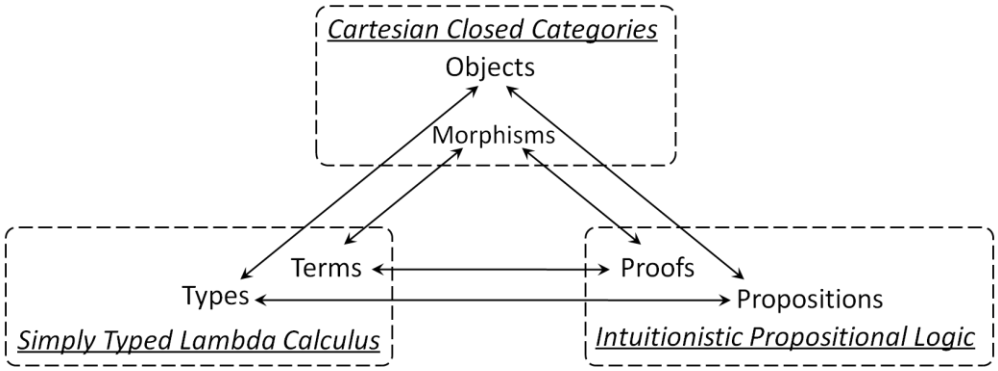
Abstract

1. Introduction

Logicians must be familiar with modus ponens $P \to Q, P \vdash Q$, a very common rule of inference, saying that given implication $P \to Q$ and proposition P, we have Q. Programmers may frequently use function application: if f is a function of type $P \to Q$ and x is an argument of type P, then the application $f\ x$ is of type Q. Interestingly, modus ponens behaves the same as function application. So, are proofs related to programs? Yes, there is an amazing precise correspondence between them which is described in the Curry-Howard Isomorphism.

In the 1930s, Haskell Curry observed a correspondence between types of combinators and propositions in intuitionist implicational logic. But, at that time, it was viewed as no more than a curiosity. About three decades later, William Howard extended this correspondence to first order logic by introducing dependent types. Therefore, this correspondence is called the Curry-Howard Isomorphism.

The Curry-Howard Isomorphism states a correspondence between systems of formal logic and computational calculi. For years, it has been extended to more expressive logics, e.g. higher order logic, and other mathematical systems, e.g. cartesian closed categories. In this project, I mainly probed into one of its extensions, the three-way-correspondence between intuitionistic propositional logic, simply-typed lambda calculus and cartesian closed categories, with propositions or types being interpreted as objects and proofs or terms as morphisms.



Intuitionistic logic is a formalization of Brouwer's intuitionism. As the founder of intuitionism, L. E. J. Brouwer avoided use of formal language or logic all his life. But his attitude did not stop others considering formalizations of parts of intuitionism. In the 1930s,

Arend Heyting, a former student of Brouwer, produced the first complete axiomatizations for intuitionistic propositional and predicate logic. In intuitionistic logic, the law of excluded middle and double negation elimination are no longer axioms.

The lambda calculus was introduced by Alonzo Church in the early 1930s as a formal system to provide a functional foundation for mathematics. Since Church's original system was shown to be logically inconsistent, he gave just a consistent subtheory of his original system dealing only with the functional part. Then, in 1940, Church also introduced a typed interpretation of the lambda calculus by giving each term a unique type. Today, the typed lambda calculus serves as the foundation of the modern type systems in computer science.

Category first appeared in Samuel Eilenberg and Saunders Mac Lane's paper written in 1945. It was originally introduced to describe the passage from one type of mathematical structure to another. In recent decades, category theory has found use for computer science. For instant, it has a profound influence on the design of functional and imperative programming languages, e.g. Haskell and Agda.

Looking from the historical perspective, these three different systems seem to have different origins, not related to each other. However, Joachim Lambek showed in the early 1970s that cartesian closed categories provided a formal analogy between proofs in intuitionistic propositional logic and types in combinatory logic. As a result, some people may use Curry-Howard-Lambek Isomorphism to refer to this three-way-correspondence.

2. Background

2.1 Intuitionistic Logic

Intuitionistic logic is also called constructive logic. As a formalization of intuitionism, it differs from classical logic not only in that syntactically some laws in classical logic are not axioms of the system but also in the meaning for statements to be true. The judgments about statements are based on the existence of a proof or a "construction" of that statement. Its existence property makes it practically useful, e.g. provided that a constructive proof that an object exists, one can turn it into an algorithm for generating an example of the object.

One vertex in the correspondence-triangle is the intuitionistic propositional logic. So the introduction to intuitionistic logic in this dissertation is up to the propositional one.

2.1.1 Syntax

The language of intuitionistic propositional logic is similar to the one of classical propositional logic. Customarily, people use $\bot$, $\rightarrow$, $\wedge$, $\vee$ as basic connectives and treat $\neg\varphi$ as an abbreviation for $\varphi \rightarrow \bot$.

2.1.1 Definition (Set of formulas) Given an infinite set of propositional variables, the *set* Φ *of formulas* in intuitionistic propositional logic is defined by induction, represented in the following grammar:

$$\Phi \ ::= \ P \ | \ \bot \ | \ \neg\Phi \ | \ (\Phi \rightarrow \Phi) \ | \ (\Phi \wedge \Phi) \ | \ (\Phi \vee \Phi)$$

where P is a *propositional variable*, $\bot$ is *contradiction*, $\neg$ is *negation*, $\rightarrow$ is *implication*, $\wedge$

is *conjunction*, and $\vee$ is *disjunction*.

Given a set Γ of propositions and a proposition φ, the relation $\Gamma \vdash \varphi$ says that there is a derivation with conclusion φ from hypotheses in Γ. Here Γ is also called *context*. If Γ is empty, we write $\vdash \varphi$ and say that φ is a *theorem*.

For notational convenience, we use the following abbreviations:

- $\varphi_1, \varphi_2, \ldots, \varphi_n$ for $\{\varphi_1, \varphi_2, \ldots, \varphi_n\}$;
- Γ, φ for $\Gamma \cup \{\varphi\}$;

The natural deduction system allows one to derive conclusions from premises. The axiom and inference rules in this deduction system show how the relation $\vdash$ is built.

2.1.2 Definition (Natural Deduction) Given a set of propositional variable, the relation $\Gamma \vdash \varphi$ is obtained by using the following axiom and inference rules

- *Axiom*

$$\frac{}{\varphi \vdash \varphi}\,(axiom)$$

Since proposition φ appears in the context, one can conclude it from the context.

- *Adding hypothesis into context*

$$\frac{\Gamma \vdash \varphi}{\Gamma, \psi \vdash \varphi}\,(add)$$

This rules allows one to add additional hypotheses into the context.

- $\rightarrow$*-introduction*

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \rightarrow \psi}\,(\rightarrow \mathrm{I})$$

If one can derive ψ from φ as a hypothesis, then $\varphi \rightarrow \psi$ is derivable without hypothesis φ.

- $\rightarrow$*-elimination*

$$\frac{\Gamma \vdash \varphi \rightarrow \psi, \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}\,(\rightarrow \mathrm{E})$$

Both the conditional claim "if φ then ψ" and φ are provided, one can conclude ψ. As mentioned in the beginning, this is a very common inference rule which is also called *modus ponens*.

- $\wedge$*-introduction*

$$\frac{\Gamma \vdash \varphi, \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\,(\wedge \mathrm{I})$$

If both φ and ψ are derivable from Γ, $\varphi \wedge \psi$ is also derivable.

- $\wedge$*-elimination*

$$\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\,(\wedge \mathrm{E})_1 \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\,(\wedge \mathrm{E})_2$$

Provided that conjunction $\varphi \wedge \psi$ is derivable from Γ, both of its components are also derivable.

- $\vee$*-introduction*

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}(\vee \mathrm{I})_1 \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}(\vee \mathrm{I})_2$$

One can conclude disjunction $\varphi \vee \psi$ from either φ or ψ.

- $\vee$-*elimination*

$$\frac{\Gamma \vdash \varphi \to \rho, \quad \Gamma \vdash \psi \to \rho, \quad \Gamma \vdash \varphi \vee \psi}{\Gamma \vdash \rho}(\vee \mathrm{E})$$

If ρ follows φ, ρ follows ψ and $\varphi \vee \psi$ is given, one can conclude ρ.

- $\perp$-*elimination*

$$\frac{\Gamma \vdash \perp}{\Gamma \vdash \varphi}(\perp \mathrm{E})$$

From contradiction $\perp$, we can derive any propositions. This rule is also called *principle of explosion* or *ex falso quodlibet*.

Some examples are given here to show how the rules above are used to build a theorem:

(1) $\vdash \varphi \to (\psi \to \varphi)$

$$\frac{\dfrac{\dfrac{\overline{\varphi \vdash \varphi}}{\varphi, \psi \vdash \varphi}(add)}{\varphi \vdash \psi \to \varphi}(\to \mathrm{I})}{\vdash \varphi \to (\psi \to \varphi)}(\to \mathrm{I})$$

(2) $\vdash (\neg\varphi \vee \psi) \to (\varphi \to \psi)$

$$\frac{\dfrac{\dfrac{\dfrac{\neg\varphi, \varphi \vdash \perp}{\neg\varphi, \varphi \vdash \psi}(\perp \mathrm{E})}{\neg\varphi \vdash \varphi \to \psi}(\to \mathrm{I})}{\vdash \neg\varphi \to (\varphi \to \psi)}(\to \mathrm{I})}{\neg\varphi \vee \psi \vdash \neg\varphi \to (\varphi \to \psi)}(add) \quad \frac{\overline{\vdash \psi \to (\varphi \to \psi)}(1)}{\neg\varphi \vee \psi \vdash \psi \to (\varphi \to \psi)}(add) \quad \overline{\neg\varphi \vee \psi \vdash \neg\varphi \vee \psi}}{\dfrac{\neg\varphi \vee \psi \vdash \varphi \to \psi}{\vdash (\neg\varphi \vee \psi) \to (\varphi \to \psi)}(\to \mathrm{I})}(\vee \mathrm{E})$$

2.1.2 Constructive Semantics

In intuitionistic logic, each proposition is given an intuitive meaning that establishes the proposition, called a *proof* or *construction*. Then all the connectives in intuitionistic propositional logic are given another interpretation, sometimes called the *BHK-interpretation*.

2.1.3 Definition (Proofs) The expression $p : \varphi$ denotes that p is a construction that establishes proposition φ, we call this p a *proof of* φ. The following rules explain the constructive semantics of propositional connectives:

- $\perp$ There is no proof of $\perp$.
- $p : \varphi \to \psi$ The proof p of implication $\varphi \to \psi$ is a method converting every proof $a : \varphi$ into a proof $p(a) : \psi$.
- $p : \varphi \wedge \psi$ The proof p of conjunction $\varphi \wedge \psi$ is a pair of proofs $\langle p_1, p_2 \rangle$ such that $p_1 : \varphi$ and $p_2 : \psi$, with projections π_1, π_2 that return the first and second proofs in a pair.
- $p : \varphi \vee \psi$ The proof p of disjunction $\varphi \vee \psi$ is a pair $\langle i, a \rangle$ where $i \in \{0,1\}$ such

that $i = 0$ and $a : \varphi$ or $i = 1$ and $a : \psi$. In words, the i indicates which disjunct is correct and a is the proof of that disjunct.

- $p : \neg\varphi$ The proof p of negation of φ is a method that transforms every proof $a : \varphi$ into $p(a) : \bot$, i.e. p tells us that φ has no proofs.

According the definition above, $\bot \rightarrow \varphi$ should have a proof which can be any method. That is because the argument does not exist ($\bot$ has no proofs) and this method is never applied to its argument, which means any method can be considered as the proof of $\bot \rightarrow \varphi$.

The following examples demonstrate the proof interpretation for some propositions:

(1) $\varphi \rightarrow (\psi \rightarrow \varphi)$

Let p is a proof of $\varphi \rightarrow (\psi \rightarrow \varphi)$. Then p is a method converting proofs of φ into a proof of $\psi \rightarrow \varphi$. If we already have $p_1 : \varphi$ and $p_2 : \psi$, then the proof of $\psi \rightarrow \varphi$ is a transformation $p_2 \longmapsto p_1$ which can be represented as $\lambda p_2.p_1$ (using λ-calculus denotation). Hence, $p = \lambda p_1.(\lambda p_2.p_1) : \varphi \rightarrow (\psi \rightarrow \varphi)$.

(2) $\varphi \rightarrow \neg\neg\varphi$

Assume that $p_1 : \varphi$ and $p_2 : \neg\varphi$. According to definition 2.1.3, we have $p_2(p_1) : \bot$. Since $\neg\neg\varphi$ stands for $\neg\varphi \rightarrow \bot$, its proof is $\lambda p_2.p_2(p_1)$. Therefore, the proof of $\varphi \rightarrow \neg\neg\varphi$ is $p = \lambda p_1.\big(\lambda p_2.p_2(p_1)\big) : \varphi \rightarrow \neg\neg\varphi$.

(3) $(\varphi \wedge \psi) \rightarrow (\psi \wedge \varphi)$

Assume $q : \varphi \wedge \psi$. According to definition 2.1.3, we have $\pi_1(q) : \varphi$ and $\pi_2(q) : \psi$. Then the pair $\langle \pi_2(q), \pi_1(q) \rangle$ is a proof of $(\psi \wedge \varphi)$. So the proof of $(\varphi \wedge \psi) \rightarrow (\psi \wedge \varphi)$ is $p = \lambda q.\langle \pi_2(q), \pi_1(q) \rangle : (\varphi \wedge \psi) \rightarrow (\psi \wedge \varphi)$.

(4) $(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)$

Assume $q : \neg\varphi \vee \psi$. According to definition 2.1.3, we have $\pi_2(q) : \neg\varphi$ if $\pi_1(q) = 0$ or $\pi_2(q) : \psi$ if $\pi_1(q) = 1$. ???

Both the law of excluded middle and double negation elimination are axioms in the system of classical logic. However, in intuitionistic one neither of them can be proved, i.e. there is no constructive proof for them.

(1) $\varphi \vee \neg\varphi$

If $\varphi \vee \neg\varphi$ has a proof p, then p should be a pair $\langle i, a \rangle$ such that $a : \varphi$ if $i = 0$ or $a : \neg\varphi$ if $i = 1$. However, for an arbitrary proposition φ we do not know whether φ or $\neg\varphi$ has a proof. As a result, the value of i cannot be known. Therefore, there is no proof of $\varphi \vee \neg\varphi$ for arbitrary φ.

(2) $\neg\neg\varphi \rightarrow \varphi$

Assume that we have a proof $p : \neg\neg\varphi$. According to definition 2.1.3, p tells us that there is no proof of $\neg\varphi$. Then we end here and unable to obtain a proof of φ. Hence, there is no proof of $\neg\neg\varphi \rightarrow \varphi$.

From the above examples, one may make a conclusion that for every theorem in

intuitionistic logic there is always a closed proof, i.e. a term without variables, while for those which are axioms in classical logic but not in intuitionistic one, it is possible to find a corresponding closed proof.

2.2 Lambda Calculus

The λ-calculus is a family of prototype programming languages. The simplest of these languages is the pure lambda calculus which studies only functions and their applicative behavior but does not contain any constants or types.

The syntax of λ-terms in pure lambda calculus is simple. A λ-term can be a term-variable, an application, or an abstraction. *Application* is one of the primitive operations. A lambda application FA denotes that the function F is applied to the argument A. Another basic operation is *abstraction*. Let $M \equiv M[x]$ be an expression possibly containing or depending on variable x. Then the lambda abstraction $\lambda x.M$ denotes the function $x \longmapsto M[x]$.

There are three kinds of equivalences playing an important role in λ-calculus. The first one, α-equivalence, states that a change of bound variables in a λ-term does not change its meaning. β-equivalence shows how to evaluate an application by using substitution. And η-equivalence refers to the idea of extensionality. All of them will be discussed in more detail in 2.2.1 which is about the proof system of the simply typed lambda calculus.

2.2.1 Simply typed lambda calculus $\boldsymbol{\lambda^{\rightarrow}}$

Church introduced a typed interpretation of lambda calculus, now called the *simply typed lambda calculus*, by giving each λ-term a unique type as its structure. The types in simply typed lambda calculus do not contain type variables. The standard type forms include functions, products, sums, initial and terminal types.

The one with only function type constructor $\rightarrow$ is called the *simply typed lambda calculus with function types*, indicated by $\lambda^{\rightarrow}$. With products, sums and functions, we have $\lambda^{\times,+,\rightarrow}$ and so on. However, $\lambda^{\rightarrow}$ is as expressive as other versions of simply typed lambda calculus. We will have a look at $\lambda^{\rightarrow}$ first and then introduce the other types into it.

To begin with, we give the definition of types in $\lambda^{\rightarrow}$.

2.2.1 Definition (Types) Assume that a set of type-constants is given. Then the *types* in $\lambda^{\rightarrow}$ are defined as follows:

i) each type-constant b is a type, called an *atom* (or *atomic type*);
ii) if σ and τ are types then $\sigma \rightarrow \tau$ is a type, called a *function type*.

There are two general frameworks for describing the denotational semantics of typed lambda calculus, Henkin models and cartesian closed categories. In a Henkin model, each type expression is interpreted as a set, the set of values of that type. But we will not go into Henkin models too much. The interpretation in CCCs will be discussed in section 3.3.

The syntax of $\lambda^{\rightarrow}$ is essentially that of the pure lambda calculus itself. By assigning type σ to a lambda term M, we have an expression $M : \sigma$ called a *type-assignment*, saying that

term M has type σ. Here, M is called its *subject* and σ its *predicate*. However, not every pure lambda term can be given a type (a counter-example is given later). The typing constraints are context sensitive.

2.2.2 Definition (Type-context) A *type-context* is any finite set of type-assignments

$$\Gamma = \{x_1 : \sigma_1, \dots, x_n : \sigma_n\}$$

whose subjects are term-variables. We say a type-context is *consistent* if no term-variable in it is the subject of more than one assignment. If not specified, the type-contexts used in this dissertation are consistent.

Since a type-context is a set, it does not change when its members are permuted or repeated. For notational convenience, the following abbreviations are often used:

- $x_1 : \sigma_1, \dots, x_n : \sigma_n$ for $\{x_1 : \sigma_1, \dots, x_n : \sigma_n\}$;
- $\Gamma, x : \sigma$ for $\Gamma \cup \{x : \sigma\}$;
- $\Gamma - x : \sigma$ for $\Gamma - \{x : \sigma\}$.

Given a type-context Γ, a λ-term M and a type σ, the expression $\Gamma \rhd M : \sigma$ is called a *term M of type σ in context Γ* or a *typing judgment*. However, not all the terms of this pattern are valid. To define well-typed lambda terms of a given type, some typing rules are needed.

2.2.3 Definition (Typing rules of $\boldsymbol{\lambda^{\rightarrow}}$) Assume that a set of term variables is provided. The well-typed terms in $\lambda^{\rightarrow}$ are defined simultaneously using the following axioms and inference rules:

- *Axioms for variable* (var): for each term-variable x and each type σ,

$$\frac{}{x : \sigma \rhd x : \sigma} \ (var)$$

  It simply says that if x has type σ in the context, intuitively, then x has type σ. In other words, a variable x has any type which it is declared to have.

- *Adding variables to type context* $(add\ var)$: suppose x is not free in M,

$$\frac{\Gamma \rhd M : \tau}{\Gamma, x : \sigma \rhd M : \tau} \ (add\ var)$$

  In words, if M has type τ in context Γ, then it has type τ in context $\Gamma, x : \sigma$. This rule allows one to add an additional hypothesis to the type context.

- *$\rightarrow$-elimination* $(\rightarrow \mathrm{E})$:

$$\frac{\Gamma \rhd M : \sigma \rightarrow \tau, \quad \Gamma \rhd N : \sigma}{\Gamma \rhd MN : \tau} \ (\rightarrow \mathrm{E})$$

  It says that by applying any function of type $\sigma \rightarrow \tau$ to an argument of type σ, we obtain a result of type τ. Therefore, it is also called *function application*.

- *$\rightarrow$-introduction* $(\rightarrow \mathrm{I})$:

$$\frac{\Gamma \rhd M : \tau}{\Gamma - x : \sigma \rhd \lambda x.\, M : \sigma \rightarrow \tau} \ (\rightarrow \mathrm{I})$$

  If a term M specifies a result of type τ for all $x : \sigma$, then the expression $\lambda x.\, M : \sigma \rightarrow \tau$ defines a function of type $\sigma \rightarrow \tau$.

(examples and counter-examples)

2.2.2 Equational proof system of $\lambda^{\rightarrow}$

To derive equations of terms in $\lambda^{\rightarrow}$ that holds in all models, we need an equational proof system for $\lambda^{\rightarrow}$. A typed equation has the form $\Gamma \rhd M = N : \sigma$ where both M and N are assumed to have type σ in context Γ. Since type assignments are included in equations, we have an equational version of the typing rules that build well-typed equations in $\lambda^{\rightarrow}$.

The simply typed lambda calculus has the same theory of α-, β- and η-equivalence as the pure lambda calculus. Since α- and β-equivalence are defined by substitution, the definition of substitution is given first.

2.2.4 Definition (Substitution) We define $[N/x]M$ to be the result of substituting N for each free occurrence x in M and making any changes of bound variables needed to prevent variables free in N from becoming bound in $[N/x]M$. More precisely, we define for all x, N, P, Q and all $y \not\equiv x$

- $[N/x]x \equiv N$;
- $[N/x]y \equiv y$;
- $[N/x](PQ) \equiv ([N/x]P)([N/x]Q)$;
- $[N/x](\lambda x.P) \equiv \lambda x.P$;
- $[N/x](\lambda y.P) \equiv \lambda y.[N/x]P$ if x is free in P and y is not free in N;
- $[N/x](\lambda y.P) \equiv \lambda z.[N/x][z/y]P$ if x is free in P and y is free in N.

In the last case, z can be any variable that is not free in P and N.

For any $N_1, \ldots, N_n$ and any distinct $x_1, \ldots, x_n$, the result of simultaneously substituting all N_i for x_i $(i = 1, \ldots, n)$ in term M is defined similarly to the definition above and denoted as $[N_1/x_1, \ldots, N_n/x_n]M$.

2.2.5 Definition (α-equivalence) Given a variable y which is not free in M, we have

$$\lambda x.M =_{\alpha} \lambda y.[y/x]M,$$

and the act of replacing an occurrence of $\lambda x.M$ in a term by $\lambda y.[y/x]M$ is called a *change of bound variables*. M and N are *α-equivalent*, notation $M =_{\alpha} N$, if N results from M by a series of changes of bound variables.

2.2.6 Definition (β-equivalence)

i) A *β-redex* is any sub-term of the form $(\lambda x.M)N$. It can be reduced by the following rule

$$(\lambda x.M)N \rightarrow_{\beta} [N/x]M.$$

If P contains a β-redex $(\lambda x.M)N$ and Q is the result of replacing it by $[N/x]M$, we say P *β-contracts* to Q, denoted as $P \rightarrow_{\beta} Q$.

ii) A *β-reduction* of a term P is a finite or infinite ordered sequence of β-contractions, i.e. $P \rightarrow_{\beta} P_1 \rightarrow_{\beta} P_2 \rightarrow_{\beta} \ldots$. A finite β-reduction is *from P to Q* if it has $n \geq 1$ contractions and $P_n =_{\alpha} Q$ or it is empty and $P =_{\alpha} Q$. If there exists a reduction from P to Q, we say P *β-reduces to Q*, denoted as $P \twoheadrightarrow_{\beta} Q$. We can see that α-conversions are allowed in a β-reduction.

iii) P and Q are *β-equivalent*, notation $P =_\beta Q$, if P can be changed to Q by a finite sequence of β-reductions and β-expansions (reversed β-reductions).

A term may be able to β-reduce to different terms at the same time. For example, the term $(\lambda x.M)((\lambda y.N)P)$ can β-reduce (in one step) to $[((\lambda y.N)P)/x]M$ by substituting $((\lambda y.N)P)$ to x in M or $(\lambda x.M)([P/y]N)$ by substituting P to y in N. It is necessary for a calculus that the result of computation is independent from the order of reduction. This property holds for all λ-terms and is described in Church-Rosser Theorem for β.

2.2.7 Definition (η-equivalence) An *η-redex* is any term of form $\lambda x.Mx$ with x is not free in M. It can be reduced in accordance with the following rule

$$\lambda x.Mx \to_\eta M.$$

The definitions of *η-contracts*, *η-reduces* ($\twoheadrightarrow_\eta$), *η-equivalence* ($=_\eta$), etc. are similar to those of the corresponding β-concepts in definition 2.2.6. However, all η-reductions are finite while β-reductions may be infinite.

Similarly, the result of computation is independent from the order of η-reduction which is described in Church-Rosser Theorem for η.

Now, we can define the typing rules for typed equations in $\lambda^\to$.

2.2.8 Definition (Typing rules for typed equations in $\lambda^\to$) The typed equations in $\lambda^\to$ are generated by using the following typing rules:

- *Reflexivity* (ref):

$$\frac{}{\Gamma \rhd M = M : \sigma}\ (ref)$$

- *Symmetry* (sym):

$$\frac{\Gamma \rhd M = N : \sigma}{\Gamma \rhd N = M : \sigma}\ (sym)$$

- Transitivity $(trans)$:

$$\frac{\Gamma \rhd M = N : \sigma,\quad \Gamma \rhd N = P : \sigma}{\Gamma \rhd M = P : \sigma}\ (trans)$$

  As an equivalence relation, the equality in $\lambda^\to$ should have the three properties above, reflexivity, symmetry and transitivity.

- *Adding variables to type context* $(add\ var)$: suppose x is free in M and N,

$$\frac{\Gamma \rhd M = N : \tau}{\Gamma, x : \sigma \rhd M = N : \tau}\ (add\ var)$$

- *$\to$-elimination* $(\to \mathrm{E})$:

$$\frac{\Gamma \rhd M = M' : \sigma \to \tau,\quad \Gamma \rhd N = N' : \sigma}{\Gamma \rhd MN = M'N' : \tau}\ (\to \mathrm{E})$$

  It says that equals applied to equals yield equals, i.e. application preserves equality.

- *$\to$-introduction* $(\to \mathrm{I})$:

$$\frac{\Gamma \rhd M = N : \tau}{\Gamma - x : \sigma \rhd \lambda x : \sigma.M = \lambda x : \sigma.N : \sigma \to \tau}\ (\to \mathrm{I})$$

  This rule says that if M and N are equal for all values of x, then the two functions

$\lambda x : \sigma. M$ and $\lambda x : \sigma. N$ are equal, i.e. lambda abstraction preserves equality as well.

The three rules above can be seen as the equational versions of the typing rules corresponding to the ones for well-typed terms.

- α-equivalence (α): suppose y is not free in M,

$$\frac{}{\Gamma \rhd \lambda x : \sigma. M = \lambda y : \sigma. [y/x]M : \sigma \to \tau} \ (\alpha)$$

It allows one to rename bound variables.

- β-equivalence (β):

$$\frac{}{\Gamma \rhd (\lambda x : \sigma. M)N = [N/x]M : \tau} \ (\beta)$$

It shows how to evaluate a function application using substitution.

- η-equivalence (η): suppose x is not free in M,

$$\frac{}{\Gamma \rhd \lambda x : \sigma. Mx = M : \sigma \to \tau} \ (\eta)$$

It says that $\lambda x : \sigma. Mx$ and M define the same function, since by (β) we have $(\lambda x : \sigma. Mx)y = My$ for any argument $y : \sigma$.

2.2.3 Other types

2.2.9 Definition (Initial and terminal types) The *initial type*, denoted as $null$, is a type such that for each type σ, there is a unique term constant

$$\mathrm{Zero}^{\sigma} : null \to \sigma.$$

The *terminal type*, denoted as $unit$, is a type such that there is only one term associated with it which is

$$* : unit.$$

2.2.9 Definition (Products) If σ and τ are types then $\sigma \times \tau$ is a type, called the *product of σ and τ*. Given $M : \sigma$ and $N : \tau$, the pair $\langle M, N \rangle$ has type $\sigma \times \tau$. The projection terms $\mathrm{Proj}_1^{\sigma,\tau} : \sigma \times \tau \to \sigma$ and $\mathrm{Proj}_2^{\sigma,\tau} : \sigma \times \tau \to \tau$ return the first and second components of a pair. The typing rules for product are given as follows:

- $\times$-*introduction* $(\times \mathrm{I})$:

$$\frac{\Gamma \rhd M : \sigma, \quad \Gamma \rhd N : \tau}{\Gamma \rhd \langle M, N \rangle : \sigma \times \tau} \ (\times \mathrm{I})$$

- $\times$-*elimination* $(\times \mathrm{E})$:

$$\frac{\Gamma \rhd M : \sigma \times \tau}{\Gamma \rhd \mathrm{Proj}_1^{\sigma,\tau}\ M : \sigma} \ (\times \mathrm{E})_1 \qquad \frac{\Gamma \rhd M : \sigma \times \tau}{\Gamma \rhd \mathrm{Proj}_2^{\sigma,\tau}\ M : \tau} \ (\times \mathrm{E})_2$$

2.2.10 Definition (Sums) If σ and τ are types then $\sigma + \tau$ is a type, called the *sum of σ and τ*. The term constants associated with sums are injections $\mathrm{Inleft}^{\sigma,\tau} : \sigma \to \sigma + \tau$ and $\mathrm{Inright}^{\sigma,\tau} : \tau \to \sigma + \tau$ and case $\mathrm{Case}^{\sigma,\tau,\rho} : (\sigma + \tau) \to (\sigma \to \rho) \to (\tau \to \rho) \to \rho$. The typing rules for sums are given as follows:

- $+$-*introduction* $(+\mathrm{I})$:

$$\frac{\Gamma \rhd M : \sigma}{\Gamma \rhd \mathrm{Inleft}^{\sigma,\tau}\ M : \sigma + \tau}\ (+\mathrm{I})_1 \qquad \frac{\Gamma \rhd M : \tau}{\Gamma \rhd \mathrm{Inleft}^{\sigma,\tau}\ M : \sigma + \tau}\ (+\mathrm{I})_2$$

- $+$-*elimination* $(+\mathrm{E})$:

$$\frac{\Gamma \rhd M : \sigma + \tau,\ \Gamma \rhd N : \sigma \to \rho,\ \Gamma \rhd P : \tau \to \rho}{\Gamma \rhd \mathrm{Case}^{\sigma,\tau,\rho}\ M\ N\ P : \rho}\ (+\mathrm{E})$$

With initial type, terminal type, function types, products and sums, the type expressions in the full simply typed lambda calculus $\lambda^{null\ ,unit\ ,\to,\times,+}$ are given by the following grammar:

$\sigma\ ::=\ b \mid null \mid unit \mid \sigma \to \sigma \mid \sigma \times \sigma \mid \sigma + \sigma.$

And the terms in $\lambda^{null\ ,unit\ ,\to,\times,+}$ are given by

$M\ ::=\ \mathrm{Zero} \mid *\ \mid MM \mid \lambda x{:}\sigma.M \mid \langle M, M\rangle \mid \mathrm{Proj}_1 M \mid \mathrm{Proj}_2 M \mid$
$\mathrm{Inleft}\ M \mid \mathrm{Inright}\ M \mid \mathrm{Case}\ M\ M\ M.$

2.3 Categories

As a relatively young branch of mathematics, category theory studies in an abstract way the properties of particular mathematical structures. It seeks to express all mathematical concepts in terms of "objects" and "morphisms" independently of what they are representing. Nowadays, categories appear in most branches of mathematics and many parts of computer science. Topoi, a kind of category, can even serve as a foundation for mathematics. Cartesian closed categories can work as a framework for describing the denotational semantics of typed lambda calculus, and more generally, programming languages.

2.3.1 Categories

2.3.1 Definition (Categories) A *category* $\mathcal{C}$ consists of:

- a collection C^o of *objects*;
- a collection C^a of *morphisms* (also called *arrows or maps*) between objects, with two maps $\mathrm{dom}, \mathrm{cod} : C^a \to C^o$ which give the domain and codomain of a morphism (we write $f : a \to b$ to denote a morphism f with $\mathrm{dom}(f) = a$ and $\mathrm{cod}(f) = b$);
- a binary map "$\circ$", called composition, mapping each pair f, g of morphisms with $\mathrm{cod}(f) = \mathrm{dom}(g)$ to a morphism $g \circ f$ such that $\mathrm{dom}(g \circ f) = \mathrm{dom}(f)$ and $\mathrm{cod}(g \circ f) = \mathrm{cod}(g)$;

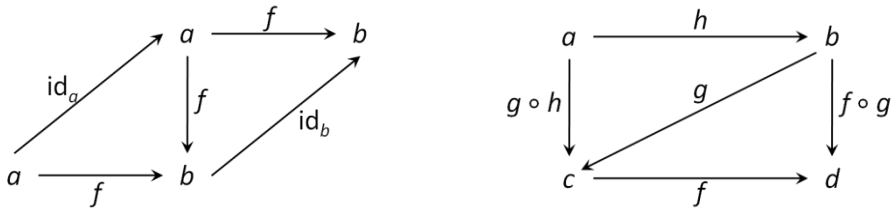
such that the following axioms hold:

- identity: for every object x, there exists a morphism $\mathrm{id}_x : x \to x$, called the identity morphism for x, such that $f = f \circ \mathrm{id}_a = \mathrm{id}_b \circ f$ for any morphism $f : a \to b$;
- associativity: $(f \circ g) \circ h = f \circ (g \circ h)$ for every $f : c \to d$, $g : b \to c$ and $h : a \to b$.

For any objects a and b of a category $\mathcal{C}$, the collection of all morphisms $f : a \to b$ is called a *hom-set* and denoted as $\mathrm{Hom}_{\mathcal{C}}(a, b)$. A category is determined by its hom-sets.

Categorists use *diagrams* to express equations. In a diagram, a morphism $f : a \to b$ is

represented as an arrow form point a to b, labeled f. A diagram *commutes* if the composition of the morphism along any path between two fixed objects is equal. The identity and associative laws in the definition of category can be represented by the following commutative diagrams:



A common example of a category is the category **SET.** It is the category whose objects are sets and whose morphisms are functions. The identity of object S in **SET** is the identity function $\mathrm{id}_S : S \to S$. The composition of morphisms is the composition of functions. As a category, it satisfies the two category axioms:

i) $f = f \circ \mathrm{id}_A = \mathrm{id}_B \circ f$, for every $f : A \to B$;

It follows by using the definition of composite function and identity function:

$(f \circ \mathrm{id}_A)(x) = f\big(\mathrm{id}_A(x)\big) = f(x)$ and $(\mathrm{id}_B \circ f)(x) = \mathrm{id}_B\big(f(x)\big) = f(x)$

ii) $(f \circ g) \circ h = f \circ (g \circ h)$, for every $f : C \to D$, $g : B \to C$ and $h : A \to B$.

This follows from the fact that composition of functions is associative:

$((f \circ g) \circ h)(x) = (f \circ g)(h(x)) = f(g(h(x))) = f((g \circ h)(x)) = (f \circ (g \circ h))(x)$

One typical use of categories is to consider categories whose objects are sets with mathematical structure and whose morphisms are functions that preserve that structure. One of the common examples is the category **POS** whose objects are posets and whose morphisms are monotone functions. It will be discussed later as an example of a CCC.

2.3.2 Categorical constructions

There are many categorical constructions, i.e. particular objects and morphisms that satisfy a given set of axioms, which enrich the language of Category Theory. When studying constructions, one observes that all concepts are defined by their relations with other objects, and these relations are established by the existence and the equality of particular morphisms. In this dissertation, the following fundamental categorical constructions will be considered.

The simplest of these is the notion of initial object and its dual, terminal object.

2.3.2 Definition (Initial and Terminal Objects) Let $\mathcal{C}$ be a category. An object a of $\mathcal{C}$ is *initial* if, for any object b in $\mathcal{C}$, there is a unique morphism from a to b. An object a in $\mathcal{C}$ is *terminal* if, for any object b in $\mathcal{C}$, there is a unique morphism from b to a.

In this dissertation, terminal objects are denoted as $unit$ and, for object a, the unique morphism is denoted as $\mathrm{One}^a : a \to unit$.

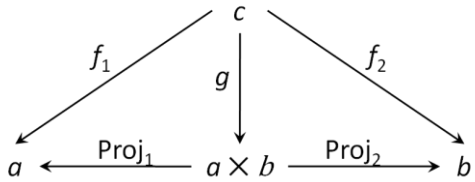
In **SET**, the initial object is the empty set $\emptyset$, and the unique morphism with $\emptyset$ for its source is the empty function whose graph is empty. Any singleton set is terminal in **SET** since for any set S, there is exactly one function from S to this singleton set.

In set theory, we can form a cartesian product of two sets and define coordinate functions for it. Then we can even form a product function of two given functions which have the same domain. This motivates a general definition of categorical products (within a category).

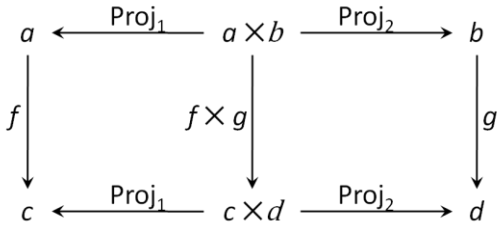
2.3.3 Definition (Products) Let a and b be objects of a category $\mathcal{C}$. The *product* of a and b is an object $a \times b$ together with two morphisms $\mathrm{Proj}_1^{a,b} : a \times b \to a$ and $\mathrm{Proj}_2^{a,b} : a \times b \to b$, and for every object c in $\mathcal{C}$, an operation $\langle \cdot , \cdot \rangle : \mathrm{Hom}(c,a) \times \mathrm{Hom}(c,b) \to \mathrm{Hom}(c, a \times b)$ such that for all morphisms $f_1 : c \to a$, $f_2 : c \to b$, and $g : c \to a \times b$, the following equations hold:

- $\mathrm{Proj}_i \circ \langle f_1, f_2 \rangle = f_i$;
- $\langle \mathrm{Proj}_1 \circ g, \mathrm{Proj}_2 \circ g \rangle = g$.

Since equations in category theory can be represented by commutative diagrams, we can give another definition of categorical products based on diagrams: Let a and b be objects of category $\mathcal{C}$. The *product* of a and b is an object $a \times b$ together with two morphisms $\mathrm{Proj}_1^{a,b} : a \times b \to a$ and $\mathrm{Proj}_2^{a,b}\ a \times b \to b$, and for every object c in $\mathcal{C}$, an operation $\langle \cdot , \cdot \rangle : \mathrm{Hom}(c,a) \times \mathrm{Hom}(c,b) \to \mathrm{Hom}(c, a \times b)$ such that for every $f_1 : c \to a$ and $f_2 : c \to b$, the morphism $\langle f_1, f_2 \rangle : c \to a \times b$ is the unique g satisfying



The cartesian product construction for morphisms can also be given a categorical definition. Given morphisms $f : a \to c$ and $g : b \to d$ the product $f \times g : a \times b \to c \times d$ is defined by $f \times g = \langle f \circ \mathrm{Proj}_1^{a,b}, g \circ \mathrm{Proj}_2^{a,b} \rangle$ whose correspondent commutative diagram is the following:



Proposition Let $\mathcal{C}$ be a category with products. Given $f_1 : b \to c$, $g_1 : a \to b$, $f_2 : b' \to c'$ and $g_2 : a' \to b'$, we have $(f_1 \times f_2) \circ (g_1 \times g_2) = (f_1 \circ g_1) \times (f_2 \circ g_2)$.

Proof

$\mathrm{Proj}_1 \circ ((f_1 \times f_2) \circ (g_1 \times g_2))$

$= (\mathrm{Proj}_1 \circ (f_1 \times f_2)) \circ (g_1 \times g_2)$
$= (f_1 \circ \mathrm{Proj}_1) \circ (g_1 \times g_2)$
$= f_1 \circ (\mathrm{Proj}_1 \circ (g_1 \times g_2))$
$= f_1 \circ (g_1 \circ \mathrm{Proj}_1)$
$= (f_1 \circ g_1) \circ \mathrm{Proj}_1$

Similarly, we have $\mathrm{Proj}_2 \circ ((f_1 \times f_2) \circ (g_1 \times g_2)) = (f_2 \circ g_2) \circ \mathrm{Proj}_2$.

By the equation $\langle \mathrm{Proj}_1 \circ g, \mathrm{Proj}_2 \circ g \rangle = g$ in definition 2.3.3,

$(f_1 \times f_2) \circ (g_1 \times g_2)$
$= \langle \mathrm{Proj}_1 \circ ((f_1 \times f_2) \circ (g_1 \times g_2)), \mathrm{Proj}_2 \circ ((f_1 \times f_2) \circ (g_1 \times g_2)) \rangle$
$= \langle (f_1 \circ g_1) \circ \mathrm{Proj}_1, (f_2 \circ g_2) \circ \mathrm{Proj}_2 \rangle$

According to the definition of products of morphisms,

$(f_1 \circ g_1) \times (f_2 \circ g_2) = \langle (f_1 \circ g_1) \circ \mathrm{Proj}_1, (f_2 \circ g_2) \circ \mathrm{Proj}_2 \rangle$.

Therefore, the equation $(f_1 \times f_2) \circ (g_1 \times g_2) = (f_1 \circ g_1) \times (f_2 \circ g_2)$ holds.

□

The products in **SET** are the cartesian product of sets. Let A and B be two sets. The cartesian product $A \times B$ is the set of pair $\langle a, b \rangle$ with $a \in A$ and $b \in B$, together with the coordinate functions $\mathrm{Proj}_1 : A \times B \to A$ and $\mathrm{Proj}_2 : A \times B \to B$ such that $\mathrm{Proj}_1(\langle a, b \rangle) = a$ and $\mathrm{Proj}_2(\langle a, b \rangle) = b$. Given two functions $f_1 : C \to A$ and $f_2 : C \to B$, the function $\langle f_1, f_2 \rangle : C \to A \times B$ is defined by $\langle f_1, f_2 \rangle(c) = (f_1(c), f_2(c))$ for all $c \in C$. Then, given $f_1 : C \to A$, $f_2 : C \to B$ and $g : C \to A \times B$, the equations in definition 2.3.3 are satisfied:

i) $(\mathrm{Proj}_i \circ \langle f_1, f_2 \rangle)(c)$
$= \mathrm{Proj}_i\ (\langle f_1, f_2 \rangle(c))$
$= \mathrm{Proj}_i\ (\langle f_1(c), f_2(c) \rangle)$
$= f_i(c)$

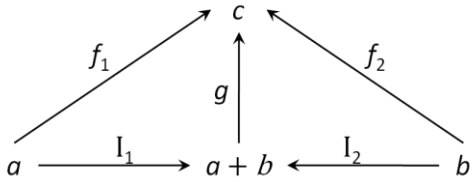
ii) $\langle \mathrm{Proj}_1 \circ g, \mathrm{Proj}_2 \circ g \rangle(c)$
$= \langle \mathrm{Proj}_1(g(c)), \mathrm{Proj}_2(g(c)) \rangle$
$= g(c)$

for all $c \in C$.

2.3.4 Definition (Coproducts) Let a and b be objects of a category $\mathcal{C}$. The *coproduct* of a and b is an object $a + b$ together with morphisms $\mathrm{I}_1^{a,b} : a \to a + b$ and $\mathrm{I}_2^{a,b} : b \to a + b$, and for every object c in $\mathcal{C}$ an operation $\langle \cdot \mid \cdot \rangle : \mathrm{Hom}(a, c) \times \mathrm{Hom}(b, c) \to \mathrm{Hom}(a + b, c)$ such that for every $f_1 : a \to c$, $f_2 : b \to c$ and $g : a + b \to c$, , the following equations hold:

- $\langle f_1 | f_2 \rangle \circ \mathrm{I}_i = f_i$;
- $\langle g \circ \mathrm{I}_1 | g \circ \mathrm{I}_2 \rangle = g$.

The corresponding commutative diagram to the equations above is shown as follows:



where $\langle f_1 | f_2 \rangle : a + b \to c$ is the unique g.

The coproducts in **SET** are the disjoint unions of sets. Let A and B be two sets. The disjoint union of them is defined by $A + B = \{(a, 1) | a \in A\} \cup \{(b, 2) | b \in B\}$ with two injection functions $\mathrm{I}_1 : A \to A + B$ that takes all a in A to $(a, 1)$ in $A + B$ and $\mathrm{I}_2 : B \to A + B$ that takes all b in B to $(b, 2)$ in $A + B$. Given two functions $f_1 : A \to C$ and $f_2 : B \to C$, the function $\langle f_1 | f_2 \rangle : A + B \to C$ is defined by $\langle f_1 | f_2 \rangle((x, i)) = f_i(x)$ for all $(x, i) \in A + B$. Given $f_1 : A \to C$, $f_2 : B \to C$ and $g : A + B \to C$, the equations in the definition above are satisfied:

i) $(\langle f_1 | f_2 \rangle \circ \mathrm{I}_i)(x)$
$= \langle f_1 | f_2 \rangle(\mathrm{I}_i(x))$
$= \langle f_1 | f_2 \rangle((x, i))$
$= f_i(x)$

ii) $\langle g \circ \mathrm{I}_1 | g \circ \mathrm{I}_2 \rangle((x, i))$
$= (g \circ \mathrm{I}_i)(x)$
$= g(\mathrm{I}_i(x))$
$= g((x, i))$

for all $(x, i) \in A + B$.

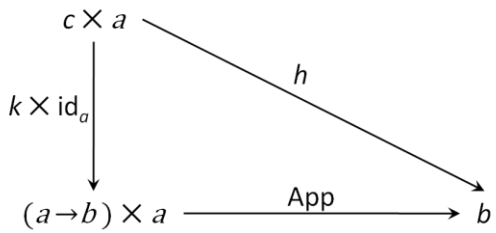
One can form a set of functions which have the same domain and codomain. Similarly, the hom-set of morphisms may form an object. This idea brings our last basic construction, exponentials.

2.3.5 Definition (Exponentials) Let $\mathcal{C}$ be a category with products for all objects, and a and b be objects of $\mathcal{C}$. The *exponential*, also called *function object*, of a and b is an object $a \to b$ together with a morphism $\mathrm{App} : (a \to b) \times a \to b$, and for every object c in $\mathcal{C}$, an operation $\mathrm{Curry} : \mathrm{Hom}(c \times a, b) \to \mathrm{Hom}(c, a \to b)$ such that for every $h : c \times a \to b$ and $k : c \to (a \to b)$, the following equations hold:

- $\mathrm{App} \circ \langle \mathrm{Curry}(h) \circ \mathrm{Proj}_1, \mathrm{Proj}_2 \rangle = h$;
- $\mathrm{Curry}(\mathrm{App} \circ \langle k \circ \mathrm{Proj}_1, \mathrm{Proj}_2 \rangle) = k$.

Using product of morphisms $f \times g = \langle f \circ \mathrm{Proj}_1, g \circ \mathrm{Proj}_2 \rangle$, the two equations above can be rewritten as $\mathrm{App} \circ (\mathrm{Curry}(h) \times \mathrm{id}) = h$ and $\mathrm{Curry}(\mathrm{App} \circ (k \times \mathrm{id})) = k$.

The commutative diagram representing the equations in the definition is shown as follows:



where the morphism $\text{Curry}(h) : c \to (a \to b)$ is the unique k.

In **SET**, the exponent set $A \to B$ of A and B is the set of functions from A to B. The function $\text{App} : (A \to B) \times A \to B$ is given by $\text{App}(\langle f, a\rangle) = f(a)$ for all $f : A \to B$ and $a \in A$. Given a function $g : C \times A \to B$, the function $\text{Curry}(g)$ is defined by $((\text{Curry}(g))(c))(a) = g(\langle c, a\rangle)$ for all $c \in C$ and $a \in A$. Given $h : C \times A \to B$ and $k : C \to (A \to B)$, the two equations hold:

i) $(\text{App} \circ \langle \text{Curry}(h) \circ \text{Proj}_1, \text{Proj}_2\rangle)(\langle c, a\rangle)$
$= \text{App}(\langle \text{Curry}(h) \circ \text{Proj}_1, \text{Proj}_2\rangle(\langle c, a\rangle))$
$= \text{App}(\langle (\text{Curry}(h) \circ \text{Proj}_1)(\langle c, a\rangle), \text{Proj}_2(\langle c, a\rangle)\rangle)$
$= \text{App}(\langle (\text{Curry}(h))(c), a\rangle)$
$= (\text{Curry}(h)(c))(a)$
$= h(\langle c, a\rangle)$

ii) $((\text{Curry}(\text{App} \circ \langle k \circ \text{Proj}_1, \text{Proj}_2\rangle))(c))(a)$
$= (\text{App} \circ \langle k \circ \text{Proj}_1, \text{Proj}_2\rangle)(\langle c, a\rangle)$
$= \text{App}(\langle k \circ \text{Proj}_1, \text{Proj}_2\rangle(\langle c, a\rangle))$
$= \text{App}(\langle (k \circ \text{Proj}_1)(\langle c, a\rangle), \text{Proj}_2(\langle c, a\rangle)\rangle)$
$= \text{App}(\langle k(c), a\rangle)$
$= (k(c))(a)$

for all $a \in A$ and $c \in C$.

2.3.3 Cartesian closed categories

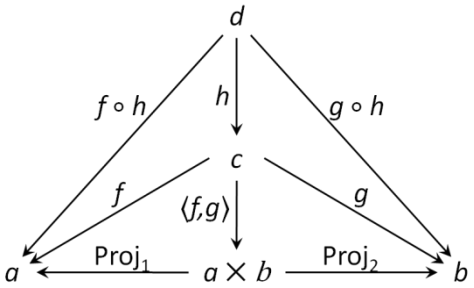
Both products and exponentials have special importance for theories of computation. A two-argument function can be reduced to a one-argument function yielding a function from the second argument to the result. This passage is called *currying*. And exponentials give a categorical interpretation to the notion of currying. Therefore, categories with products and exponentials for every pair of objects are important enough to deserve a special name.

2.3.6 Definition (Cartesian Closed Categories) A category $\mathcal{C}$ is *cartesian closed* iff

- It contains a terminal object $unit$;
- For all objects a and b in $\mathcal{C}$, there is a product;
- For all objects a and b in $\mathcal{C}$, there is an exponential.

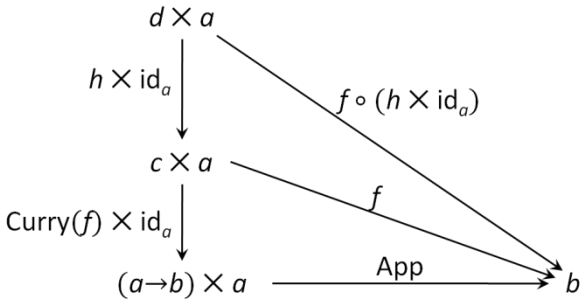
Proposition The following two useful identities hold in all cartesian closed categories:

i) $\langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle$
where $f : c \to a$, $g : c \to b$ and $h : d \to c$;



ii) $\mathrm{Curry}(f) \circ h = \mathrm{Curry}(f \circ (h \times \mathrm{id}_a))$

where $f : a \times b \to c$ and $h : d \to a$.



Proof

These two equations have been proved by the diagrams following them. But they can also be proved by the equations given in definition 2.3.3 and definition 2.3.5:

i) $\mathrm{Proj}_1 \circ (\langle f, g\rangle \circ h) = (\mathrm{Proj}_1 \circ \langle f, g\rangle) \circ h = f \circ h$

$\mathrm{Proj}_2 \circ (\langle f, g\rangle \circ h) = (\mathrm{Proj}_2 \circ \langle f, g\rangle) \circ h = g \circ h$

$\langle \mathrm{Proj}_1 \circ (\langle f, g\rangle \circ h), \mathrm{Proj}_2 \circ (\langle f, g\rangle \circ h)\rangle = \langle f \circ h, g \circ h\rangle$

Since we have $\langle \mathrm{Proj}_1 \circ g, \mathrm{Proj}_2 \circ g\rangle = g$, then $\langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle$ holds.

ii) $f = \mathrm{App} \circ (\mathrm{Curry}(f) \times \mathrm{id}_a))$ (by $\mathrm{App} \circ (\mathrm{Curry}(h) \times \mathrm{id})) = h$)

$\mathrm{Curry}(f \circ (h \times \mathrm{id}_a))$

$= \mathrm{Curry}((\mathrm{App} \circ (\mathrm{Curry}(f) \times \mathrm{id}_a))) \circ (h \times \mathrm{id}_a))$

$= \mathrm{Curry}(\mathrm{App} \circ ((\mathrm{Curry}(f) \times \mathrm{id}_a)) \circ (h \times \mathrm{id}_a)))$

(by $(f_1 \times f_2) \circ (g_1 \times g_2) = (f_1 \circ g_1) \times (f_2 \circ g_2)$)

$= \mathrm{Curry}(\mathrm{App} \circ ((\mathrm{Curry}(f) \circ h) \times (\mathrm{id}_a \circ \mathrm{id}_a)))$

$= \mathrm{Curry}(\mathrm{App} \circ ((\mathrm{Curry}(f) \circ h) \times \mathrm{id}_a))$

(by $\mathrm{Curry}(\mathrm{App} \circ (k \times \mathrm{id})) = k$)

$= \mathrm{Curry}(f) \circ h$

□

The following gives some examples and one non-example of CCCs.

(1) **SET**

SET has been already given as an example of each categorical construction. It is cartesian closed since it satisfies the three conditions:

- Any singleton set can be the terminal object (it does not matter which one it exactly is

since all the singleton sets are isomorphic);

- The product of sets A and B is the cartesian product of A and B;
- The exponential of sets A and B is the set of functions from A to B.

(2) **POS**

POS is the category whose objects are posets and whose morphisms are monotone maps. The identity of object $\mathcal{A} = (A, \leq_A)$ in **POS** is the identity map $\mathrm{id}^A : A \to A$ such that $\mathrm{id}^A(a) = a$ for all $a \in A$. The composition of morphisms is the composition of maps which still preserves the monotonicity property.

POS is cartesian closed:

- Any singleton poset can be the terminal object $unit$. The map $\mathrm{One}^A : A \to unit$ from any poset $\mathcal{A} = (A, \leq_A)$ in **POS** to this singleton poset is unique.
- The product $\mathcal{A} \times \mathcal{B} = (A \times B, \leq_{A\times B})$ of posets $\mathcal{A} = (A, \leq_A)$ and $\mathcal{B} = (B, \leq_B)$ is the cartesian product of set A and B with the ordering $\leq_{A\times B}$ which is defined by $(a, b) \leq_{A\times B} (a', b')$ iff $a \leq_A a'$ and $b \leq_B b'$. The projections are the coordinate functions $\mathrm{Proj}_1^{A,B} : A \times B \to A$ and $\mathrm{Proj}_2^{A,B} : A \times B \to B$. Given two monotone maps $f_1 : A \to B$ and $f_2 : A \to C$, the map $\langle f_1, f_2\rangle : A \to B \times C$ is defined by $\langle f_1, f_2\rangle(a) = (f_1(a), f_2(a))$ for all $a \in A$. Suppose $a, a' \in A$ and $a \leq a'$, then $f_1(a) \leq f_1(a')$ and $f_2(a) \leq f_2(a')$. According to the definition of $\langle\cdot,\cdot\rangle$, we have $\langle f_1, f_2\rangle(a) \leq \langle f_1, f_2\rangle(a')$, therefore, the map $\langle f_1, f_2\rangle$ is monotone. In the same way as **SET**, **POS** satisfies the equations in definition 2.3.3.
- The object $\mathcal{A} \to \mathcal{B} = (A \to B, \leq_{A\to B})$ is a poset consisting of the set of monotone maps from A to B and a ordering $\leq_{A\to B}$ defined by, given $f, f' : A \to B$, $f \leq_{A\to B} f'$ iff $f(a) \leq_B f'(a)$ for all $a \in A$. Given $g : A \times B \to C$, the map $\mathrm{Curry}(g) : A \to (B \to C)$ is defined by $((\mathrm{Curry}(g))(a))(b) = g((a, b))$ for all $a \in A$ and $b \in B$. Suppose $a, a' \in A$, $b \in B$ and $a \leq_A a'$, we have $(\mathrm{Curry}(g))(a) \leq_{B\to C} (\mathrm{Curry}(g))(a')$ because $((\mathrm{Curry}(g))(a))(b) = g((a, b)) \leq_C g(a', b) = ((\mathrm{Curry}(g))(a'))(b)$; therefore, the map $\mathrm{Curry}(g)$ is monotone. The map $\mathrm{App}^{B,C} : (B \to C) \times B \to C$ is defined by $\mathrm{App}((h, b)) = h(b)$ for all $h : B \to C$ and $b \in B$. Suppose $h, h' : B \to C$ and $b, b' \in B$ with $h \leq_{B\to C} h'$ and $b \leq_B b'$, then $(h, b) \leq_{(B\to C)\times B} (h', b')$ by the definition of $\leq_{(B\to C)\times B}$ and $\mathrm{App}((h, b)) = h(b) \leq_C h(b') \leq_C h'(b') = \mathrm{App}((h', b'))$; therefore, App is also monotone. **POS** satisfies the equations in definition 2.3.5. The proof is carried out in the same way of **SET**.

(3) $\mathbf{POS}_\perp$

$\mathbf{POS}_\perp$ is same as the category **POS** except that each poset has a least element $\perp$. We construct its identity, composition, terminal object, product, exponential in the same way of **POS.** Since the categorical constructions of morphism still preserve the monotonicity of morphisms; therefore, $\mathbf{POS}_\perp$ is cartesian closed.

(4) $\mathbf{POS}_\perp$! – non-example

$\mathbf{POS}_\perp$! is the category whose objects are posets with least element $\perp$ and whose

morphisms are monotone maps with the property that $\perp$ in one poset is mapped to $\perp$ in another one, i.e. for any monotone map $f : A \to B$, $f(\perp) = \perp$.

However, $\mathbf{POS}_{\perp}!$ is not cartesian closed. It has terminal object. But it does not have products and exponentials. Suppose that the products in $\mathbf{POS}_{\perp}!$ are constructed in the same way of **POS.** (the monotonicity cannot be preserved)

3. Correspondences

3.1 Every type-derivation in $\lambda^{\to}$ leads to a proof in intuitionistic implicational logic

3.2 Every proof in intuitionistic propositional logic can be encoded by a lambda term

3.3 Every well-typed lambda term can be interpreted as a morphism in a CCC

As mentioned in the previous section, cartesian closed categories can work as a more general but also more abstract framework for describing the denotational semantics of typed lambda calculus.

The lambda calculus $\lambda^{unit,\times,\to}$ has terminal type, product types and function types. Correspondently, a CCC has terminal object, products and exponentials. A closed relation between them seems to be obvious. However, the lambda calculus $\lambda^{\to}$ with only function type constructor is as expressive as $\lambda^{unit,\times,\to}$. Therefore, both the type expressions and well-typed lambda terms in $\lambda^{\to}$ can be interpreted in any CCC, and this interpretation should be sound and complete.

3.3.1 Definition (The Interpretation of Terms) Given a typed lambda calculus $\lambda^{\to}$ and a cartesian closed category $\mathcal{C}$, we choose an object of $\mathcal{C}$ for each type constant and a morphism for each term constant, and then all type expressions and typing contexts can be interpreted as objects and well-typed terms as morphisms. For notational simplicity, $\mathcal{C}[\![\ ]\!]$ is omitted to denote the interpretation of type expressions, typing contexts and terms of $\lambda^{\to}$ in $\mathcal{C}$:

(1) The interpretation $\mathcal{C}[\![\sigma]\!]$ of type expression σ is defined as follows:
- $\mathcal{C}[\![b]\!] = \hat{b}$, given as an object constant in $\mathcal{C}$;
- $\mathcal{C}[\![\sigma \to \tau]\!] = \mathcal{C}[\![\sigma]\!] \to \mathcal{C}[\![\tau]\!]$.

(2) The interpretation $\mathcal{C}[\![\Gamma]\!]$ of typing context Γ is defined by induction on the length of the context:
- $\mathcal{C}[\![\emptyset]\!] = unit$;
- $\mathcal{C}[\![\Gamma, x : \sigma]\!] = \mathcal{C}[\![\Gamma]\!] \times \mathcal{C}[\![\sigma]\!]$.

(3) The interpretation $\mathcal{C}[\![\Gamma \rhd M : \sigma]\!]$ of a well-typed term is a morphism from $\mathcal{C}[\![\Gamma]\!]$ to $\mathcal{C}[\![\sigma]\!]$ which is defined by induction on the proof of the typing judgment $\Gamma \rhd M : \sigma$:

- $\mathcal{C}[\![x:\sigma \rhd x:\sigma]\!] = \mathrm{Proj}_2^{unit,\sigma}$;
- $\mathcal{C}[\![\Gamma \rhd MN:\tau]\!] = \mathrm{App}^{\sigma,\tau} \circ \langle \mathcal{C}[\![\Gamma \rhd M:\sigma\to\tau]\!], \mathcal{C}[\![\Gamma \rhd N:\sigma]\!]\rangle$;
- $\mathcal{C}[\![\Gamma \rhd \lambda x{:}\sigma.M:\sigma\to\tau]\!] = \mathrm{Curry}(\mathcal{C}[\![\Gamma, x:\sigma \rhd M:\tau]\!])$;
- $\mathcal{C}[\![\Gamma_1, x:\sigma, \Gamma_2 \rhd M:\tau]\!] = \mathcal{C}[\![\Gamma_1,\Gamma_2 \rhd M:\tau]\!] \circ \chi_f^{[\![\Gamma_1, x:\sigma,\Gamma_2]\!]}$

  where $(\Gamma_1, x{:}\sigma, \Gamma_2)_f = \Gamma_1,\Gamma_2$ and $\Gamma_1 \cup \Gamma_2$ contains all the free variables of M.

In this definition, $f:\{1,\dots,m\}\to\{1,\dots,n\}$ is an *m*,*n*-function. If $\Gamma = x_1\sigma_1,\dots,x_n\sigma_n$ is an ordered type-context of length n, the ordered type-context Γ_f of length m is defined by $\Gamma_f = x_{f(1)}{:}\sigma_{f(1)},\dots,x_{f(m)}{:}\sigma_{f(m)}$. If $\Gamma \rhd M:\sigma$ is a well-typed term and Γ_f contains all the free variables of M, then the interpretation of $\Gamma_f \rhd M:\sigma$ can be related to the one of $\Gamma \rhd M:\sigma$ by using a combination $\chi_f^{[\![\Gamma]\!]}: [\![\Gamma]\!] \to [\![\Gamma_f]\!]$ of pairing and projection functions.

Before giving the definition of the combination, we need a few notational conventions. If $h_i: a \to a_i$ is a morphism from object a to a_i, for $1 \le i \le n$, then we write $\langle h_1,\dots,h_n\rangle$ for the morphism $\langle h_1,\dots,h_n\rangle = \langle\langle\langle h_1,h_2\rangle,\dots\rangle,h_n\rangle : a \to (((a_1\times a_2)\times\dots)\times a_n)$ and $\mathrm{Proj}_i^{a_1\times a_2\times\dots\times a_n}: a_1\times a_2\times\dots\times a_n \to a_i$ for a composition of projection morphisms Proj_1 and Proj_2 so that $\mathrm{Proj}_i^{a_1\times a_2\times\dots\times a_n}\circ\langle h_1,\dots,h_n\rangle = h_i$.

Now, given an *m*,*n*-function f, we can define $\chi_f^{unit\times a_1\times\dots\times a_n}$ by

$$\chi_f^{unit\times a_1\times\dots\times a_n} = \langle \mathrm{One}^{unit\times a_1\times\dots\times a_n}, \mathrm{Proj}_{f(1)+1}^{unit\times a_1\times\dots\times a_n},\dots,\mathrm{Proj}_{f(m)+1}^{unit\times a_1\times\dots\times a_n}\rangle$$
$$: unit\times a_1\times\dots\times a_n \to unit\times a_{f(1)}\times\dots\times a_{f(m)}.$$

The terminal object $unit$ is included in the type of χ_f since the interpretation of any type-context contains $unit$.

Some lemmas are needed in the proof of soundness and completeness of the interpretation defined in 3.3.1.

The first one is the substitution lemma, which will be used in the proof of soundness later.

3.3.2 Lemma (CCC Substitution) If $\Gamma, x:\sigma \rhd M:\tau$ and $\Gamma \rhd N:\sigma$ are well-typed terms, then $[\![\Gamma \rhd [N/x]M:\tau]\!] = [\![\Gamma, x:\sigma \rhd M:\tau]\!] \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N:\sigma]\!]\rangle$.

Proof

The proof is carried out by induction on typing derivation. The base case is the one whose term is a term variable. The inductive steps have two cases: application and abstraction.

Base case $M:\tau \equiv x:\sigma$

$[\![\Gamma \rhd [N/x]M:\tau]\!] = [\![\Gamma \rhd [N/x]x:\sigma]\!] = [\![\Gamma \rhd N:\sigma]\!]$

$[\![\Gamma, x:\sigma \rhd M:\tau]\!] \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N:\sigma]\!]\rangle$

$= [\![\Gamma, x:\sigma \rhd x:\sigma]\!] \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N:\sigma]\!]\rangle$

$= \mathrm{Proj}_2^{[\![\Gamma]\!],\sigma} \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N:\sigma]\!]\rangle$ (by $\mathrm{Proj}_i \circ \langle f_1, f_2\rangle = f_i$)

$= [\![\Gamma \rhd N : \sigma]\!]$

Inductive steps

(1) Application $M : \tau \equiv M_1 M_2 : \tau_2$

Induction hypothesis:

$[\![\Gamma \rhd [N/x]M_1 : \tau_1 \to \tau_2]\!] = [\![\Gamma, x{:}\sigma \rhd M_1 : \tau_1 \to \tau_2]\!] \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle$ and $[\![\Gamma \rhd [N/x]M_2 : \tau_1]\!] = [\![\Gamma, x{:}\sigma \rhd M_2 : \tau_1]\!] \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle$.

$[\![\Gamma \rhd [N/x]M : \tau]\!] = [\![\Gamma \rhd [N/x]M_1 M_2 : \tau_2]\!]$

$= \mathrm{App}^{\tau_1,\tau_2} \circ \langle [\![\Gamma \rhd [N/x]M_1 : \tau_1 \to \tau_2]\!], [\![\Gamma \rhd [N/x]M_2 : \tau_1]\!] \rangle$

$= \mathrm{App}^{\tau_1,\tau_2} \circ \left(\langle [\![\Gamma, x{:}\sigma \rhd M_1 : \tau_1 \to \tau_2]\!], [\![\Gamma, x{:}\sigma \rhd M_2 : \tau_1]\!] \rangle \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle \right)$

$= (\mathrm{App}^{\tau_1,\tau_2} \circ \langle [\![\Gamma, x{:}\sigma \rhd M_1 : \tau_1 \to \tau_2]\!], [\![\Gamma, x{:}\sigma \rhd M_2 : \tau_1]\!] \rangle) \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle$

$= [\![\Gamma, x{:}\sigma \rhd M_1 M_2 : \tau_2]\!] \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle$

$= [\![\Gamma, x{:}\sigma \rhd M : \tau]\!] \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle$

(2) Abstraction $M : \tau \equiv \lambda y{:}\rho.M' : \rho \to \tau$

Induction hypothesis:

$[\![\Gamma, y{:}\rho \rhd [N/x]M' : \tau]\!]$

$= [\![\Gamma, y{:}\rho, x{:}\sigma \rhd M' : \tau]\!] \circ \langle id_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y{:}\rho \rhd N{:}\sigma]\!] \rangle$

$= \left([\![\Gamma, x{:}\sigma, y{:}\rho \rhd M' : \tau]\!] \circ \chi_f^{[\![\Gamma, y:\rho, x:\sigma]\!]} \right) \circ \langle id_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y{:}\rho \rhd N{:}\sigma]\!] \rangle$

$= [\![\Gamma, x{:}\sigma, y{:}\rho \rhd M' : \tau]\!] \circ \left(\chi_f^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ \langle id_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y{:}\rho \rhd N{:}\sigma]\!] \rangle \right)$

$= [\![\Gamma, x{:}\sigma, y{:}\rho \rhd M' : \tau]\!] \circ \left(\langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle \times id_{[\![y:\rho]\!]} \right)$

$[\![\Gamma \rhd [N/x]M : \tau]\!] = [\![\Gamma \rhd [N/x](\lambda y{:}\rho.M') : \rho \to \tau]\!]$

$= [\![\Gamma \rhd \lambda y{:}\rho.[N/x]M' : \rho \to \tau]\!]$

$= \mathrm{Curry}([\![\Gamma, y{:}\rho \rhd [N/x]M' : \tau]\!])$

$= \mathrm{Curry}\left([\![\Gamma, x{:}\sigma, y{:}\rho \rhd M : \tau]\!] \circ \left(\langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle \times id_{[\![y:\rho]\!]} \right) \right)$

(by $\mathrm{Curry}(f \circ (h \times \mathrm{id})) = \mathrm{Curry}(f) \circ h$)

$= \mathrm{Curry}([\![\Gamma, x{:}\sigma, y{:}\rho \rhd M : \tau]\!]) \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle$

$= [\![\Gamma, x{:}\sigma \rhd \lambda y{:}\rho.M : \rho \to \tau]\!] \circ \langle id_{[\![\Gamma]\!]}, [\![\Gamma \rhd N{:}\sigma]\!] \rangle$

□

3.3.3 Theorem (Soundness) Given any well-typed terms $\Gamma \rhd M : \sigma$ and $\Gamma \rhd N : \sigma$ with $M =_{\alpha\beta\eta} N$, then the interpretations of them are same, i.e. $[\![\Gamma \rhd M : \sigma]\!] = [\![\Gamma \rhd N : \sigma]\!]$, in every CCC.

Proof

(1) α-equivalence

During the interpretation, we can see that the names of term variables never appear in the interpretation; therefore, α-equivalence should hold in the interpretation. Here, an equational proof of a stronger form of α-equivalence is given.

If $x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M : \sigma$ and $y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N : \sigma$ are well-typed, with $N \equiv_\alpha [y_1, \ldots, y_k/x_1, \ldots, x_k]M$, then $[\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M{:}\sigma]\!] = [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N{:}\sigma]\!]$.

The proof can be carried out by induction on the term $M : \sigma$. The base case is the one when M is a variable while the inductive steps contains application and abstraction.

Base case – $M : \sigma \equiv x_i{:}\sigma_i$

$[\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M : \sigma]\!]$

$= [\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd x_i{:}\sigma_i]\!]$

$= \mathrm{Proj}_{i+1}^{unit, \sigma_1, \ldots, \sigma_k}$

$[\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N : \sigma]\!]$

$= [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd [y_1, \ldots, y_k/x_1, \ldots, x_k]M]\!]$

$= [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd [y_1, \ldots, y_k/x_1, \ldots, x_k]x_i{:}\sigma_i]\!]$

$= [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd y_i{:}\sigma_i]\!]$

$= \mathrm{Proj}_{i+1}^{unit, \sigma_1, \ldots, \sigma_k}$

$[\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd x_i{:}\sigma_i]\!] = [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd [y_1, \ldots, y_k/x_1, \ldots, x_k]x_i{:}\sigma_i]\!]$

Inductive steps:

(a) Application – $M : \sigma \equiv M_1 M_2 : \tau_2$

$N \equiv N_1 N_2$ with $N_1 =_\alpha [y_1, \ldots, y_k/x_1, \ldots, x_k]M_1$ and $N_2 =_\alpha [y_1, \ldots, y_k/x_1, \ldots, x_k]M_2$.

Induction hypothesis:

$[\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M_1 : \tau_1 \to \tau_2]\!] = [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N_1 : \tau_1 \to \tau_2]\!]$ and

$[\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M_2 : \tau_1]\!] = [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N_2 : \tau_1]\!]$

$[\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M : \sigma]\!]$

$= [\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M_1 M_2 : \tau_2]\!]$

$= \mathrm{App}^{\tau_1, \tau_2} \circ \langle [\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M_1 : \tau_1 \to \tau_2]\!], [\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M_2 : \tau_1]\!] \rangle$

$= \mathrm{App}^{\tau_1, \tau_2} \circ \langle [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N_1 : \tau_1 \to \tau_2]\!], [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N_2 : \tau_1]\!] \rangle$

$= [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N_1 N_2 : \tau_2]\!]$

$= [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N : \sigma]\!]$

(b) Abstraction – $M : \sigma \equiv \lambda x_{k+1}{:}\sigma_{k+1}.M' : \sigma_{k+1} \to \sigma$

$N \equiv \lambda y_{k+1}{:}\sigma_{k+1}.N'$ with $N' = [y_1, \ldots, y_{k+1}/x_1, \ldots, x_{k+1}]M'$

Induction hypothesis:

$[\![x_1{:}\sigma_1, \ldots, x_{k+1}{:}\sigma_{k+1} \rhd M' : \sigma]\!] = [\![y_1{:}\sigma_1, .., y_{k+1}{:}\sigma_{k+1} \rhd N' : \sigma]\!]$

$[\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd M : \sigma]\!]$

$= [\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k \rhd \lambda x_{k+1}{:}\sigma_{k+1}.M' : \sigma_{k+1} \to \sigma]\!]$

$= \mathrm{Curry}([\![x_1{:}\sigma_1, \ldots, x_k{:}\sigma_k, x_{k+1}{:}\sigma_{k+1} \rhd M' : \sigma]\!])$

$= \mathrm{Curry}([\![y_1{:}\sigma_1, \ldots, y_k{:}\sigma_k, y_{k+1}{:}\sigma_{k+1} \rhd N' : \sigma]\!])$

$= [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd \lambda y_{k+1}{:}\sigma_{k+1}.N' : \sigma_{k+1} \to \sigma]\!]$

$= [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N : \sigma]\!]$

Therefore, $[\![x_1{:}\sigma_1, \dots, x_k{:}\sigma_k \rhd M{:}\sigma]\!] = [\![y_1{:}\sigma_1, .., y_k{:}\sigma_k \rhd N{:}\sigma]\!]$.

(2) β-equivalence

We can represent β-equivalence in the following form:

$\Gamma \rhd (\lambda x{:}\sigma.M)N =_\beta [N/x]M : \tau \Longrightarrow [\![\Gamma \rhd (\lambda x{:}\sigma.M)N : \tau]\!] = [\![\Gamma \rhd [N/x]M : \tau]\!]$

$[\![\Gamma \rhd (\lambda x{:}\sigma.M)N : \tau]\!]$

$= \mathrm{App} \circ \langle [\![\Gamma \rhd \lambda x{:}\sigma.M : \sigma \to \tau]\!], [\![\Gamma \rhd N : \sigma]\!] \rangle$

$= \mathrm{App} \circ \langle \mathrm{Curry}([\![\Gamma, x{:}\sigma \rhd M : \tau]\!]), [\![\Gamma \rhd N : \sigma]\!] \rangle$ (by $\langle f, g\rangle = (f \times id) \circ \langle id, g\rangle$)

$= \mathrm{App} \circ (\mathrm{Curry}([\![\Gamma, x{:}\sigma \rhd M : \tau]\!]) \times id) \circ \langle id, [\![\Gamma \rhd N : \sigma]\!] \rangle$

(by $\mathrm{App} \circ (\mathrm{Curry}(h) \times id) = h$)

$= [\![\Gamma, x{:}\sigma \rhd M : \tau]\!] \circ \langle \mathrm{id}, [\![\Gamma \rhd N : \sigma]\!] \rangle$ (by Substitution Lemma)

$= [\![\Gamma \rhd [N/x]M : \tau]\!]$

(3) η-equivalence

We can represent η-equivalence in the following form: given x is not free in M,

$\Gamma \rhd \lambda x{:}\sigma.Mx =_\eta M : \sigma \to \tau \Longrightarrow [\![\Gamma \rhd \lambda x{:}\sigma.Mx : \sigma \to \tau]\!] = [\![\Gamma \rhd M : \sigma \to \tau]\!]$

$[\![\Gamma \rhd \lambda x{:}\sigma.Mx : \sigma \to \tau]\!]$

$= \mathrm{Curry}([\![\Gamma, x{:}\sigma \rhd Mx : \tau]\!])$

$= \mathrm{Curry}(\mathrm{App} \circ \langle [\![\Gamma, x{:}\sigma \rhd M : \sigma \to \tau]\!], [\![\Gamma, x{:}\sigma \rhd x{:}\sigma]\!] \rangle)$

(by $[\![\Gamma, x{:}\sigma \rhd M : \tau]\!] = [\![\Gamma \rhd M : \tau]\!] \circ \mathrm{Proj}_1^{[\![\Gamma]\!],\sigma}$ and $[\![\Gamma, x{:}\sigma \rhd x{:}\sigma]\!] = \mathrm{Proj}_2^{[\![\Gamma]\!],\sigma}$)

$= \mathrm{Curry}\left(\mathrm{App} \circ \langle [\![\Gamma \rhd M : \sigma \to \tau]\!] \circ \mathrm{Proj}_1^{[\![\Gamma]\!],\sigma}, \mathrm{Proj}_2^{[\![\Gamma]\!],\sigma} \rangle\right)$

(by $\mathrm{Curry}(\mathrm{App} \circ \langle k \circ \mathrm{Proj}_1, \mathrm{Proj}_2\rangle) = k$)

$= [\![\Gamma \rhd M : \sigma \to \tau]\!]$

□

3.3.4 Theorem (Completeness) Given any well-typed terms $\Gamma \rhd M{:}\sigma$ and $\Gamma \rhd N : \sigma$, there exists a CCC $\mathcal{C}$ such that if $\mathcal{C}[\![\Gamma \rhd M{:}\sigma]\!] = \mathcal{C}[\![\Gamma \rhd N : \sigma]\!]$, then $\Gamma \rhd M =_{\alpha\beta\eta} N : \sigma$.

Proof

The category $\mathcal{C}$ is generated by $\lambda^\to$ in the following way:

The objects of $\mathcal{C}$ are sequences of type expressions. To be specific, the empty sequence is the terminal object $unit$ and a sequence $\sigma_1, \sigma_2, \dots, \sigma_n$ is the product of these types. For notational convenience, we write $\vec{x}_k$ for a sequence $x_1, x_2, \dots, x_k$ of k variables, similarly $\vec{\sigma}_k$ for a sequence of k type expressions, and $\vec{x}_k{:}\vec{\sigma}_k$ for typing context $x_1 : \sigma_1, \dots, x_k{:}\sigma_k$.

The morphisms from $\vec{\sigma}_m$ to $\vec{\tau}_n$ are given by n-tuples of terms over m free variables.

To put it more specifically, a morphism $\vec{\sigma}_m \to \vec{\tau}_n$ is a tuple of equivalence classes of terms,

$$[\vec{x}_m{:}\vec{\sigma}_m \rhd M_{\boldsymbol{i}} : \tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n]$$
$$= [\vec{x}_m{:}\vec{\sigma}_m \rhd M_1 : \tau_1, \dots, \vec{x}_m{:}\vec{\sigma}_m \rhd M_n : \tau_n]$$
$$= \{\langle \vec{x}_m{:}\vec{\sigma}_m \rhd N_1 : \tau_1, \dots, \vec{x}_m{:}\vec{\sigma}_m \rhd N_n : \tau_n\rangle \mid \vec{x}_m{:}\vec{\sigma}_m \rhd M_i : \tau_i = N_i : \tau_i \text{ for } i = 1, \dots, n\}.$$

Composition in $\mathcal{C}$ is defined by substitution. Given morphisms $\vec{\tau}_m \to \vec{\rho}_n$ and $\vec{\sigma}_l \to \vec{\tau}_m$, the composition of them is

$$[\vec{y}_m{:}\vec{\tau}_m \rhd N_{\boldsymbol{i}} : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n] \circ [\vec{x}_l{:}\vec{\sigma}_l \rhd M_{\boldsymbol{i}} : \tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m]$$
$$= \left[\vec{x}_l{:}\vec{\sigma}_l \rhd \left[\vec{y}_m/\vec{M}_m\right]N_{\boldsymbol{i}} : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] : \vec{\sigma}_l \to \vec{\rho}_n$$

(1) $\mathcal{C}$ is cartesian closed

The cartesian closed structure of $\mathcal{C}$ is obtained as follows:

i) Terminal object $unit$ with unique morphism One

As mentioned above, the empty sequence of types is the terminal object $unit$. According to the definition of morphisms above, a morphism from an object to $\vec{\tau}_n$ is given by n-tuples of terms. Then, the morphism from an object to $unit$, the empty sequence, should be given by the empty tuple. For every object $\vec{\sigma}_k$, the morphism $\mathrm{One} : \vec{\sigma}_k \to unit$ is defined by the empty tuple, i.e. $\mathrm{One} = [\,]$. Clearly, it is unique for every object $\vec{\sigma}_k$.

ii) Product objects with projection morphisms and function $\langle\cdot,\cdot\rangle$

Given two objects $\vec{\sigma}_m$ and $\vec{\tau}_n$, the product of them is obtained by their concatenation, $\sigma_1, \dots, \sigma_m, \tau_1, \dots, \tau_n$. Then, the projection morphisms is given by

$$\mathrm{Proj}_1^{\vec{\sigma}_m,\vec{\tau}_n} = [\vec{x}_m{:}\vec{\sigma}_m, \vec{y}_n{:}\vec{\tau}_n \rhd x_{\boldsymbol{i}}{:}\sigma_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] : \vec{\sigma}_m \times \vec{\tau}_n \to \vec{\sigma}_m$$
$$\mathrm{Proj}_2^{\vec{\sigma}_m,\vec{\tau}_n} = [\vec{x}_m{:}\vec{\sigma}_m, \vec{y}_n{:}\vec{\tau}_n \rhd y_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] : \vec{\sigma}_m \times \vec{\tau}_n \to \vec{\tau}_n$$

And, given $[\vec{x}_l{:}\vec{\sigma}_l \rhd M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] : \vec{\sigma}_l \to \vec{\tau}_m$ and $[\vec{x}_l{:}\vec{\sigma}_l \rhd N_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n] : \vec{\sigma}_l \to \vec{\rho}_n$, the product function $\langle\cdot,\cdot\rangle$ is defined by the concatenation of tuples:

$$\langle[\vec{x}_l{:}\vec{\sigma}_l \rhd M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m], [\vec{x}_l{:}\vec{\sigma}_l \rhd N_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n]\rangle$$
$$= [\vec{x}_l{:}\vec{\sigma}_l \rhd M_1{:}\tau_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd M_m{:}\tau_m, \vec{x}_l{:}\vec{\sigma}_l \rhd N_1{:}\rho_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd N_n{:}\rho_n] : \vec{\sigma}_l \to \vec{\tau}_m \times \vec{\rho}_n$$

The equations in definition 2.3.3 are satisfied. For any

$f_1 = [\vec{x}_l{:}\vec{\sigma}_l \rhd M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] : \vec{\sigma}_l \to \vec{\tau}_m$,

$f_2 = [\vec{x}_l{:}\vec{\sigma}_l \rhd N_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n]{:}\vec{\sigma}_l \to \vec{\rho}_n$, and

$g = [\vec{x}_l{:}\vec{\sigma}_l \rhd M_1{:}\tau_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd M_m{:}\tau_m, \vec{x}_l{:}\vec{\sigma}_l \rhd N_1{:}\rho_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd N_n{:}\rho_n] : \vec{\sigma}_l \to \vec{\tau}_m \times \vec{\rho}_n$,

we have

$$\mathrm{Proj}_1^{\vec{\tau}_m,\vec{\rho}_n} \circ \langle f_1, f_2\rangle$$
$$= [\vec{y}_m{:}\vec{\tau}_m, \vec{z}_n{:}\vec{\rho}_n \rhd y_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] \circ$$
$$\langle[\vec{x}_l{:}\vec{\sigma}_l \rhd M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m], [\vec{x}_l{:}\vec{\sigma}_l \rhd N_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n]\rangle$$
$$= [\vec{y}_m{:}\vec{\tau}_m, \vec{z}_n{:}\vec{\rho}_n \rhd y_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] \circ$$

$$[\vec{x}_l{:}\vec{\sigma}_l \rhd M_1{:}\tau_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd M_m{:}\tau_m, \vec{x}_l{:}\vec{\sigma}_l \rhd N_1{:}\rho_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd N_n{:}\rho_n]$$

$$= \left[\vec{x}_l{:}\vec{\sigma}_l \rhd \left[\vec{y}_m, \vec{z}_n/\vec{M}_m, \vec{N}_n\right] y_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m\right]$$

$$= [\vec{x}_l{:}\vec{\sigma}_l \rhd M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m]$$

$$= f_1$$

Similarly, we have $\mathrm{Proj}_2^{\vec{\tau}_m, \vec{\rho}_n} \circ \langle f_1, f_2 \rangle = f_2$

$$\langle \mathrm{Proj}_1^{\vec{\tau}_m, \vec{\rho}_n} \circ g, \mathrm{Proj}_2^{\vec{\tau}_m, \vec{\rho}_n} \circ g \rangle$$

$$= \langle [\vec{y}_m{:}\vec{\tau}_m, \vec{z}_n{:}\vec{\rho}_n \rhd y_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] \circ$$

$$[\vec{x}_l{:}\vec{\sigma}_l \rhd M_1{:}\tau_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd M_m{:}\tau_m, \vec{x}_l{:}\vec{\sigma}_l \rhd N_1{:}\rho_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd N_n{:}\rho_n], \mathrm{Proj}_2^{\vec{\tau}_m, \vec{\rho}_n} \circ g \rangle$$

$$= \langle \left[\vec{x}_l{:}\vec{\sigma}_l \rhd \left[\vec{y}_m, \vec{z}_n/\vec{M}_m, \vec{N}_n\right] y_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m\right], \mathrm{Proj}_2^{\vec{\tau}_m, \vec{\rho}_n} \circ g \rangle$$

$$= \langle [\vec{x}_l{:}\vec{\sigma}_l \rhd M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m], \mathrm{Proj}_2^{\vec{\tau}_m, \vec{\rho}_n} \circ g \rangle$$

$$= \langle [\vec{x}_l{:}\vec{\sigma}_l \rhd M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m], [\vec{x}_l{:}\vec{\sigma}_l \rhd N_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n] \rangle$$

$$= [\vec{x}_l{:}\vec{\sigma}_l \rhd M_1{:}\tau_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd M_m{:}\tau_m, \vec{x}_l{:}\vec{\sigma}_l \rhd N_1{:}\rho_1, \dots, \vec{x}_l{:}\vec{\sigma}_l \rhd N_n{:}\rho_n]$$

$$= g$$

iii) Exponentials with function Curry and morphism App

The exponential of objects $\vec{\sigma}_m$ and $\vec{\tau}_n$ is the sequence of types of the following pattern $\vec{\sigma}_m \to \vec{\tau}_n = \sigma_1 \to \cdots \to \sigma_m \to \tau_1, \dots, \sigma_1 \to \cdots \to \sigma_m \to \tau_n$.

Given any $h = [\vec{x}_l{:}\vec{\sigma}_l, \vec{y}_m{:}\vec{\tau}_m \rhd M_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n] : \vec{\sigma}_l \times \vec{\tau}_m \to \vec{\rho}_n$, the morphism $\mathrm{Curry}(h)$ is defined by

$$\mathrm{Curry}(h)$$

$$= \mathrm{Curry}([\vec{x}_l{:}\vec{\sigma}_l, \vec{y}_m{:}\vec{\tau}_m \rhd M_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n])$$

$$= [\vec{x}_l{:}\vec{\sigma}_l \rhd \lambda \vec{y}_m{:}\vec{\tau}_m . M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \to \cdots \to \tau_m \to \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n] : \vec{\sigma}_l \to (\vec{\tau}_m \to \vec{\rho}_n)$$

For any $\vec{\sigma}_m$ and $\vec{\tau}_n$, the morphism $\mathrm{App}^{\vec{\sigma}_m, \vec{\tau}_n}$ is defined by

$$\mathrm{App}^{\vec{\sigma}_m, \vec{\tau}_n}$$

$$= \left[\vec{f}_n{:}\vec{\sigma}_m \to \vec{\tau}_n, \vec{x}_m{:}\vec{\sigma}_m \rhd f_{\boldsymbol{i}}\, x_1 \dots\, x_m : \tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] : (\vec{\sigma}_m \to \vec{\tau}_n) \times \vec{\sigma}_m \to \vec{\tau}_n$$

The equations in definition 2.3.5 should hold. For any

$h = [\vec{x}_l{:}\vec{\sigma}_l, \vec{y}_m{:}\vec{\tau}_m \rhd M_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n] : \vec{\sigma}_l \times \vec{\tau}_m \to \vec{\rho}_n$ and

$k = [\vec{x}_l{:}\vec{\sigma}_l \rhd \lambda \vec{y}_m{:}\vec{\tau}_m . M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \to \cdots \to \tau_m \to \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n] : \vec{\sigma}_l \to (\vec{\tau}_m \to \vec{\rho}_n)$,

we have

$$\mathrm{App}^{\vec{\tau}_m, \vec{\rho}_n} \circ \langle \mathrm{Curry}(h) \circ \mathrm{Proj}_1^{\vec{\sigma}_l, \vec{\tau}_m}, \mathrm{Proj}_2^{\vec{\sigma}_l, \vec{\tau}_m} \rangle$$

$$= \left[\vec{f}_n{:}\vec{\tau}_m \to \vec{\rho}_n, \vec{z}_m{:}\vec{\tau}_m \rhd f_{\boldsymbol{i}}\, z_1 \dots\, z_m{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] \circ$$

$$\langle \mathrm{Curry}([\vec{x}_l{:}\vec{\sigma}_l, \vec{y}_m{:}\vec{\tau}_m \rhd M_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n]) \circ [\vec{x}_l{:}\vec{\sigma}_l, \vec{y}_m{:}\vec{\tau}_m \rhd x_{\boldsymbol{i}}{:}\sigma_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, l],$$

$$[\vec{x}_l{:}\vec{\sigma}_l, \vec{y}_m{:}\vec{\tau}_m \rhd y_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] \rangle$$

$$= \left[\vec{f}_n : \vec{\tau}_m \to \vec{\rho}_n, \vec{z}_m : \vec{\tau}_m \rhd f_{\boldsymbol{i}}\, z_1 \dots z_m : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] \circ$$
$$\langle [\vec{x}_l : \vec{\sigma}_l \rhd \lambda \vec{y}_m : \vec{\tau}_m . M_{\boldsymbol{i}} : \tau_{\boldsymbol{i}} \to \dots \to \tau_m \to \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n] \circ$$
$$[\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd x_{\boldsymbol{i}} : \sigma_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, l], [\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_{\boldsymbol{i}} : \tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] \rangle$$

$$= \left[\vec{f}_n : \vec{\tau}_m \to \vec{\rho}_n, \vec{z}_m : \vec{\tau}_m \rhd f_{\boldsymbol{i}}\, z_1 \dots z_m : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] \circ$$
$$\langle [\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd [\vec{x}_l / \vec{x}_l](\lambda \vec{y}_m : \vec{\tau}_m . M_{\boldsymbol{i}}) : \tau_1 \to \dots \to \tau_m \to \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n],$$
$$[\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_1 : \tau_1, \dots, \vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_m : \tau_m] \rangle$$

$$= \left[\vec{f}_n : \vec{\tau}_m \to \vec{\rho}_n, \vec{z}_m : \vec{\tau}_m \rhd f_{\boldsymbol{i}}\, z_1 \dots z_m : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] \circ$$
$$\langle [\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd \lambda \vec{y}_m : \vec{\tau}_m . M_{\boldsymbol{i}} : \tau_1 \to \dots \to \tau_m \to \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n],$$
$$[\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_1 : \tau_1, \dots, \vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_m : \tau_m] \rangle$$

$$= \left[\vec{f}_n : \vec{\tau}_m \to \vec{\rho}_n, \vec{z}_m : \vec{\tau}_m \rhd f_{\boldsymbol{i}}\, z_1 \dots z_m : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] \circ$$
$$[\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd \lambda \vec{y}_m : \vec{\tau}_m . M_1 : \tau_1 \to \dots \to \tau_m \to \rho_1, \dots,$$
$$\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd \lambda \vec{y}_m : \vec{\tau}_m . M_n : \tau_1 \to \dots \to \tau_m \to \rho_n,$$
$$\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_1 : \tau_1, \dots, \vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_m : \tau_m]$$

$$= \left[\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd \left[\vec{f}_n, \vec{z}_m / \lambda \vec{y}_m : \vec{\tau}_m . M_1, \dots, \lambda \vec{y}_m : \vec{\tau}_m . M_n, \vec{y}_m\right](f_{\boldsymbol{i}}\, z_1 \dots z_m) : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right]$$

$$= [\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd M_{\boldsymbol{i}} : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n]$$

$$= h$$

$$\mathrm{Curry}\left(\mathrm{App}^{\vec{\tau}_m, \vec{\rho}_n} \circ \langle k \circ \mathrm{Proj}_1^{\vec{\sigma}_l, \vec{\tau}_m}, \mathrm{Proj}_2^{\vec{\sigma}_l, \vec{\tau}_m} \rangle\right)$$

$$= \mathrm{Curry}(\left[\vec{f}_n : \vec{\tau}_m \to \vec{\rho}_n, \vec{z}_m : \vec{\tau}_m \rhd f_{\boldsymbol{i}}\, z_1 \dots z_m : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] \circ$$
$$\langle [\vec{x}_l : \vec{\sigma}_l \rhd \lambda \vec{y}_m : \vec{\tau}_m . M_{\boldsymbol{i}} : \tau_{\boldsymbol{i}} \to \dots \to \tau_m \to \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n] \circ$$
$$[\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd x_{\boldsymbol{i}} : \sigma_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, l], [\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_{\boldsymbol{i}} : \tau_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, m] \rangle)$$

$$= \mathrm{Curry}(\left[\vec{f}_n : \vec{\tau}_m \to \vec{\rho}_n, \vec{z}_m : \vec{\tau}_m \rhd f_{\boldsymbol{i}}\, z_1 \dots z_m : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] \circ$$
$$\langle [\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd [\vec{x}_l / \vec{x}_l](\lambda \vec{y}_m : \vec{\tau}_m . M_{\boldsymbol{i}}) : \tau_1 \to \dots \to \tau_m \to \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n],$$
$$[\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_1 : \tau_1, \dots, \vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_m : \tau_m] \rangle)$$

$$= \mathrm{Curry}(\left[\vec{f}_n : \vec{\tau}_m \to \vec{\rho}_n, \vec{z}_m : \vec{\tau}_m \rhd f_{\boldsymbol{i}}\, z_1 \dots z_m : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] \circ$$
$$\langle [\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd \lambda \vec{y}_m : \vec{\tau}_m . M_{\boldsymbol{i}} : \tau_1 \to \dots \to \tau_m \to \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n],$$
$$[\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_1 : \tau_1, \dots, \vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_m : \tau_m] \rangle)$$

$$= \mathrm{Curry}(\left[\vec{f}_n : \vec{\tau}_m \to \vec{\rho}_n, \vec{z}_m : \vec{\tau}_m \rhd f_{\boldsymbol{i}}\, z_1 \dots z_m : \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n\right] \circ$$
$$[\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd \lambda \vec{y}_m : \vec{\tau}_m . M_1 : \tau_1 \to \dots \to \tau_m \to \rho_1, \dots,$$
$$\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd \lambda \vec{y}_m : \vec{\tau}_m . M_n : \tau_1 \to \dots \to \tau_m \to \rho_n,$$
$$\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_1 : \tau_1, \dots, \vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd y_m : \tau_m])$$

$$= \mathrm{Curry}([\vec{x}_l : \vec{\sigma}_l, \vec{y}_m : \vec{\tau}_m \rhd \left[\vec{f}_n, \vec{z}_m / \lambda \vec{y}_m : \vec{\tau}_m . M_1, \dots, \lambda \vec{y}_m : \vec{\tau}_m . M_n, \vec{y}_m\right](f_{\boldsymbol{i}}\, z_1 \dots z_m) : \rho_{\boldsymbol{i}} \mid$$

$$\boldsymbol{i} = 1, \dots, n])$$

$$= \text{Curry}([\vec{x}_l{:}\vec{\sigma}_l, \vec{y}_m{:}\vec{\tau}_m \rhd M_{\boldsymbol{i}}{:}\rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n])$$

$$= [\vec{x}_l{:}\vec{\sigma}_l \rhd \lambda\vec{y}_m{:}\vec{\tau}_m.M_{\boldsymbol{i}}{:}\tau_{\boldsymbol{i}} \to \cdots \to \tau_m \to \rho_{\boldsymbol{i}} \mid \boldsymbol{i} = 1, \dots, n]$$

$$= k$$

Therefore, $\mathcal{C}$ is cartesian closed.

(2) $\boldsymbol{\mathcal{C}[\![\Gamma \rhd M : \sigma]\!] = \mathcal{C}[\![\Gamma \rhd N : \sigma]\!] \Longrightarrow \Gamma \rhd M =_{\alpha\beta\eta} N : \sigma}$

Given any well-typed term $\Gamma \rhd M : \sigma$ in $\lambda^{\to}$, its interpretation in $\mathcal{C}$ obtained in (1) is the equivalence class of itself. Precisely,

$$\mathcal{C}[\![\Gamma \rhd M : \sigma]\!] = [\Gamma \rhd M : \sigma] = \{\langle \Gamma \rhd N : \sigma \rangle | \Gamma \rhd M : \sigma = N : \sigma\}.$$

Then, if $\Gamma \rhd M : \sigma$ and $\Gamma \rhd N : \sigma$ have the same interpretation, they should be in the same equivalence class, i.e. $\Gamma \rhd M =_{\alpha\beta\eta} N : \sigma$**.**

Therefore, given any well-typed terms $\Gamma \rhd M{:}\sigma$ and $\Gamma \rhd N : \sigma$, there exists a CCC $\mathcal{C}$ such that if $\mathcal{C}[\![\Gamma \rhd M{:}\sigma]\!] = \mathcal{C}[\![\Gamma \rhd N : \sigma]\!]$, then $\Gamma \rhd M =_{\alpha\beta\eta} N : \sigma$.

□

References

1