University of Birmingham

School of Computer Science

Final Year Project

The Curry-Howard Isomorphism

Author:
Chuangjie Xu

Supervisor:
Prof. Achim Jung

April 9, 2011

Abstract

Systems of formal logic, as encountered in *proof theory*, tightly correspond to computational calculi, as found in *type theory*, which is known as the *Curry-Howard Isomorphism*. This correspondence has been extended to cartesian closed categories, special kinds of categories in *category theory*. This project focuses on this three-way correspondence and the ways in which they connect to each other. While the correspondence looks superficially straightforward, a considerable quantity of care has to be applied to prove it in full generality. Most of the proofs given in this dissertation are carried out by induction on derivations or terms. The main conclusions drawn from this study are that one can obtain a lambda term in simply typed lambda calculus from a proof in intuitionistic propositional logic, and vice versa. In addition, cartesian closed categories can be used as a framework for describing the denotational semantics of both the simply typed lambda calculus and intuitionistic propositional logic.

Acknowledgements

I am sincerely indebted and grateful to Achim Jung, my supervisor, for his encouragement, guidance and kindness, for keeping this research-oriented project enjoyable, and for helping me develop an understanding in the subject of theoretical computer science.

My heartfelt gratitude also goes to Martin Escardo, for supervising my individual study project and organising the course of domain theory, which are greatly beneficial to this project as well as my preparation for further study in their research area.

Besides, I would like to thank the group of Mathematical Foundations of Computer Science, for giving mathematical lunch talks every week, which provides me a valuable opportunity to know what current researches the mathematicians and computer scientists in this department are doing.

Last but not least, words alone cannot express the gratitude I owe to my family, particularly to my parents, for the unconditional love and everything they have given me.

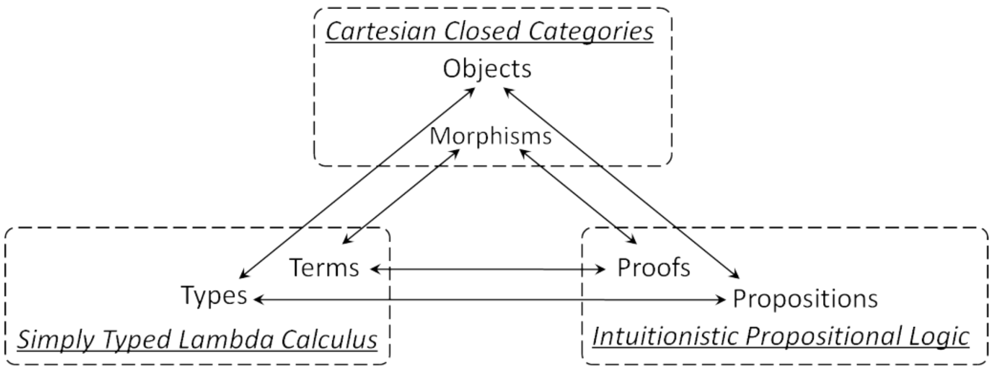
Contents

1 Introduction

Logicians must be familiar with *modus ponens*, $A \to B, A \vdash B$, a very common rule of inference, stating that given implication $A \to B$ and proposition A, one can conclude B. Programmers may frequently use *function application*: if f is a function of type $A \to B$ and x is an argument of type A, then the application fx has type B. Interestingly, modus ponens behaves the same as function application. It seems that proofs should be related to programs. Indeed, there is a precise correspondence between them, which is known as the Curry-Howard Isomorphism [10, 15].

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. As a result, this correspondence is called the Curry-Howard Isomorphism [5].

The Curry-Howard Isomorphism states a correspondence between systems of formal logic and computational calculi. 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 [1, 9, 10, 12].



Intuitionistic logic is a formalisation of Brouwers 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 formalisations of parts of intuitionism. In the 1930s, Arend Heyting, a former student of Brouwer, produced the first complete axiomatisations for intuitionistic propositional and predicate logic [13]. In intuitionistic logic, *reductio ad absurdum* is not a rule; therefore, neither of the law of excluded middle and double negation elimination is provable.

The lambda calculus was introduced by Alonzo Church in the early 1930s as a formal system to provide a functional foundation for mathematics [2, 5]. 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.

Categories first appeared in Samuel Eilenberg and Saunders Mac Lane's paper [4] 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 instance, 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 diverse 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 [8]. 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 formalisation of intuitionism, it differs from classical logic not only in that some laws in classical logic are not axioms of the system, but also in the meaning for statements to be true. The judgements about statements are based on the existence of a proof or a "construction" of that statement. This 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 intuitionistic propositional logic. Consequently, 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 $\bot, \to, \wedge, \vee$ as basic connectives and treat $\neg\varphi$ as an abbreviation for $\varphi \to \bot$.

Definition 2.1.1 (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:

$$\Phi \ ::= \ P \mid \bot \mid \neg\Phi \mid (\Phi \to \Phi) \mid (\Phi \wedge \Phi) \mid (\Phi \vee \Phi)$$

where P is a *propositional variable*, $\bot$ is *contradiction*, $\neg$ is *negation*, $\to$ is *implication*, $\wedge$ is *conjunction*, and $\vee$ is *disjunction*.

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

For notational convenience, we use the following abbreviations:

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

The *natural deduction system*, one kind of proof calculi, allows one to derive conclusions from premises. The logical reasoning in this system is expressed by inference rules which are closely related to the "natural" way of reasoning. The general form of an *inference rule* is

$$\frac{P_1, \cdots, P_n}{Q} \text{ name of the inference rule}$$

where $P_1, \cdots, P_n$ are premises and Q is the conclusion. A rule without premises is an *axiom*. Each inference rule is an atomic step in a derivation which demonstrates how the relation $\vdash$ is built.

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

- *Axiom* (*axiom*)

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

  Intuitively, one can conclude proposition φ since it already appears in the context. Someone may give it a more general form "$\Gamma, \varphi \vdash \varphi$". However, this form can be obtained by using $\varphi \vdash \varphi$ and weakening which is given below as another inference rule.

- *Adding hypotheses into context* (*add*)

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

  The property captured in this rule states that hypotheses of any derived conclusions can be extended with additional assumptions. This rule is also called *weakening*.

- $\rightarrow$*-introduction* ($\rightarrow$I)

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

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

- $\rightarrow$*-elimination* ($\rightarrow$E)

$$\frac{\Gamma_1 \vdash \varphi \rightarrow \psi \qquad \Gamma_2 \vdash \varphi}{\Gamma_1 \cup \Gamma_2 \vdash \psi}\ (\rightarrow\mathrm{E})$$

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

- $\wedge$*-introduction* ($\wedge$I)

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

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

- $\wedge$*-elimination* ($\wedge$E)

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

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

- $\vee$*-introduction* ($\vee$I)

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

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

- $\vee$*-elimination* ($\vee$E)

$$\frac{\Gamma \vdash \varphi \vee \psi \qquad \Gamma \vdash \varphi \rightarrow \rho \qquad \Gamma \vdash \psi \rightarrow \rho}{\Gamma \vdash \rho}\ (\vee\mathrm{E})$$

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

- $\perp$*-elimination* $\perp$E

$$\dfrac{\Gamma \vdash \bot}{\Gamma \vdash \varphi}\ (\bot\text{E})$$

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

What syntactically makes it different from classical propositional logic is that *reductio ad absurdum* (*RAA*) is not a rule in the natural deduction system of intuitionistic logic, which results in some theorems in classical logic being unprovable in intuitionistic logic.

2.1.2 Proofs

In intuitionistic logic, the judgements about propositions are not based on truth values. We say a proposition in intuitionistic logic holds if we can construct a *proof tree*, also called a *proof*, by using the inference rules in the natural deduction system (definition 2.1.2).

Some examples are given here to demonstrate how a proof of a proposition is built:

(1) $\varphi, \neg\varphi \vdash \bot$

$$\dfrac{\dfrac{}{\varphi \vdash \varphi} \qquad \dfrac{\dfrac{}{\neg\varphi \vdash \neg\varphi}}{\neg\varphi \vdash \varphi \to \bot}\ (\neg\varphi \text{ stands for } \varphi \to \bot)}{\varphi, \neg\varphi \vdash \bot}\ (\to\text{E})$$

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

$$\dfrac{\dfrac{\dfrac{}{\varphi, \neg\varphi \vdash \bot}\ (1)}{\varphi \vdash \neg\varphi \to \bot}\ (\to\text{I})}{\varphi \vdash \neg\neg\varphi}\ (\neg\neg\varphi \text{ stands for } \neg\varphi \to \bot)$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\psi \vdash \psi}}{\psi, \varphi \vdash \psi}\ (add)}{\psi \vdash \varphi \to \psi}\ (\to\text{I})}{\vdash \psi \to (\varphi \to \psi)}\ (\to\text{I})$$

(4) $\vdash \neg\varphi \to (\varphi \to \psi)$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\varphi, \varphi \vdash \bot}\ (1)}{\neg\varphi, \varphi \vdash \psi}\ (\bot\text{E})}{\neg\varphi \vdash \varphi \to \psi}\ (\to\text{I})}{\vdash \neg\varphi \to (\varphi \to \psi)}\ (\to\text{I})$$

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

$$\dfrac{\dfrac{\dfrac{}{\neg\varphi \vee \psi \vdash \neg\varphi \vee \psi} \qquad (3) \qquad (4)}{\neg\varphi \vee \psi \vdash \varphi \to \psi}\ (\vee\text{E})}{\vdash (\neg\varphi \vee \psi) \to (\varphi \to \psi)}\ (\to\text{I})$$

Though double negation introduction "$\varphi \to \neg\neg\varphi$" is a theorem in intuitionistic propositional logic, double negation elimination "$\neg\neg\varphi \to \varphi$" is unprovable. In classical logic, double negation elimination can be proven by using the rule *reductio ad absurdum*. However, the natural deduction system in intuitionistic propositional logic does not contain *RAA*. That's why double negation elimination cannot be proven in this system.

By the same token, the law of excluded middle “$\varphi \vee \neg\varphi$” is unprovable, either. We can also look at it from an intuitive view. A proof of disjunction $\varphi \vee \psi$ is either a proof of φ or a proof of ψ. Intuitionists only accept the proof if they know it is exactly the proof of φ or the one of ψ. For an arbitrary proposition φ, however, we do not know whether φ has a proof or $\neg\varphi$ has a proof. Therefore, we cannot give $\varphi \vee \neg\varphi$ a general proof and it cannot be proven in intuitionistic propositional logic.

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 $P \equiv P[x]$ be an expression possibly containing or depending on variable x. Then the lambda abstraction $\lambda x.P$ denotes the function $x \mapsto P[x]$. Here, P is called the *scope* of the *abstractor* λx. In a term M, if variable x occurs not in the scope of λx, we say x is *free* in M and write the set of free variables in M as $FV(M)$.

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 section 2.2.2.

2.2.1 Simply Typed Lambda Calculus $\lambda^{\rightarrow}$

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 $\lambda^{\rightarrow,\times,+}$ 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 the introduction to the other simple types can be found in section 2.2.3.

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

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

- each type-constant ι is a type, called an *atom* (or *atomic type*);
- 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. 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 λ-term can be given a type. The typing constraints are context sensitive. A *type-context* is any finite set of type-assignments $\Gamma = \{x_1{:}\sigma_1, \cdots, x_n{:}\sigma_n\}$ whose subjects are term-variables. We say a type-context is *consistent* (or *conflict free*) if no term-variable in it is the subject of more than one assignment. If not specified, the type-contexts appearing in this dissertation are consistent.

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

- $x_1{:}\sigma_1, \cdots, x_n{:}\sigma_n$ for $\{x_1{:}\sigma_1, \cdots, x_n{:}\sigma_n\}$;
- $\Gamma, x{:}\sigma$ 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 *typing judgement*, meaning that term M has type σ in context Γ. However, not all the terms of this pattern are valid. To define the set of well-typed lambda terms of a given type, some typing rules are needed.

Definition 2.2.2 (Typing Rules of $\lambda^{\rightarrow}$). 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:

- *Axiom* (*axiom*)

  For each term-variable x and each type σ,

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

  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*)

  Suppose x is not free in M and does not appear in Γ,

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

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

- $\rightarrow$*-introduction* ($\rightarrow$I)

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

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

- $\rightarrow$*-elimination* ($\rightarrow$E)

  $$\frac{\Gamma_1 \rhd M : \sigma \rightarrow \tau \qquad \Gamma_2 \rhd N : \sigma}{\Gamma_1 \cup \Gamma_2 \rhd MN : \tau}\ (\rightarrow\mathrm{E})$$

  By applying any function of type $\sigma \rightarrow \tau$ to an argument of type σ, we obtain a result of type τ. Therefore, this rule is also called *function application*.

Different literatures may represent the typing rules in different forms. For instance, someone combines (*axiom*) and (*add*) as

$$\frac{}{\Gamma, x{:}\sigma \rhd x : \sigma}$$

and does not include weakening (*add*) as a basic typing rule. But there are some valid terms that cannot be generated without this rule. To illustrate, $\lambda x{:}\sigma.\lambda x{:}\tau.x : \sigma\to(\tau\to\tau)$ is a valid closed lambda term which cannot be obtained without weakening. That's because the type derivation of it cannot start with $x{:}\sigma, x{:}\tau \rhd x : \tau$ whose context has conflict. With weakening, its derivation is built as follows.

$$\dfrac{\dfrac{\dfrac{x{:}\tau \rhd x : \tau}{\rhd \lambda x{:}\tau.x : \tau\to\tau}\ (\to\mathrm{I})}{x{:}\sigma \rhd \lambda x{:}\tau.x : \tau\to\tau}\ (add)}{\rhd \lambda x{:}\sigma.\lambda x{:}\tau.x : \sigma\to(\tau\to\tau)}\ (\to\mathrm{I})$$

The tree built by following the typing rules is called a *type-derivation*. Since a lambda term must be a term variable, an application or a lambda abstraction, and for each of them there is only one typing rule to obtain it, a well-typed lambda term should have exactly one unique type-derivation.

Proposition 2.2.3. Each well-typed lambda term M uniquely determines a type-derivation that ends in $\Gamma \rhd M : \sigma$ where Γ contains type assignments for all the free variables of M (and nothing else). [1]

Proof.

Since all the lambda terms are defined inductively, we can prove the proposition by induction on the terms.

The base case is a term variable x. As a term variable, it should have a type which it is declared to have in the type context. We give it type σ, and then have $x{:}\sigma \rhd x : \sigma$ according to (*axiom*). Therefore, the type-derivation of $x : \sigma$ is unique.

There are two inductive cases, applications and abstractions.

Assume that $M : \sigma\to\tau$ uniquely determines a type derivation ending with context Γ_M, that $N : \sigma$ uniquely determines a type derivation ending with context Γ_N, and that $\Gamma_M \cup \Gamma_N$ is consistent. To obtain an application, the only rule which can be applied to them is ($\to$E). Hence, their application $MN : \tau$ uniquely determines a type derivation that ends in $\Gamma_M \cup \Gamma_N \rhd MN : \tau$.

Similarly, let us assume that $M : \tau$ uniquely determines a type derivation which ends with context $\Gamma, x{:}\sigma$. To obtain a lambda abstraction $\lambda x{:}\sigma.M : \sigma\to\tau$, the only typing rule can be used is ($\to$I). Consequently, $\lambda x{:}\sigma.M : \sigma\to\tau$ uniquely determines a type derivation which ends in $\Gamma \rhd \lambda x{:}\sigma.M : \sigma\to\tau$.

□

As mentioned before, not every term can be typed. Take the term xx for example. As an application, its first component x should have a function type $\sigma\to\tau$ and then its second x should have type σ. In order to construct a derivation tree, we need two sub-trees, one of which ends in $\Gamma_1 \rhd x : \sigma\to\tau$ and another one ends in $\Gamma_2 \rhd x : \sigma$, in accordance with the rule ($\to$E). Since x is a term-variable, the axiom (*axiom*) tells us both $x : \sigma\to\tau$ and $x : \sigma$ should appear in context $\Gamma_1 \cup \Gamma_2$, which means that $\Gamma_1 \cup \Gamma_2$ is inconsistent. Therefore, we are unable to find a consistent context for xx and it cannot be typed.

[1] It is uniquely determined except for the occurrence of the weakening rule (*add*).

2.2.2 Equational Proof System of $\lambda^{\rightarrow}$

To derive equations of terms in $\lambda^{\rightarrow}$ that hold in all models, we need an equational proof system for $\lambda^{\rightarrow}$. A typed equation has the form $\Gamma \vdash M = N : \sigma$, 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.

Definition 2.2.4 (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 $x \in FV(P)$ and $y \notin FV(N)$;
- $[N/x](\lambda y.P) \equiv \lambda z.[N/x][z/y]P$ if $x \in FV(P)$ and $y \in FV(N)$.

In the last case, z can be any variable that $z \notin FV(P)$ and $z \notin FV(N)$.

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

Definition 2.2.5 (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.

Definition 2.2.6 (β-equivalence).

- A *β-redex* is any sub-term of the form $(\lambda x.M)N$. It can be reduced by
  $$(\lambda x.M)N \rightarrow_\beta [N/x]M$$
  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 $P \rightarrow_\beta Q$.
- A *β-reduction* of a term P is a finite or infinite ordered sequence of β-contractions, i.e. $P \rightarrow_\beta P_1 \rightarrow_\beta P_2 \rightarrow_\beta \cdots$. 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.
- 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 reversed β-reductions (also called *β-expansions*).

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 $((\lambda y.N)P)$ for x in M or $(\lambda x.M)([P/y]N)$ by substituting P for 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 stated in *Church-Rosser Theorem for β*.

Definition 2.2.7 (**η-equivalence**). An *η-redex* is any term of form $\lambda x.Mx$ with $x \notin FV(M)$. It can be reduced by

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

The definition 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 stated in *Church-Rosser Theorem for η*.

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

Definition 2.2.8 (**Typing Rules for Typed Equations in $\lambda^\rightarrow$**). The typed equations in $\lambda^\rightarrow$ are generated by using the following typing rules:

- *Reflexivity* (ref)

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

- *Symmetry* (sym)

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

- *Transitivity* ($trans$)

$$\frac{\Gamma \triangleright M = N : \sigma \qquad \Gamma \triangleright N = P : \sigma}{\Gamma \triangleright M = P : \sigma}\ (trans)$$

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

- *Adding variables to type context* (add)
  suppose x is not free in M or N,

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

  It allows one to add additional variables to the type contexts of typed equations.

- *$\rightarrow$-introduction* ($\rightarrow$I)

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

  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.

- *$\rightarrow$-elimination* ($\rightarrow$E)

$$\frac{\Gamma \triangleright M = M' : \sigma \rightarrow \tau \qquad \Gamma \triangleright N = N' : \sigma}{\Gamma \triangleright MN = M'N' : \tau}\ (\rightarrow\mathrm{E})$$

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

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,
  $$\frac{}{\Gamma \triangleright \lambda x{:}\sigma.M = \lambda y{:}\sigma.[y/x]M : \sigma \to \tau}\ (\alpha)$$
  It allows one to rename bound variables.
- *β-equivalence* (β)
  $$\frac{}{\Gamma \triangleright (\lambda x{:}\sigma.M)N = [N/x]M : \tau}\ (\beta)$$
  It shows how to evaluate a function application using substitution.
- *η-equivalence* (η)
  suppose x is not free in M,
  $$\frac{}{\Gamma \triangleright \lambda x{:}\sigma.Mx = M : \sigma{\to}\tau}\ (\eta)$$
  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 of type σ.

2.2.3 Other Types

Except function type, different versions of the simply typed lambda calculus may have products, sums, initial and terminal types. Their definitions and the typing rules for them are listed below.

Definition 2.2.9 (**Initial and Terminal Types**). The *initial type*, denoted as *null*, is an empty type (i.e. there is no instance of initial type) and, for each type σ, there is a unique term constant

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

The *terminal type*, denoted as *unit*, is a type such that there is only one term associated with it, $* : \mathit{unit}$, and, for each type σ, there is a unique term constant

$$\mathrm{One}^{\sigma} : \sigma \to \mathit{unit}$$

Definition 2.2.10 (**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 $\mathrm{Proj}_2^{\sigma,\tau} : \sigma \times \tau \to \tau$ are the coordinate functions that return the first and second components in a pair. The typing rules for products are given as follows:

- *$\times$-introduction* ($\times$I)
  $$\frac{\Gamma \triangleright M : \sigma \qquad \Gamma \triangleright N : \tau}{\Gamma \triangleright \langle M, N\rangle : \sigma \times \tau}\ (\times\mathrm{I})$$
- *$\times$-elimination* ($\times$E)
  $$\frac{\Gamma \triangleright M : \sigma \times \tau}{\Gamma \triangleright \mathrm{Proj}_1^{\sigma,\tau} M : \sigma}\ (\times\mathrm{E}_1) \qquad \frac{\Gamma \triangleright M : \sigma \times \tau}{\Gamma \triangleright \mathrm{Proj}_2^{\sigma,\tau} M : \tau}\ (\times\mathrm{E}_2)$$

Definition 2.2.11 (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 $\text{Inleft}^{\sigma,\tau} : \sigma \to \sigma + \tau$ and $\text{Inright}^{\sigma,\tau} : \tau \to \sigma + \tau$ that construct terms of type $\sigma + \tau$ from a term of type either σ or τ, and $\text{Case}^{\sigma,\tau,\rho} : (\sigma + \tau) \to (\sigma \to \rho) \to (\tau \to \rho) \to \rho$. The typing rules for sums are given as follows:

- $+$-*introduction* (+I)

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

- $+$-*elimination* (+E)

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

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

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

where b is the type constant.

The λ-terms in $\lambda^{null,unit,\to,\times,+}$ are given by

$$\begin{aligned} M \ ::= \ & \text{Zero} \mid * \mid \text{One} \mid M\ M \mid \lambda x : \sigma.M \mid \langle M, M \rangle \mid \text{Proj}_1\ M \mid \\ & \text{Proj}_2\ M \mid \text{Inleft}\ M \mid \text{Inright}\ M \mid \text{Case}\ M\ M\ M \end{aligned}$$

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 "object" and "morphisms" independently of what they are representing. Nowadays, categories appear in most branches of mathematics and many parts of computer science. For instance, topoi, a kind of category, can even serve as a foundation for mathematics. Cartesian closed categories, as another example, can work as a framework for describing the denotational semantics of typed lambda calculus, and more generally, programming languages.

2.3.1 Categories

The formal definition of categories is given first.

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

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

such that the following axioms hold:

- identity: for every object A, there exists a morphism $\mathrm{id}_A : A \to A$, called the identity morphism for A, such that $f = f \circ \mathrm{id}_A$ and $g = \mathrm{id}_A \circ g$ for any morphisms f and g with $dom(f) = cod(g) = A$;
- associativity: $h \circ (g \circ f) = (h \circ g) \circ f$ for every $f : A \to B$, $g : B \to C$, and $h : C \to D$.

For any objects A and B in 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 definition 2.3.1 can be represented by the commutative diagrams in figure 2.1.

A common example of a category is $\textsc{Set}$ which is the category whose objects are sets and whose morphisms are functions. The identity of object S in $\textsc{Set}$ is the identity function $\mathrm{id}_S : S \to S$ such that $\mathrm{id}_S(s) = s$ for all $s \in S$. The composition of morphisms is the composition of functions, i.e. $(g \circ f)(x) = g(f(x))$. As a category, it satisfies the two category axioms:

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

The identity follows by using the definitions of composite functions and identity

$$
\begin{tikzcd}
 & A \arrow[r, "f"] & B \\
C \arrow[ur, "g"] \arrow[r, "g"'] & A \arrow[u, "\mathrm{id}_A"'] \arrow[ur, "f"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "g \circ f"'] & B \arrow[dl, "g"'] \arrow[d, "h \circ g"] \\
C \arrow[r, "h"'] & D
\end{tikzcd}
$$

Figure 2.1: Diagrams of identity and associativity

functions:
$(f \circ \mathrm{id}_A)(a) = f(\mathrm{id}_A(a)) = f(a)$ and $(\mathrm{id}_B \circ f)(a) = \mathrm{id}_B(f(a)) = f(a)$.

ii) $h \circ (g \circ f) = (h \circ g) \circ f$ for every $f : A \to B$, $g : B \to C$, and $h : C \to D$
The associativity follows from the fact that composition of functions is associative:
$(h \circ (g \circ f))(a) = h((g \circ f)(a)) = h(g(f(a)))$ and
$((h \circ g) \circ f)(a) = (h \circ g)(f(a)) = h(g(f(a)))$.

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 in 2.3.3.

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

Definition 2.3.2 (Initial and terminal objects). Let $\mathcal{C}$ be a category. An object A in $\mathcal{C}$ 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 $\mathrm{One}^A : A \to \mathit{unit}$.

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).

Definition 2.3.3 (Products). Let A and B be objects in 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 every $f_1 : C \to A$, $f_2 : C \to B$, 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 in 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 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 that makes the diagram in figure 2.2 commute.

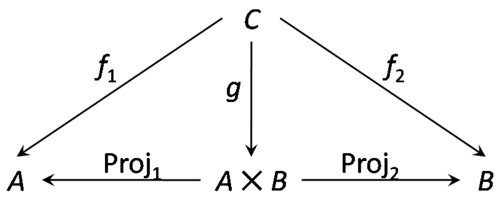


Figure 2.2: Diagram of product $A \times B$

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 given in figure 2.3.

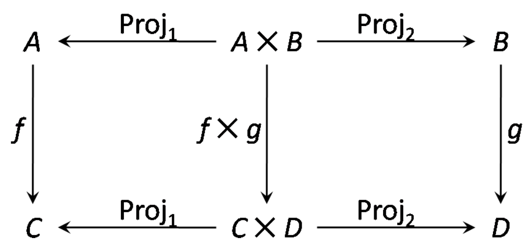


Figure 2.3: Diagram of product of morphisms

Proposition 2.3.4. Let $\mathcal{C}$ be a category with products. Given $f_1 : A \to B$, $g_1 : B \to C$, $f_2 : A' \to B'$ and $g_2 : B' \to C'$, the equation $(g_1 \times g_2) \circ (f_1 \times f_2) = (g_1 \circ f_1) \times (g_2 \circ f_2)$ holds.

Proof.

$$
\begin{aligned}
& \mathrm{Proj}_1 \circ ((g_1 \times g_2) \circ (f_1 \times f_2)) \\
= \; & (\mathrm{Proj}_1 \circ (g_1 \times g_2)) \circ (f_1 \times f_2) \\
= \; & (g_1 \circ \mathrm{Proj}_1) \circ (f_1 \times f_2) \\
= \; & g_1 \circ (\mathrm{Proj}_1 \circ (f_1 \times f_2)) \\
= \; & g_1 \circ (f_1 \circ \mathrm{Proj}_1) \\
= \; & (g_1 \circ f_1) \circ \mathrm{Proj}_1
\end{aligned}
$$

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

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

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

According to the definition of products of morphisms,

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

Therefore, the equation $(g_1 \times g_2) \circ (f_1 \times f_2) = (g_1 \circ f_1) \times (g_2 \circ f_2)$ 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 $\langle f_1, f_2 \rangle(c) = \langle f_1(c), f_2(c) \rangle$ for all $c \in C$. Then, given every $f_1 : C \to A$, $f_2 : C \to B$ and $g : C \to A \times B$, the equations in definition 2.3.3 are satisfied:

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

$$
\begin{aligned}
& (\mathrm{Proj}_i \circ \langle f_1, f_2 \rangle)(c) \\
= \; & \mathrm{Proj}_i(\langle f_1, f_2 \rangle(c)) \\
= \; & \mathrm{Proj}_i(\langle f_1(c), f_2(c) \rangle) \\
= \; & f_i(c)
\end{aligned}
$$

for any $c \in C$.

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

$$
\begin{aligned}
& \langle \mathrm{Proj}_1 \circ g, \mathrm{Proj}_2 \circ g \rangle(c) \\
= \; & \langle (\mathrm{Proj}_1 \circ g)(c), (\mathrm{Proj}_2 \circ g)(c) \rangle \\
= \; & \langle \mathrm{Proj}_1(g(c)), \mathrm{Proj}_2(g(c)) \rangle \\
= \; & g(c)
\end{aligned}
$$

for any $c \in C$.

Definition 2.3.5 (**Coproducts**)**.** Let A and B be objects in 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 $\langle \cdot | \cdot \rangle : \mathrm{Hom}(A, C) \times \mathrm{Hom}(B, C) \to \mathrm{Hom}(A + B, C)$ 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 in figure 2.4, where $\langle f_1 | f_2 \rangle$ is the unique g.

$$\begin{tikzcd}
& C & \\
A \arrow[ur, "f_1"] \arrow[r, "\mathrm{I}_1"] & A+B \arrow[u, "g"] & B \arrow[ul, "f_2"'] \arrow[l, "\mathrm{I}_2"']
\end{tikzcd}$$

Figure 2.4: Diagram of coproduct $A + B$

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 $A + B = \{\langle a, 1\rangle | a \in A\} \cup \{\langle b, 2\rangle | b \in B\}$ with two injection functions $\mathrm{I}_1 : A \to A + B$ that takes all a in A to $\langle a, 1\rangle$ in $A + B$ and $\mathrm{I}_2 : B \to A + B$ that takes all b in B to $\langle b, 2\rangle$ 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 $\langle f_1, f_2\rangle(\langle x, i\rangle) = f_i(x)$ for all $\langle x, i\rangle \in A + B$. Given any $f_1 : A \to C$, $f_2 : B \to C$ and $g : A + B \to C$, the equations in definition 2.3.5 are satisfied:

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

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

for any $\langle x, i\rangle \in A + B$.

ii) $\langle g \circ \mathrm{I}_1 | g \circ \mathrm{I}_2 \rangle = g$

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

for any $\langle x, i\rangle \in A + B$.

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.

Definition 2.3.6 (Exponentials). Let $\mathcal{C}$ be a category with products for all objects, and A and B be objects in $\mathcal{C}$. 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 the definition of products of morphisms, $f \times g = \langle f \circ \mathrm{Proj}_1, g \circ \mathrm{Proj}_2 \rangle$, the two equations in definition 2.3.6 can be rewritten as $\mathrm{App} \circ (\mathrm{Cutty}(h) \times \mathrm{id}) = h$ and $\mathrm{Curry}(\mathrm{App} \circ (k \times \mathrm{id})) = k$.

The commutative diagram representing the equations in the definition is shown in figure 2.5,where the morphism $\mathrm{Curry}(h) : C \to (A \to B)$ is the unique k.

$$
\begin{tikzcd}
C \times A \arrow[d, "k \times \mathrm{id}_A"'] \arrow[dr, "h"] & \\
(A \to B) \times A \arrow[r, "\mathrm{App}"] & B
\end{tikzcd}
$$

Figure 2.5: Diagram of exponential $A \to B$

In Set, the exponent set $A \to B$ of A and B is the set of functions from A to B. The function $\mathrm{App} : (A \to B) \times A \to B$ is given by $\mathrm{App}(\langle f, a \rangle) = f(a)$ for all $f : A \to B$ and $a \in A$. Given a function $g : C \times A \to B$, the function $\mathrm{Curry}(g)$ is defined by $((\mathrm{Curry}(g))(c))(a) = g(\langle c, a \rangle)$ for all $c \in C$ and $a \in A$. Given any $h : C \times A \to B$ and $k : C \to (A \to B)$, the two equations in definition 2.3.6 hold:

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

$$
\begin{aligned}
& (\mathrm{App} \circ \langle \mathrm{Curry}(h) \circ \mathrm{Proj}_1, \mathrm{Proj}_2 \rangle)(\langle c, a \rangle) \\
= \; & \mathrm{App}(\langle \mathrm{Curry}(h) \circ \mathrm{Proj}_1, \mathrm{Proj}_2 \rangle(\langle c, a \rangle)) \\
= \; & \mathrm{App}(\langle (\mathrm{Curry}(h) \circ \mathrm{Proj}_1)(\langle c, a \rangle), \mathrm{Proj}_2(\langle c, a \rangle) \rangle) \\
= \; & \mathrm{App}(\langle \mathrm{Curry}(h)(\mathrm{Proj}_1(\langle c, a \rangle)), a \rangle) \\
= \; & \mathrm{App}(\langle (\mathrm{Curry}(h))(c), a \rangle) \\
= \; & ((\mathrm{Curry}(h))(c))(a) \\
= \; & h(\langle c, a \rangle)
\end{aligned}
$$

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

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

$$
\begin{aligned}
& ((\mathrm{Curry}(\mathrm{App} \circ \langle k \circ \mathrm{Proj}_1, \mathrm{Proj}_2 \rangle))(c))(a) \\
= \; & (\mathrm{App} \circ \langle k \circ \mathrm{Proj}_1, \mathrm{Proj}_2 \rangle)(\langle c, a \rangle) \\
= \; & \mathrm{App}(\langle k \circ \mathrm{Proj}_1, \mathrm{Proj}_2 \rangle(\langle c, a \rangle)) \\
= \; & \mathrm{App}(\langle (k \circ \mathrm{Proj}_1)(\langle c, a \rangle), \mathrm{Proj}_2(\langle c, a \rangle) \rangle) \\
= \; & \mathrm{App}(\langle k(\mathrm{Proj}_1(\langle c, a \rangle)), a \rangle) \\
= \; & \mathrm{App}(\langle k(c), a \rangle) \\
= \; & (k(c))(a)
\end{aligned}
$$

for any $a \in A$ 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.

Definition 2.3.7 (Cartesian Closed Categories). A category $\mathcal{C}$ is *cartesian closed* iff

- it contains a terminal object *unit*;
- for every pair of objects A and B in $\mathcal{C}$, there is a product;
- for every pair of objects A and B in $\mathcal{C}$, there is an exponential.

Proposition 2.3.8. The following identities hold in every cartesian closed category:

(1) $\langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle$
where $f : C \to A$, $g : C \to B$ and $h : D \to C$;

(2) $\langle f, g\rangle = (f \times \mathrm{id}_B) \circ \langle \mathrm{id}_C, g\rangle$
where $f : C \to A$ and $g : C \to B$;

(3) $\mathrm{Curry}(f) \circ h = \mathrm{Curry}(f \circ (h \times \mathrm{id}))$
where $f : A \times B \to C$ and $h : D \to A$.

Proof. These three equations have been proved by the diagrams in figures 2.6, 2.7 , and 2.8. But they can also be proved by using the equations given in definition 2.3.3 and 2.3.6 .

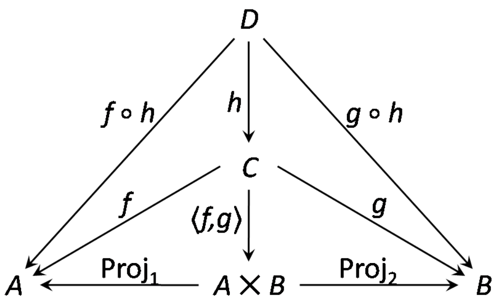


Figure 2.6: Diagram of $\langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle$

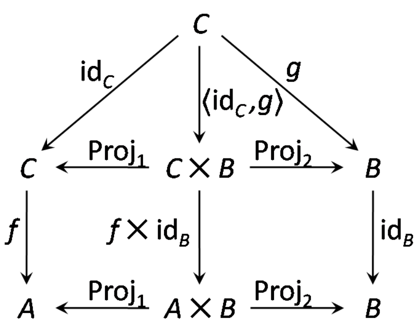


Figure 2.7: Diagram of $\langle f, g\rangle = (f \times \mathrm{id}_B) \circ \langle \mathrm{id}_C, g\rangle$

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

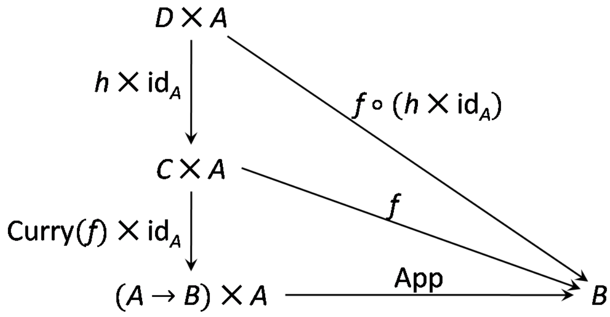


Figure 2.8: Diagram of $\mathrm{Curry}(f) \circ h = \mathrm{Curry}(f \circ (h \times \mathrm{id}))$

$\langle \mathrm{Proj}_1 \circ (\langle f, g\rangle \circ h), \mathrm{Proj}_2 \circ (\langle f, g\rangle \circ h)\rangle = \langle f \circ h, g \circ h\rangle$
Since we have $\langle \mathrm{Proj}_1 \circ g, \mathrm{Proj}_2 \circ g\rangle = g$ (in definition 2.3.3), then $\langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle$ holds.

(2) $f \times \mathrm{id}_B = \langle f \circ \mathrm{Proj}_1, \mathrm{id}_B \circ \mathrm{Proj}_2\rangle$ (by the definition of products of morphisms)

$$
\begin{aligned}
& (f \times \mathrm{id}_B) \circ \langle \mathrm{id}_C, g\rangle \\
= \; & \langle f \circ \mathrm{Proj}_1, \mathrm{id}_B \circ \mathrm{Proj}_2\rangle \circ \langle \mathrm{id}_C, g\rangle \\
= \; & \langle f \circ \mathrm{Proj}_1, \mathrm{Proj}_2\rangle \circ \langle \mathrm{id}_C, g\rangle \\
& (\text{by } \langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle \text{ in (1) }) \\
= \; & \langle f \circ \mathrm{Proj}_1 \circ \langle \mathrm{id}_C, g\rangle, \mathrm{Proj}_2 \circ \langle \mathrm{id}_C, g\rangle\rangle \\
& (\text{by } \mathrm{Proj}_i \circ \langle f_1, f_2\rangle = f_i) \\
= \; & \langle f \circ \mathrm{id}_C, g\rangle \\
= \; & \langle f, g\rangle
\end{aligned}
$$

(3) $f = \mathrm{App} \circ (\mathrm{Curry}(f) \times \mathrm{id}))$ (by $\mathrm{App} \circ (\mathrm{Cutty}(h) \times \mathrm{id}) = h$ in definition 2.3.6)

$$
\begin{aligned}
& \mathrm{Curry}(f \circ (h \times \mathrm{id})) \\
= \; & \mathrm{Curry}((\mathrm{App} \circ (\mathrm{Curry}(f) \times \mathrm{id}))) \circ (h \times \mathrm{id})) \\
= \; & \mathrm{Curry}(\mathrm{App} \circ ((\mathrm{Curry}(f) \times \mathrm{id})) \circ (h \times \mathrm{id}))) \\
& (\text{by proposition 2.3.4 } (g_1 \times g_2) \circ (f_1 \times f_2) = (g_1 \circ f_1) \times (g_2 \circ f_2)) \\
= \; & \mathrm{Curry}(\mathrm{App} \circ ((\mathrm{Curry}(f) \circ h) \times (\mathrm{id} \circ \mathrm{id}))) \\
= \; & \mathrm{Curry}(\mathrm{App} \circ ((\mathrm{Curry}(f) \circ h) \times \mathrm{id}))) \\
& (\text{by } \mathrm{Curry}(\mathrm{App} \circ (k \times \mathrm{id})) = k \text{ in definition 2.3.6}) \\
= \; & \mathrm{Curry}(f) \circ h
\end{aligned}
$$

□

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

$\textsc{Pos}$ is the category whose objects are posets and whose morphisms are monotone maps. The identity of object $\mathcal{A} = (A, \leq_A)$ in $\textsc{Pos}$ 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. Both identity and composition still preserve the monotonicity property.

$\textsc{Pos}$ is cartesian closed:

- Any singleton poset can be the terminal object *unit*. Then the map $\mathrm{One}^A : A \to unit$ from any poset $\mathcal{A}$ 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 sets A and B with the ordering $\leq_{A\times B}$ which is defined by $\langle a, b\rangle \leq_{A\times B} \langle a', b'\rangle$ iff $a \leq_A a'$ and $b \leq_B b'$.
  The projections are the coordinate functions $\mathrm{Proj}_1 : A \times B \to A$ and $\mathrm{Proj}_2 : A \times B \to B$ which just return one of the two components in the pair. So they preserve the monotonicity.
  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 $\langle f_1, f_2\rangle(a) = \langle f_1(a), f_2(a)\rangle$ for all $a \in A$. Suppose $a, a' \in A$ and $a \leq_A a'$, then $f_1(a) \leq_B f_1(a')$ and $f_2(a) \leq_C f_2(a')$. According to the above definition, we have $\langle f_1, f_2\rangle(a) \leq_{B\times C} \langle f_1, f_2\rangle(a')$. Hence, the map $\langle f_1, f_2\rangle$ is monotone.
  In the same way as $\textsc{Set}$, $\textsc{Pos}$ satisfies the equations in definition 2.3.3.
- The exponential $\mathcal{A} \to \mathcal{B} = (A \to B, \leq_{A\to B})$ of posets $\mathcal{A} = (A, \leq_A)$ and $\mathcal{B} = (B, \leq_B)$ is 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 a monotone map $g : A \times B \to C$, the map $\mathrm{Curry}(g) : A \to (B \to C)$ is defined by $((\mathrm{Curry}(g))(a))(b) = g(\langle a, b\rangle)$ 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 $((\mathrm{Curry}(g))(a))(b) = g(\langle a, b\rangle) \leq_C g(\langle a', b\rangle) = ((\mathrm{Curry}(g))(a'))(b)$ for any $b \in B$; therefore, Curry(g) is monotone.
  The map $\mathrm{App} : (B \to C) \times B \to C$ is defined by $\mathrm{App}(\langle h, b\rangle) = h(b)$ 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 $b \leq_B b'$, we have $\langle h, b\rangle \leq_{(B\to C)\times B} \langle h', b'\rangle$ by the definition of $\leq_{(B\to C)\times B}$ and then $\mathrm{App}(\langle h, b\rangle) = h(b) \leq_C h(b') \leq_C h'(b') = \mathrm{App}(\langle h', b'\rangle)$; therefore, App is also monotone.
  $\textsc{Pos}$ also satisfies the equations in definition 2.3.6. The proof is carried out in the same way of $\textsc{Set}$.

(3) $\textsc{Pos}_\perp$

$\textsc{Pos}_\perp$ is the same as the category $\textsc{Pos}$ except that each poset has a least element $\perp$. We construct its identity, composition, terminal object, products, and exponentials in the same way of $\textsc{Pos}$.

However, the least elements in products and exponentials should be defined particularly. If $\perp_A$ and $\perp_B$ are least elements in posets $\mathcal{A}$ and $\mathcal{B}$, the least element in product $\mathcal{A} \times \mathcal{B}$ is $\perp_{A\times B} = \langle \perp_A, \perp_B\rangle$. The least element $\perp_{A\to B}$ in exponential

$\mathcal{A} \to \mathcal{B}$ is the map converting any $a \in A$ to $\perp_B$, i.e. $\perp_{A\to B}(a) = \perp_B$ for all $a \in A$. Since the definition of morphisms in $\textsc{Pos}_\perp$ is still the same as the one in $\textsc{Pos}$, $\textsc{Pos}_\perp$ should be cartesian closed.

(4) $\textsc{Pos}_{\perp!}$

In category $\textsc{Pos}_{\perp!}$, the objects are posets with least elements and the morphisms are monotone maps that preserve the least element $\perp$, i.e. for every monotone map $f : A \to B$, $f(\perp_A) = \perp_B$.

Though it looks similar to $\textsc{Pos}_\perp$, $\textsc{Pos}_{\perp!}$ is not cartesian closed. If we still define the products and exponentials in the same way as $\textsc{Pos}$, we we find that it cannot satisfy all the three conditions of cartesian closed categories:

- Any singleton poset is still a terminal object in $\textsc{Pos}_{\perp!}$.
- The cartesian products can work as products in $\textsc{Pos}_{\perp!}$. For any $f_1 : A \to B$ and $f_2 : A \to C$, the map $\langle f_1, f_2\rangle$ has been proven to be monotone in (2). According to the definition of $\langle\cdot,\cdot\rangle$, $\langle f_1, f_2\rangle(\perp_A) = \langle f_1(\perp_A), f_2(\perp_A)\rangle = \langle \perp_B, \perp_C\rangle = \perp_{B\times C}$. It preserves $\perp$; therefore, the cartesian products are products in in $\textsc{Pos}_{\perp!}$.
- The exponentials defined in (2) are not exponentials in $\textsc{Pos}_{\perp!}$. Assume $\textsc{Pos}_{\perp!}$ has the same exponentials as $\textsc{Pos}$, then for any morphism $f : A \times B \to C$ there should be a unique $g = \mathrm{Curry}(f) : A \to (B \to C)$ which is isomorphic to f. Since $g(\perp_A) = \perp_{B\to C}$ and $\perp_{B\to C}(b) = \perp_C$ for any $b \in B$, we have $(g(\perp_A))(b) = \perp_C$. However, f may not map all $\langle \perp_A, b\rangle$ to $\perp_C$, which means that $f \ncong g$. So the exponentials in $\textsc{Pos}$ are not exponentials in $\textsc{Pos}_{\perp!}$.

One may still think that $\textsc{Pos}_{\perp!}$ is cartesian closed because we may be able to find other products or exponentials for it.

If we construct a "product" of A and B by

$$A \bigotimes B = \{\langle a, b\rangle | a \in A - \{\perp_A\}, b \in B - \{\perp_B\}\} \cup \{\langle \perp_A, \perp_B\rangle\},$$

then the exponentials from $\textsc{Pos}$ work in $\textsc{Pos}_{\perp!}$ now since $\langle \perp_A, b\rangle$ is not an element in "product" $A \bigotimes B$. However, this $\bigotimes$ is not a categorical product. Given any $f_1 : A \to B$ and $f_2 : A \to C$, the map $\langle f_1, f_2\rangle$ should map all $a \in A$ to elements in $B \bigotimes C$. But $\langle \perp_{A\to B}, f_2\rangle(a) = \langle \perp_{A\to B}(a), f_2(a)\rangle = \langle \perp_B, f_2(a)\rangle$ and $\langle \perp_B, f_2(a)\rangle \notin B \bigotimes C$ if $f_2(a) \neq \perp_C$.

The fact is that $\textsc{Pos}_{\perp!}$ does not have exponentials. Let *unit* be the terminal object in $\textsc{Pos}_{\perp!}$. We know that for any object A, $\mathit{unit} \times A \cong A$. Then their hom-sets should be isomorphic as well, i.e. $\mathrm{Hom}(A, A) \cong \mathrm{Hom}(\mathit{unit} \times A, A)$. Provided that $\textsc{Pos}_{\perp!}$ has exponentials, we have $\mathrm{Hom}(\mathit{unit} \times A, A) \cong \mathrm{Hom}(\mathit{unit}, A \to A)$ and so $\mathrm{Hom}(A, A) \cong \mathrm{Hom}(\mathit{unit}, A \to A)$. If A is not singleton (terminal object), then there can be many monotone maps from A to A, which means $\mathrm{Hom}(A, A)$ can have many elements. However, $\mathrm{Hom}(\mathit{unit}, A \to A)$ has only one morphism since there is only one map from the singleton poset to any other poset. $\mathrm{Hom}(A, A)$ is not isomorphic to $\mathrm{Hom}(\mathit{unit}, A \to A)$; therefore, $\textsc{Pos}_{\perp!}$ cannot have exponentials.

3 Correspondences

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

The proof trees in 2.1.1 have a similar shape to the type-derivation trees in 2.2.1. In fact, the rules in natural deduction system to construct relations $\Gamma \vdash \varphi$ work in the same way as the typing rules to build well-typed λ-terms $\Gamma \rhd x : \sigma$. Let us have a look at the simplest example first, the type-derivations in simply typed lambda calculus $\lambda^{\rightarrow}$ and the proofs in intuitionistic implicational logic.

$$\dfrac{\dfrac{\dfrac{\overline{x{:}\sigma \rhd x : \sigma}}{x{:}\sigma, y{:}\tau \rhd x : \sigma}\,(add)}{x{:}\sigma \rhd \lambda y{:}\tau.x : \tau \rightarrow \sigma}\,(\rightarrow\text{I})}{\emptyset \rhd \lambda x{:}\sigma.\lambda y{:}\tau.x : \sigma \rightarrow (\tau \rightarrow \sigma)}\,(\rightarrow\text{I})$$

The tree above is a type-derivation of λ-term $\lambda x{:}\sigma.\lambda y{:}\tau.x : \sigma \rightarrow (\tau \rightarrow \sigma)$. Let us erase the terms in the tree. If we consider every type-symbol as a propositional variable and the relation "$\rhd$" as "$\vdash$", then the tree in the right side is exactly a proof tree, which means we obtain a proof of the conclusion represented by the type of the term in the bottom of the tree. Here, the proven proposition is $\sigma \rightarrow (\tau \rightarrow \sigma)$.

$$\dfrac{\dfrac{\dfrac{\overline{\cancel{x{:}}\ \sigma \rhd \cancel{x :}\ \sigma}}{\cancel{x{:}}\ \sigma,\ \cancel{y{:}}\ \tau \rhd \cancel{x :}\ \sigma}\,(add)}{\cancel{x{:}}\ \sigma \rhd \cancel{\lambda y{:}\tau.x :}\ \tau \rightarrow \sigma}\,(\rightarrow\text{I})}{\emptyset \rhd \cancel{\lambda x{:}\sigma.\lambda y{:}\tau.x :}\ \sigma \rightarrow (\tau \rightarrow \sigma)}\,(\rightarrow\text{I}) \quad \Longrightarrow \quad \dfrac{\dfrac{\dfrac{\overline{\sigma \rhd \sigma}}{\sigma, \tau \rhd \sigma}\,(add)}{\sigma \rhd \tau \rightarrow \sigma}\,(\rightarrow\text{I})}{\emptyset \rhd \sigma \rightarrow (\tau \rightarrow \sigma)}\,(\rightarrow\text{I})$$

Proposition 3.1.1. Every type-derivation in $\lambda^{\rightarrow}$ leads to a proof in intuitionistic implicational logic.

Proof.

This proposition can be proven by induction on the height of type-derivation trees. In order to make the proof more readable, the types are denoted by using the propositional variable symbols $(\varphi, \psi ...)$.

The base case is the one when the height of the tree is 1, i.e. a type-derivation without hypotheses which is the typing rule $(axiom)$. By erasing the term and replacing the relation symbol "$\rhd$" by "$\vdash$", we obtain the axiom in the natural deduction system.

$$\dfrac{}{x{:}\varphi \rhd x : \varphi}\,(axiom) \quad \Longrightarrow \quad \dfrac{}{\varphi \vdash \varphi}\,(axiom)$$

Assume that from very type-derivation tree of height at most n, we get a corresponding proof by erasing the terms in the tree. Then, what we need to prove is that the tree of height $n + 1$ also leads to a corresponding proof. According to the typing rules of $\lambda^{\rightarrow}$, there are two inductive cases, one with lambda abstraction and another with application:

(1) Lambda abstraction

The induction hypothesis in this case is that the following type-derivation has height n and leads to a proof on its right side,

$$\dfrac{\vdots}{\Gamma, x{:}\varphi \rhd M : \psi} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma', \varphi \vdash \psi}$$

Since the proof tree is obtained by erasing the terms in the derivation tree, the context Γ' should contain all the types in Γ and nothing else. By using introduction rule for function type on $\Gamma, x{:}\varphi \rhd M : \psi$, we have a type-derivation of term $\lambda x{:}\varphi.M$

$$\dfrac{\dfrac{\vdots}{\Gamma, x{:}\varphi \rhd M : \psi}}{\Gamma \rhd \lambda x{:}\varphi.M : \varphi \rightarrow \psi}\ (\rightarrow\mathrm{I})$$

By using introduction rule for implication on $\Gamma', \varphi \vdash \psi$, we have the following proof

$$\dfrac{\dfrac{\vdots}{\Gamma', \varphi \vdash \psi}}{\Gamma' \vdash \varphi \rightarrow \psi}\ (\rightarrow\mathrm{I})$$

The conclusion of the proof, $\varphi \rightarrow \psi$, is the type of term $\lambda x{:}\varphi.M$. Besides, Γ' contains all the types in Γ. So we can say that the proof of $\Gamma' \vdash \varphi \rightarrow \psi$ is obtained by deleting the terms in the type derivation of $\Gamma \rhd \lambda x{:}\varphi.M : \varphi \rightarrow \psi$. In other words, the derivation of $\Gamma \rhd \lambda x{:}\varphi.M : \varphi \rightarrow \psi$ leads to a proof of $\Gamma' \vdash \varphi \rightarrow \psi$.

(2) Application

Assume that each of the following two derivation-trees, one of which has height n and another has height at most n, leads to a corresponding proof

$$\dfrac{\vdots}{\Gamma \rhd M : \varphi \rightarrow \psi} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \vdash \varphi \rightarrow \psi}$$

$$\dfrac{\vdots}{\Gamma \rhd N : \varphi} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \vdash \varphi}$$

Similarly to (1), Γ' contains all the types in Γ. Then, according to function application, we obtain a derivation of MN

$$\dfrac{\dfrac{\vdots}{\Gamma \rhd M : \varphi \rightarrow \psi} \qquad \dfrac{\vdots}{\Gamma \rhd N : \varphi}}{\Gamma \rhd MN : \psi}\ (\rightarrow\mathrm{E})$$

We also get a proof of ψ by applying *modus ponens* on $\varphi \rightarrow \psi$ and φ

$$\dfrac{\dfrac{\vdots}{\Gamma' \vdash \varphi \rightarrow \psi} \qquad \dfrac{\vdots}{\Gamma' \vdash \varphi}}{\Gamma' \vdash \psi}\ (\rightarrow\mathrm{E})$$

Then we can see that if we erase the terms in Γ and term MN, we also have $\Gamma' \vdash \psi$. Hence, the derivation of $\Gamma \rhd MN : \psi$ leads to a proof of $\Gamma' \vdash \psi$.

Therefore, every type-derivation in $\lambda^{\rightarrow}$ leads to a proof in intuitionistic implicational logic.

□

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

Proposition 3.1.1 in last section tells us that, from every type-derivation, one can obtain a proof. In this section, we will have a look at the inverse direction and extend this connection to intuitionistic propositional logic and full simply typed lambda calculus.

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

Given the above proof of proposition $\varphi \to (\tau \to \varphi)$, we construct a type-derivation tree in the following steps:

(1) Treat each proposition in the context as a type and assign a term variable to it. Atomic propositions are considered as atomic types while implications, conjunctions, disjunctions and contradiction correspond to function types, product types, sum types and initial type, respectively. The same term variables are assigned to the same propositions in all the contexts through out the whole tree. Each term variable is assigned to one proposition only in order to make the type context consistent.

$$\dfrac{\dfrac{\dfrac{\overline{x : \varphi \vdash \varphi}}{x : \varphi, y : \tau \vdash \varphi}\,(add)}{x : \varphi \vdash \tau \to \varphi}\,(\to\mathrm{I})}{\vdash \varphi \to (\tau \to \varphi)}\,(\to\mathrm{I})$$

(2) Use the type assignments in the context Γ to construct a proper term M of type φ from the top of the tree. Then a typing judgement $\Gamma \vdash M : \varphi$ is obtained. If the proposition (or type) on the right side of $\vdash$ already exists in the context, then the term of that type is the term variable which has been assign to that type in the context.

$$\dfrac{\dfrac{\dfrac{\overline{x : \varphi \vdash x : \varphi}}{x : \varphi, y : \tau \vdash x : \varphi}\,(add)}{x : \varphi \vdash ? : \tau \to \varphi}\,(\to\mathrm{I})}{\vdash ? : \varphi \to (\tau \to \varphi)}\,(\to\mathrm{I})$$

In the third typing judgement (counted from the top of the tree), the proposition $\tau \to \varphi$ does not appear in the context. So the term of type $\tau \to \varphi$ is not a term variable. Besides, there is only one sub-tree above the judgement. So the term must be a lambda abstraction where the bound variable in the abstractor has type τ and the scope has type φ. Then we need to go up to look for the variable which is assigned to τ in the context and the term which has type φ. From the second typing judgement, we know y is assigned to τ and term x has type φ. Therefore, we construct the term in the third typing judgement as $\lambda y : \tau . x : \tau \to \varphi$. The last typing judgement is obtained in the same way.

$$\cfrac{\cfrac{\cfrac{\overline{x:\varphi \vdash x:\varphi}}{x:\varphi, y:\tau \vdash x:\varphi}\,(add)}{x:\varphi \vdash \lambda y:\tau.x:\tau \to \varphi}\,(\to\mathrm{I})}{\vdash \lambda x:\varphi.\lambda y:\tau.x:\varphi \to (\tau \to \varphi)}\,(\to\mathrm{I})$$

(3) Replace the relation "$\vdash$" by "$\triangleright$".

$$\cfrac{\cfrac{\cfrac{\overline{x:\varphi \triangleright x:\varphi}}{x:\varphi, y:\tau \triangleright x:\varphi}\,(add)}{x:\varphi \triangleright \lambda y:\tau.x:\tau \to \varphi}\,(\to\mathrm{I})}{\triangleright \lambda x:\varphi.\lambda y:\tau.x:\varphi \to (\tau \to \varphi)}\,(\to\mathrm{I})$$

Proposition 3.2.1. Every proof in intuitionistic propositional logic leads to a type-derivation in full simply typed lambda calculus.

Proof.

Its proof is also carried out by induction on the height of proof tree. In this proof, the type symbols $(\sigma, \tau, \rho \cdots)$ are adopted as proposition symbols in order to make the proof easier to read.

The base case is the proof tree of height 1 which is the axiom in the natural deduction system. In (*axiom*), the concluded proposition on the right side of $\vdash$ also appears in the context. Both of them are assigned to a same term variable. By replacing $\vdash$ by $\triangleright$, we obtain the typing rule (*axiom*).

$$\overline{\sigma \vdash \sigma}\ (axiom) \implies \overline{x:\sigma \triangleright x:\sigma}\ (axiom)$$

In the inductive step, let us assume that all the proofs of height at most n lead to a corresponding type-derivation. Then what we want to prove is that the proof of height $n+1$ can lead to a type-derivation as well. Except the axiom, the natural deduction system of intuitionisitic propositional logic has several other inference rules for all the propositional connectives, $\to$, $\wedge$, $\vee$, and $\perp$. Each them is an inductive case.

(1) $(\to\mathrm{I})$

The induction hypothesis here is that the proof of height n with conclusion $\Gamma, \sigma \vdash \tau$ lead to a type derivation of $\Gamma', x:\sigma \triangleright M:\tau$ where Γ' is a consistent type context whose types are propositions in Γ.

$$\cfrac{\vdots}{\Gamma, \sigma \vdash \tau} \implies \cfrac{\vdots}{\Gamma', x:\sigma \triangleright M:\tau}$$

By following the introduction rule of implication $(\to\mathrm{I})$, we get the following proof whose height is $n+1$.

$$\cfrac{\cfrac{\vdots}{\Gamma, \sigma \vdash \tau}}{\Gamma \vdash \sigma \to \tau}\,(\to\mathrm{I})$$

According to the typing rule for introducing function type $(\to\mathrm{I})$, we get another type derivation as follows.

$$\cfrac{\cfrac{\vdots}{\Gamma', x:\sigma \triangleright M:\tau}}{\Gamma' \triangleright \lambda x:\sigma.M:\sigma \to \tau}\,(\to\mathrm{I})$$

Since all the types in type context Γ' are the propositions in Γ and the term $\lambda x : \sigma.M$ has type $\sigma \to \tau$, the proof of $\Gamma \vdash \sigma \to \tau$ leads to the derivation of $\Gamma' \rhd \lambda x : \sigma.M : \sigma \to \tau$.

(2) ($\to$E)

In this case, we assume the two proofs below, one of which has height n and another one has height at most n, lead to the type derivations next to them.

$$\dfrac{\vdots}{\Gamma \vdash \sigma \to \tau} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \rhd M : \sigma \to \tau}$$

$$\dfrac{\vdots}{\Gamma \vdash \sigma} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \rhd N : \sigma}$$

We use modus ponens ($\to$E) and then have a proof of $\Gamma \vdash \tau$.

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \sigma \to \tau} \qquad \dfrac{\vdots}{\Gamma \vdash \sigma}}{\Gamma \vdash \tau}\ (\to\mathrm{E})$$

We use function application ($\to$E) and then have a type derivation with conclusion $\Gamma' \rhd MN : \tau$.

$$\dfrac{\dfrac{\vdots}{\Gamma' \rhd N : \sigma} \qquad \dfrac{\vdots}{\Gamma' \rhd N : \sigma}}{\Gamma' \rhd MN : \tau}\ (\to\mathrm{E})$$

We can see that Γ' encodes all the propositions in Γ and MN has type τ; therefore, the proof of $\Gamma \vdash \tau$ leads to the type derivation of $\Gamma' \rhd MN : \tau$.

(3) ($\wedge$I)

Assume the two following proofs, one of which has height n and another one has height at most n, lead to the type derivations on their right.

$$\dfrac{\vdots}{\Gamma \vdash \sigma} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \rhd M : \sigma}$$

$$\dfrac{\vdots}{\Gamma \vdash \tau} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \rhd N : \tau}$$

By introduction rule of conjunction ($\wedge$I), we get a proof of $\Gamma \vdash \sigma \wedge \tau$.

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \sigma} \qquad \dfrac{\vdots}{\Gamma \vdash \tau}}{\Gamma \vdash \sigma \wedge \tau}\ (\wedge\mathrm{I})$$

By introduction rule of product type ($\times$I), we get a type derivation of $\Gamma' \rhd \langle M, N \rangle : \sigma \times \tau$.

$$\dfrac{\dfrac{\vdots}{\Gamma' \rhd M : \sigma} \qquad \dfrac{\vdots}{\Gamma' \rhd N : \tau}}{\Gamma' \rhd \langle M, N \rangle : \sigma \times \tau}\ (\times\mathrm{I})$$

Then, the proof of $\Gamma \vdash \sigma \wedge \tau$ leads to the type derivation of $\Gamma' \rhd \langle M, N \rangle : \sigma \times \tau$.

(4) ($\wedge$E)

Assume the following proof of height n leads to the type derivation next to it.

$$\dfrac{\vdots}{\Gamma \vdash \sigma \wedge \tau} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \rhd M : \sigma \times \tau}$$

By the elimination rule of conjunction ($\wedge$E), we get a proof of $\Gamma \vdash \sigma$ and another one of $\Gamma \vdash \tau$.

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \sigma \wedge \tau}}{\Gamma \vdash \sigma}\,(\wedge \mathrm{E}_1) \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash \sigma \wedge \tau}}{\Gamma \vdash \tau}\,(\wedge \mathrm{E}_2)$$

By the elimination rule of product type ($\times$E), we get a type derivation of $\Gamma' \triangleright \mathrm{Proj}_1 M : \sigma$ and another one of $\Gamma' \triangleright \mathrm{Proj}_2 M : \tau$.

$$\dfrac{\dfrac{\vdots}{\Gamma' \triangleright M : \sigma \times \tau}}{\Gamma' \triangleright \mathrm{Proj}_1 M : \sigma}\,(\times \mathrm{E}_1) \qquad \dfrac{\dfrac{\vdots}{\Gamma' \triangleright M : \sigma \times \tau}}{\Gamma' \triangleright \mathrm{Proj}_2 M : \tau}\,(\times \mathrm{E}_2)$$

Hence, both proofs of $\Gamma \vdash \sigma$ and $\Gamma \vdash \tau$ lead to the type derivations of $\Gamma' \triangleright \mathrm{Proj}_1 M : \sigma$ and $\Gamma' \triangleright \mathrm{Proj}_2 M : \tau$ respectively.

(5) ($\vee$I)

Assume the following proof of height n leads to the type derivation on its right.

$$\dfrac{\vdots}{\Gamma \vdash \sigma} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \triangleright M : \sigma}$$

By the introduction rule of disjunction ($\vee$I), we have a proof of $\Gamma \vdash \sigma \vee \tau$ and another one of $\Gamma \vdash \tau \vee \sigma$.

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \sigma}}{\Gamma \vdash \sigma \vee \tau}\,(\vee \mathrm{I}_1) \qquad \dfrac{\dfrac{\vdots}{\Gamma \vdash \sigma}}{\Gamma \vdash \tau \vee \sigma}\,(\vee \mathrm{I}_2)$$

By the introduction rule of sum type (+I), we have a type derivation of $\Gamma' \triangleright \mathrm{Inleft} M : \sigma + \tau$ and another one of $\Gamma' \triangleright \mathrm{Inright} M : \tau + \sigma$.

$$\dfrac{\dfrac{\vdots}{\Gamma' \triangleright M : \sigma}}{\Gamma' \triangleright \mathrm{Inleft} M : \sigma + \tau}\,(+\mathrm{I}_1) \qquad \dfrac{\dfrac{\vdots}{\Gamma' \triangleright M : \sigma}}{\Gamma' \triangleright \mathrm{Inright} M : \tau + \sigma}\,(+\mathrm{I}_2)$$

Hence, both proofs of $\Gamma \vdash \sigma \vee \tau$ and $\Gamma \vdash \tau \vee \sigma$ lead to the type derivations of $\Gamma' \triangleright \mathrm{Inleft} M : \sigma + \tau$ and $\Gamma' \triangleright \mathrm{Inright} M : \tau + \sigma$ respectively.

(6) ($\vee$E)

Assume the three following proofs, one of which has height n and the other two have height at most n, lead to the type derivations next to them.

$$\dfrac{\vdots}{\Gamma \vdash \sigma \vee \tau} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \triangleright M : \sigma + \tau}$$

$$\dfrac{\vdots}{\Gamma \vdash \sigma \to \rho} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \triangleright N : \sigma \to \rho}$$

$$\dfrac{\vdots}{\Gamma \vdash \tau \to \rho} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \triangleright P : \tau \to \rho}$$

By the elimination rule of disjunction ($\vee$E), we get a proof of $\Gamma \vdash \rho$.

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \sigma \vee \tau} \quad \dfrac{\vdots}{\Gamma \vdash \sigma \to \rho} \quad \dfrac{\vdots}{\Gamma \vdash \tau \to \rho}}{\Gamma \vdash \rho}\,(\vee \mathrm{E})$$

By the elimination rule of sum type (+E), we get a type derivation of $\Gamma' \vdash$ Case MNP : ρ.

$$\dfrac{\dfrac{\vdots}{\Gamma' \rhd M : \sigma + \tau} \quad \dfrac{\vdots}{\Gamma' \rhd N : \sigma \to \rho} \quad \dfrac{\vdots}{\Gamma' \rhd P : \tau \to \rho}}{\Gamma' \rhd \text{Case } MNP : \rho} \, (+\text{E})$$

Hence, the proof of $\Gamma \vdash \rho$ leads to the type derivation of $\Gamma' \rhd$ Case $MNP : \rho$.

(7) ($\bot$E)

Assume that we have a proof of $\bot$ under the context Γ and from this proof we obtain a type derivation of a term M of initial type *null*.

$$\dfrac{\vdots}{\Gamma \vdash \bot} \quad \Longrightarrow \quad \dfrac{\vdots}{\Gamma' \rhd M : \mathit{null}}$$

This may look very strange since we mentioned in section 2.2.3 that there is no instance of initial type but here we say term M has initial type. The reason for this is that there is no closed term of initial type. But under some particular assumptions (type contexts), some terms with free variables can have initial type. For instance, if we apply term $x : \tau$ to $f : \tau \to \mathit{null}$ where both of them appear in the type context, then the application fx should have type *null* in the same context, in accordance with function application.

By the inference rule ($\bot$E), we have a proof for any σ in the same context Γ.

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \bot}}{\Gamma \vdash \sigma} \, (\bot\text{E})$$

Since for every type σ there is always a unique term $\text{Zero}^{\sigma} : \mathit{null} \to \sigma$, we can apply term M to Zero^{σ} and then the result has type σ.

$$\dfrac{\dfrac{}{\Gamma' \rhd \text{Zero}^{\sigma} : \mathit{null} \to \sigma} \quad \dfrac{\vdots}{\Gamma' \rhd M : \mathit{null}}}{\Gamma' \rhd \text{Zero}^{\sigma} M : \sigma} \, (\to\text{E})$$

Hence, the proof of $\Gamma \vdash \sigma$ leads to the type derivation of $\Gamma' \rhd \text{Zero}^{\sigma} M : \sigma$.

Therefore, every proof in intuitionistic propositional logic can be encoded by a lambda term in full simply typed lambda calculus.

□

According to proposition 2.2.3, each well-typed lambda term uniquely determines a type-derivation. As a result, *every proof can be encoded in a well-typed lambda term.* Specifically, the proofs of theorems (whose contexts are empty) can be encoded in closed terms while the ones of propositions having assumptions in their contexts can be encoded in terms with the free variables encoding the assumptions in the typing contexts. For instance, the combinator $\mathbf{K} \equiv \lambda x : \sigma.\lambda y : \tau.x : \sigma \to (\tau \to \sigma)$ encodes the proof of theorem $\vdash \sigma \to (\tau \to \sigma)$. Once the proof of the theorem is required, we just build the type-derivation tree of the term and then erase all the terms but keep the types in the tree. Then the proof is decoded from its corresponding term.

This and the last sections demonstrate a closed connection between proofs in intuitionistic propositional logic and terms in simply typed lambda calculus. To put it more specific, let us have a look at this correspondence from the following points of view.

For one thing, *propositions correspond to types.* If we take the set of propositional variables equal to the set of type variables, then the set of propositional formulas is identical to the set of simple types. The reason for this is that each propositional connective corresponds to a simple type constructor. Implication $\rightarrow$ corresponds to function constructor $\rightarrow$, conjunction $\wedge$ to product constructor $\times$, and disjunction $\vee$ to sum constructor $+$.

For another, *proofs correspond to terms.* Since propositional connectives correspond to simple type constructors, the inference rule for these connectives should have the same behaviour with the typing rules for the simple type constructors. From 3.1 and 3.2, it is obvious that the inference rules in natural deduction system build proof trees in the same way as the typing rules build type derivations.

3.3 Every well-typed lambda term can be interpreted as a morphism in a cartesian closed category

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 $\lambda^{unit,\times,\rightarrow}$ 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 $\lambda^{\rightarrow}$ with only function type constructor is as expressive as $\lambda^{unit,\times,\rightarrow}$. Therefore, both the type expressions and well-typed lambda terms in $\lambda^{\rightarrow}$ can be interpreted in any CCC, and this interpretation should be sound and complete.

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

(1) The interpretation $\mathcal{C}[\![\sigma]\!]$ of type expression σ is defined as follows:

- $\mathcal{C}[\![b]\!] \quad = \quad \hat{b}$, given as an object constant in $\mathcal{C}$;
- $\mathcal{C}[\![\sigma \rightarrow \tau]\!] \quad = \quad \mathcal{C}[\![\sigma]\!] \rightarrow \mathcal{C}[\![\sigma]\!]$.

(2) The interpretation $\mathcal{C}[\![\Gamma]\!]$ of type context Γ is defined by induction on the length of the context:

- $\mathcal{C}[\![\emptyset]\!] \quad = \quad unit$;
- $\mathcal{C}[\![\Gamma, x : \sigma]\!] \quad = \quad \mathcal{C}[\![\Gamma]\!] \times \mathcal{C}[\![\sigma]\!]$.

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

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

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

In this definition, $f : \{1, \ldots, m\} \rightarrow \{1, \ldots, n\}$ is called an *m,n-function.* If $\Gamma \equiv x_1 : \sigma_1, \ldots, x_n : \sigma_n$ is an ordered type context of length n, then the ordered type context Γ_f of length m is defined by $\Gamma_f = x_{f(1)} : \sigma_{f(1)}, \ldots, x_{f(m)} : \sigma_{f(m)}$. If $\Gamma \triangleright M : \sigma$ is a well-typed term and Γ_f contains all the free variables of M, then the interpretation of $\Gamma_f \triangleright M : \sigma$ can be related to the one of $\Gamma \triangleright M : \sigma$ by using a combination $\chi_f^{[\![\Gamma]\!]} : [\![\Gamma]\!] \rightarrow [\![\Gamma_f]\!]$ 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, $1 \leq i \leq n$, then we write $\langle h_1, \ldots, h_n \rangle$ for $\langle \langle \ldots \langle h_1, h_2 \rangle, \ldots \rangle, h_n \rangle : A \to ((\ldots(A_1 \times A_2) \times \ldots) \times A_n)$. Then we define the projections $\mathrm{Proj}_i^{A_1 \times A_2 \times \ldots \times A_n} : A_1 \times A_2 \times \ldots \times A_n \to A_i$ by a composition of projection morphisms Proj_1 and Proj_2. The following gives a inductive definition of projection $\mathrm{Proj}_i^{A_1 \times A_2 \times \ldots \times A_n}$:

- $\mathrm{Proj}_1^A \quad = \quad \mathrm{id}_A$
- $\mathrm{Proj}_n^{A_1 \times \ldots \times A_n} \quad = \quad \mathrm{Proj}_2^{(A_1 \times \ldots \times A_{n-1}), A_n} \qquad n > 1$
- $\mathrm{Proj}_i^{A_1 \times \ldots \times A_n} \quad = \quad \mathrm{Proj}_i^{A_1 \times \ldots \times A_{n-1}} \circ \mathrm{Proj}_1^{(A_1 \times \ldots \times A_{n-1}), A_n} \qquad 1 \leq i < n$

The tupling and projections defined above behave similarly to the pairs and Proj_1 and Proj_2. The following equations hold in any category with products:

- $\mathrm{Proj}_i \circ \langle f_1, \ldots, f_n \rangle \quad = \quad f_i$
- $\langle f_1, \ldots, f_n \rangle \circ h \quad = \quad \langle f_1 \circ h, \ldots, f_n \circ h \rangle$
- $\langle \mathrm{Proj}_1^{A_1 \times \ldots \times A_n}, \ldots, \mathrm{Proj}_n^{A_1 \times \ldots \times A_n} \rangle \quad = \quad \mathrm{id}_{A_1 \times \ldots \times A_n}$

Given an m, n-function f, we define $\chi_f^{unit \times A_1 \times \ldots \times A_n}$ by

$$\chi_f^{unit \times A_1 \times \ldots \times A_n} = \langle \mathrm{One}^{unit \times A_1 \times \ldots \times A_n}, \mathrm{Proj}_{f(1)+1}^{unit \times A_1 \times \ldots \times A_n}, \ldots, \mathrm{Proj}_{f(m)+1}^{unit \times A_1 \times \ldots \times A_n} \rangle$$
$$: unit \times A_1 \times \ldots \times A_n \to unit \times A_{f(1)} \times \ldots \times A_{f(m)}.$$

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 during the proof of soundness and completeness of the interpretation in definition 3.3.1.

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

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

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

$$\begin{aligned} & [\![\Gamma, x : \sigma \triangleright x : \sigma]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!] \rangle \\ = \ & \mathrm{Proj}_2^{[\![\Gamma]\!], \sigma} \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!] \rangle \qquad (\text{by } \mathrm{Proj}_i \circ \langle f_1, f_2 \rangle = f_i) \\ = \ & [\![\Gamma \triangleright N : \sigma]\!] \\ = \ & [\![\Gamma \triangleright [N/x]x : \sigma]\!] \end{aligned}$$

Inductive steps

Application $M : \tau \equiv M_1 M_2 : \tau_2$

Assume the following two equations hold

- $[\![\Gamma \triangleright [N/x]M_1 : \tau_1 \to \tau_2]\!] \quad = \quad [\![\Gamma, x : \sigma \triangleright M_1 : \tau_1 \to \tau_2]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!]\rangle$
- $[\![\Gamma \triangleright [N/x]M_2 : \tau_1]\!] \quad = \quad [\![\Gamma, x : \sigma \triangleright M_2 : \tau_1]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!]\rangle$

Then what we want to show is that $[\![\Gamma \triangleright [N/x](M_1 M_2) : \tau_2]\!] = [\![\Gamma, x : \sigma \triangleright M_1 M_2 : \tau_2]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!]\rangle$ holds.

$$
\begin{aligned}
& [\![\Gamma \triangleright [N/x](M_1 M_2) : \tau_2]\!] \\
= \;& \mathrm{App}^{\tau_1,\tau_2} \circ \langle [\![\Gamma \triangleright [N/x]M_1 : \tau_1 \to \tau_2]\!], [\![\Gamma \triangleright [N/x]M_2 : \tau_1]\!]\rangle \\
= \;& \mathrm{App}^{\tau_1,\tau_2} \circ \langle [\![\Gamma, x : \sigma \triangleright M_1 : \tau_1 \to \tau_2]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!]\rangle, \\
& \qquad\qquad [\![\Gamma, x : \sigma \triangleright M_2 : \tau_1]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!]\rangle\rangle \\
& (\text{by } \langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle \text{ in proposition 2.3.8 }) \\
= \;& \mathrm{App}^{\tau_1,\tau_2} \circ (\langle [\![\Gamma, x : \sigma \triangleright M_1 : \tau_1 \to \tau_2]\!], [\![\Gamma, x : \sigma \triangleright M_2 : \tau_1]\!]\rangle \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!]\rangle) \\
= \;& (\mathrm{App}^{\tau_1,\tau_2} \circ \langle [\![\Gamma, x : \sigma \triangleright M_1 : \tau_1 \to \tau_2]\!], [\![\Gamma, x : \sigma \triangleright M_2 : \tau_1]\!]\rangle) \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!]\rangle \\
= \;& [\![\Gamma, x : \sigma \triangleright M_1 M_2 : \tau_2]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!]\rangle
\end{aligned}
$$

Abstraction $M : \tau \equiv \lambda y : \rho.M' : \rho \to \tau$

Assume $[\![\Gamma, y : \rho \triangleright [N/x]M' : \tau]\!] = [\![\Gamma, y : \rho, x : \sigma \triangleright M' : \tau]\!] \circ \langle \mathrm{id}_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y : \rho \triangleright N : \sigma]\!]\rangle$, then we have

$$
\begin{aligned}
& [\![\Gamma, y : \rho \triangleright [N/x]M' : \tau]\!] \\
= \;& [\![\Gamma, y : \rho, x : \sigma \triangleright M' : \tau]\!] \circ \langle \mathrm{id}_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y : \rho \triangleright N : \sigma]\!]\rangle \\
& ((\Gamma, y : \rho, x : \sigma)_f = \Gamma, x : \sigma, y : \rho \text{ which swaps the positions of } y \text{ and } x) \\
= \;& ([\![\Gamma, x : \sigma, y : \rho \triangleright M' : \tau]\!] \circ \chi_f^{[\![\Gamma, y:\rho, x:\sigma]\!]}) \circ \langle \mathrm{id}_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y : \rho \triangleright N : \sigma]\!]\rangle \\
= \;& [\![\Gamma, x : \sigma, y : \rho \triangleright M' : \tau]\!] \circ (\chi_f^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ \langle \mathrm{id}_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y : \rho \triangleright N : \sigma]\!]\rangle) \\
= \;& [\![\Gamma, x : \sigma, y : \rho \triangleright M' : \tau]\!] \circ (\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!]\rangle \times \mathrm{id}_{[\![\rho]\!]})
\end{aligned}
$$

The step simplifying $\chi_f^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ \langle \mathrm{id}_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y : \rho \triangleright N : \sigma]\!]\rangle$ is a routine calculation involving pairing and projection morphisms. According to the definitions of tupling and χ,

$$
\begin{aligned}
\chi_f^{[\![\Gamma, y:\rho, x:\sigma]\!]} = \langle \chi_g^{[\![\Gamma, y:\rho, x:\sigma]\!]}, \mathrm{Proj}_{k+2}^{[\![\Gamma]\!] \times [\![\rho]\!] \times [\![\sigma]\!]}, \mathrm{Proj}_{k+1}^{[\![\Gamma]\!] \times [\![\rho]\!] \times [\![\sigma]\!]}\rangle \\
: [\![\Gamma]\!] \times [\![\rho]\!] \times [\![\sigma]\!] \to [\![\Gamma]\!] \times [\![\sigma]\!] \times [\![\rho]\!]
\end{aligned}
$$

where k is the length of $[\![\Gamma]\!]$ and g is a n, m-function such that $(\Gamma, y : \rho, x : \sigma)_g = \Gamma$. Then the simplification is done as follows:

$$
\begin{aligned}
& \chi_f^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ \langle \mathrm{id}_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y : \rho \rhd N : \sigma]\!]\rangle \\
& \text{(denote } h \text{ as } \langle \mathrm{id}_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y : \rho \rhd N : \sigma]\!]\rangle) \\
= \ & \langle \chi_g^{[\![\Gamma, y:\rho, x:\sigma]\!]}, \mathrm{Proj}_{k+2}^{[\![\Gamma]\!]\times[\![\rho]\!]\times[\![\sigma]\!]}, \mathrm{Proj}_{k+1}^{[\![\Gamma]\!]\times[\![\rho]\!]\times[\![\sigma]\!]}\rangle \circ h \\
& \text{(by } \langle f_1, \ldots, f_n\rangle \circ h = \langle f_1 \circ h, \ldots, f_n \circ h\rangle) \\
= \ & \langle \chi_g^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ h, \mathrm{Proj}_{k+2}^{[\![\Gamma]\!]\times[\![\rho]\!]\times[\![\sigma]\!]} \circ h, \mathrm{Proj}_{k+1}^{[\![\Gamma]\!]\times[\![\rho]\!]\times[\![\sigma]\!]} \circ h\rangle \\
& \text{(by } \mathrm{Proj}_n^{A_1\times\ldots\times A_n} = \mathrm{Proj}_2^{(A_1\times\ldots\times A_{n-1}),A_n}) \\
= \ & \langle \chi_g^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ h, \mathrm{Proj}_2^{([\![\Gamma]\!]\times[\![\rho]\!]),[\![\sigma]\!]} \circ \langle \mathrm{id}_{[\![\Gamma, y:\rho]\!]}, [\![\Gamma, y : \rho \rhd N : \sigma]\!]\rangle, \mathrm{Proj}_{k+1}^{[\![\Gamma]\!]\times[\![\rho]\!]\times[\![\sigma]\!]} \circ h\rangle \\
& \text{(by } \mathrm{Proj}_2 \circ \langle f_1, f_2\rangle = f_2) \\
= \ & \langle \chi_g^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ h, [\![\Gamma, y : \rho \rhd N : \sigma]\!], \mathrm{Proj}_{k+1}^{[\![\Gamma]\!]\times[\![\rho]\!]\times[\![\sigma]\!]} \circ h\rangle \\
& \text{(by the definition of } \mathrm{Proj}_i) \\
= \ & \langle \chi_g^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ h, [\![\Gamma, y : \rho \rhd N : \sigma]\!], \mathrm{Proj}_2^{[\![\Gamma]\!],[\![\rho]\!]} \circ \mathrm{Proj}_1^{[\![\Gamma]\!]\times[\![\rho]\!],[\![\sigma]\!]} \circ h\rangle \\
= \ & \langle \chi_g^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ h, [\![\Gamma, y : \rho \rhd N : \sigma]\!], \mathrm{Proj}_2^{[\![\Gamma]\!],[\![\rho]\!]} \circ \mathrm{id}_{[\![\Gamma, y:\rho]\!]}\rangle \\
= \ & \langle \chi_g^{[\![\Gamma, y:\rho, x:\sigma]\!]} \circ h, [\![\Gamma, y : \rho \rhd N : \sigma]\!], \mathrm{Proj}_2^{[\![\Gamma]\!],[\![\rho]\!]}\rangle \\
& \text{(by the definitions of } \chi \text{ and } \mathrm{Proj}_i) \\
= \ & \langle \mathrm{id}_{[\![\Gamma]\!]} \circ \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\rho]\!]}, [\![\Gamma, y : \rho \rhd N : \sigma]\!], \mathrm{Proj}_2^{[\![\Gamma]\!],[\![\rho]\!]}\rangle \\
& \text{(if } (\Gamma, y : \rho)_{f'} = \Gamma, \text{ then } \chi_{f'} = \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\rho]\!]}) \\
= \ & \langle \mathrm{id}_{[\![\Gamma]\!]} \circ \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\rho]\!]}, [\![\Gamma \rhd N : \sigma]\!] \circ \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\rho]\!]}, \mathrm{Proj}_2^{[\![\Gamma]\!],[\![\rho]\!]}\rangle \\
= \ & \langle\langle \mathrm{id}_{[\![\Gamma]\!]} \circ \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\rho]\!]}, [\![\Gamma \rhd N : \sigma]\!] \circ \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\rho]\!]}\rangle, \mathrm{Proj}_2^{[\![\Gamma]\!],[\![\rho]\!]}\rangle \\
& \text{(by } \langle f, g\rangle \circ h = \langle f \circ h, g \circ h\rangle) \\
= \ & \langle\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \rhd N : \sigma]\!]\rangle \circ \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\rho]\!]}, \mathrm{Proj}_2^{[\![\Gamma]\!],[\![\rho]\!]}\rangle \\
= \ & \langle\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \rhd N : \sigma]\!]\rangle \circ \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\rho]\!]}, \mathrm{id}_{[\![\rho]\!]} \circ \mathrm{Proj}_2^{[\![\Gamma]\!],[\![\rho]\!]}\rangle \\
& \text{(by the definition of products of morphisms)} \\
= \ & \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \rhd N : \sigma]\!]\rangle \times \mathrm{id}_{[\![\rho]\!]}
\end{aligned}
$$

Now we need to show that $[\![\Gamma \rhd [N/x](\lambda y : \rho.M') : \rho \to \tau]\!] = [\![\Gamma, x : \sigma \rhd \lambda y : \rho.M' : \rho \to \tau]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \rhd N : \sigma]\!]\rangle$ holds.

$$
\begin{aligned}
& [\![\Gamma \rhd [N/x](\lambda y : \rho.M') : \rho \to \tau]\!] \\
= \ & [\![\Gamma \rhd \lambda y : \rho.[N/x]M' : \rho \to \tau]\!] \\
= \ & \mathrm{Curry}([\![\Gamma, y : \rho \rhd [N/x]M' : \tau]\!]) \\
= \ & \mathrm{Curry}([\![\Gamma, x : \sigma, y : \rho \rhd M' : \tau]\!] \circ (\langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \rhd N : \sigma]\!]\rangle \times \mathrm{id}_{[\![\rho]\!]})) \\
& \text{(by } \mathrm{Curry}(f) \circ h = \mathrm{Curry}(f \circ (h \times \mathrm{id})) \text{ in proposition 2.3.8)} \\
= \ & \mathrm{Curry}([\![\Gamma, x : \sigma, y : \rho \rhd M' : \tau]\!]) \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \rhd N : \sigma]\!]\rangle \\
= \ & [\![\Gamma, x : \sigma \rhd \lambda y : \rho.M' : \rho \to \tau]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \rhd N : \sigma]\!]\rangle
\end{aligned}
$$

Therefore, for all well-typed $\Gamma, x : \sigma \triangleright M : \tau$ and $\Gamma \triangleright N : \sigma$, it holds that $[\![\Gamma \triangleright [N/x]M : \tau]\!] = [\![\Gamma, x : \sigma \triangleright M : \tau]\!] \circ \langle \mathrm{id}_{[\![\Gamma]\!]}, [\![\Gamma \triangleright N : \sigma]\!] \rangle$.

$\square$

Theorem 3.3.3 (Soundness). *Given any well-typed* $\Gamma \triangleright M : \sigma$ *and* $\Gamma \triangleright N : \sigma$ *with* $M =_{\alpha\beta\eta} N$*, then they have the same interpretation, i.e.* $[\![\Gamma \triangleright M : \sigma]\!] = [\![\Gamma \triangleright N : \sigma]\!]$*, in every CCC.*

Proof.

The proof is divided into three sub-proofs, one for α-equivalence, one for β-equivalence, and the last one for η-equivalence.

(1) α-equivalence

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

Stronger form of α-equivalence: if terms $x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright M : \sigma$ and $y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright N : \sigma$ are well-typed, with $N =_\alpha [y_1, \ldots, y_n/x_1, \ldots x_n]M$, then $[\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright M : \sigma]\!] = [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright N : \sigma]\!]$.

This stronger form of α-equivalence can be proven by induction on the term $M : \sigma$. The base case is the one when M is a variable while the inductive steps contains two cases, application and abstraction.

Base case $M : \sigma \equiv x_i : \sigma_i$

$$
\begin{aligned}
& [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright [y_1, \ldots, y_n/x_1, \ldots x_n]x_i : \sigma_i]\!] \\
= \; & [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright y_i : \sigma_i]\!] \\
= \; & \mathrm{Proj}_{i+1}^{unit \times [\![\sigma_1]\!] \times \ldots \times [\![\sigma_n]\!]} \\
= \; & [\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright x_i : \sigma_i]\!]
\end{aligned}
$$

Inductive steps

Application $M : \sigma \equiv M_1 M_2 : \tau_2$

In this case, $N \equiv N_1 N_2$ where $N_1 =_\alpha [y_1, \ldots, y_n/x_1, \ldots x_n]M_1$ and $N_2 =_\alpha [y_1, \ldots, y_n/x_1, \ldots x_n]M_2$. The induction hypotheses contains two equations:

- $[\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright M_1 : \tau_1 \to \tau_2]\!] \quad = \quad [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright N_1 : \tau_1 \to \tau_2]\!]$
- $[\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright M_2 : \tau_1]\!] \quad = \quad [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright N_2 : \tau_1]\!]$

With the two equations above, we can show that

$$
\begin{aligned}
& [\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright M_1 M_2 : \tau_2]\!] \\
= \; & \mathrm{App} \circ \langle [\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright M_1 : \tau_1 \to \tau_2]\!], [\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright M_2 : \tau_1]\!] \rangle \\
= \; & \mathrm{App} \circ \langle [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright N_1 : \tau_1 \to \tau_2]\!], [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright N_2 : \tau_1]\!] \rangle \\
= \; & [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright N_1 N_2 : \tau_2]\!] \\
= \; & [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright [y_1, \ldots, y_n/x_1, \ldots x_n](M_1 M_2) : \tau_2]\!]
\end{aligned}
$$

Abstraction $M : \sigma \equiv \lambda x_{n+1} : \sigma_{n+1}.M' : \sigma_{n+1} \to \tau$

In this case, $N \equiv \lambda y_{n+1} : \sigma_{n+1}.N'$ where $N' =_\alpha [y_1, \ldots, y_{n+1}/x_1, \ldots x_{n+1}]M'$. We assume that $[\![x_1 : \sigma_1, \ldots, x_{n+1} : \sigma_{n+1} \triangleright M' : \tau]\!] = [\![y_1 : \sigma_1, \ldots, y_{n+1} : \sigma_{n+1} \triangleright N' : \tau]\!]$ holds. Then we prove that

$$\begin{aligned}
& [\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright \lambda x_{n+1} : \sigma_{n+1}.M' : \sigma_{n+1} \to \tau]\!] \\
= \;& \mathrm{Curry}([\![x_1 : \sigma_1, \ldots, x_{n+1} : \sigma_{n+1} \triangleright M' : \tau]\!]) \\
= \;& \mathrm{Curry}([\![y_1 : \sigma_1, \ldots, y_{n+1} : \sigma_{n+1} \triangleright N' : \tau]\!]) \\
= \;& [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright \lambda y_{n+1} : \sigma_{n+1}.N' : \sigma_{n+1} \to \tau]\!] \\
= \;& [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright [y_1, \ldots, y_n/x_1, \ldots x_n] (\lambda y_{n+1} : \sigma_{n+1}.M') : \sigma_{n+1} \to \tau]\!]
\end{aligned}$$

Hence, it holds that $[\![x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright M : \sigma]\!] = [\![y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright N : \sigma]\!]$ for all well typed $x_1 : \sigma_1, \ldots, x_n : \sigma_n \triangleright M : \sigma$ and $y_1 : \sigma_1, \ldots, y_n : \sigma_n \triangleright N : \sigma$ with $N =_\alpha [y_1, \ldots, y_n/x_1, \ldots x_n]M$.

(2) β-equivalence

To prove that the interpretation preserves β-equivalence, we need show that $\Gamma \triangleright (\lambda x : \sigma.M)N =_\beta [N/x]M : \tau \implies [\![\Gamma \triangleright (\lambda x : \sigma.M)N : \tau]\!] = [\![\Gamma \triangleright [N/x]M : \tau]\!]$.

$$\begin{aligned}
& [\![\Gamma \triangleright (\lambda x : \sigma.M)N : \tau]\!] \\
= \;& \mathrm{App} \circ \langle [\![\Gamma \triangleright \lambda x : \sigma.M : \sigma \to \tau]\!], [\![\Gamma \triangleright N : \tau]\!] \rangle \\
= \;& \mathrm{App} \circ \langle \mathrm{Curry}([\![\Gamma, x : \sigma \triangleright M : \tau]\!]), [\![\Gamma \triangleright N : \tau]\!] \rangle \\
& (\text{by } \langle f, g \rangle = (f \times \mathrm{id}) \circ \langle \mathrm{id}, g \rangle) \\
= \;& \mathrm{App} \circ ((\mathrm{Curry}([\![\Gamma, x : \sigma \triangleright M : \tau]\!]) \times \mathrm{id}) \circ \langle \mathrm{id}, [\![\Gamma \triangleright N : \tau]\!] \rangle) \\
= \;& (\mathrm{App} \circ (\mathrm{Curry}([\![\Gamma, x : \sigma \triangleright M : \tau]\!]) \times \mathrm{id})) \circ \langle \mathrm{id}, [\![\Gamma \triangleright N : \tau]\!] \rangle \\
& (\text{by } \mathrm{App} \circ (\mathrm{Curry}(h) \times \mathrm{id}) = h) \\
= \;& [\![\Gamma, x : \sigma \triangleright M : \tau]\!] \circ \langle \mathrm{id}, [\![\Gamma \triangleright N : \tau]\!] \rangle \\
& (\text{by Substitution Lemma}) \\
= \;& [\![\Gamma \triangleright [N/x]M : \tau]\!]
\end{aligned}$$

(3) η-equivalence

The preservation of η-equivalence can be represented by
$\Gamma \triangleright \lambda x : \sigma.Mx =_\eta M : \sigma \to \tau \implies [\![\Gamma \triangleright \lambda x : \sigma.Mx : \sigma \to \tau]\!] = [\![\Gamma \triangleright M : \sigma \to \tau]\!]$
for all $x \notin FV(M)$.

$$\begin{aligned}
& [\![\Gamma \triangleright \lambda x : \sigma.Mx : \sigma \to \tau]\!] \\
= \;& \mathrm{Curry}([\![\Gamma, x : \sigma \triangleright Mx : \tau]\!]) \\
= \;& \mathrm{Curry}(\mathrm{App} \circ \langle [\![\Gamma, x : \sigma \triangleright M : \sigma \to \tau]\!], [\![\Gamma, x : \sigma \triangleright x : \sigma]\!] \rangle) \\
& (\text{if } (\Gamma, x : \sigma)_f = \Gamma, \text{ then } \chi_f = \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\sigma]\!]}) \\
= \;& \mathrm{Curry}(\mathrm{App} \circ \langle [\![\Gamma \triangleright M : \sigma \to \tau]\!] \circ \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\sigma]\!]}, [\![\Gamma, x : \sigma \triangleright x : \sigma]\!] \rangle) \\
= \;& \mathrm{Curry}(\mathrm{App} \circ \langle [\![\Gamma \triangleright M : \sigma \to \tau]\!] \circ \mathrm{Proj}_1^{[\![\Gamma]\!],[\![\sigma]\!]}, \mathrm{Proj}_2^{[\![\Gamma]\!],[\![\sigma]\!]} \rangle) \\
& (\text{by } \mathrm{Curry}(\mathrm{App} \circ \langle k \circ \mathrm{Proj}_1, \mathrm{Proj}_2 \rangle) = k) \\
= \;& [\![\Gamma \triangleright M : \sigma \to \tau]\!]
\end{aligned}$$

Therefore, for any well-typed $\Gamma \triangleright M : \sigma$ and $\Gamma \triangleright N : \sigma$, it holds in every CCC that $\Gamma \triangleright M =_{\alpha\beta\eta} N : \sigma \implies [\![\Gamma \triangleright M : \sigma]\!] = [\![\Gamma \triangleright N : \sigma]\!]$.

$\square$

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

Proof.

The category $\mathcal{C}$ is generated by $\lambda^{\rightarrow}$ 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 $[\sigma_1, \ldots \sigma_n]$ represents the products of the $[\sigma_i]$. For notational convenience, we write $\vec{x}_k$ for a sequence $[x_1, \ldots, x_k]$ of k variables, similarly $\vec{\sigma}_k$ for a sequence of k type expressions, and $\vec{x}_k : \vec{\sigma}_k$ for type context $x_1 : \sigma_1, \ldots, x_k : \sigma_k$.

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 from $\vec{\sigma}_m$ to $\vec{\tau}_n$ is a n-tuple of equivalence classes of terms, i.e.

$$\begin{aligned} & [\vec{x}_m : \vec{\sigma}_m \triangleright M_i : \tau_i | i = 1, \ldots, n] \\ = \quad & [\vec{x}_m : \vec{\sigma}_m \triangleright M_1 : \tau_1, \ldots, \vec{x}_m : \vec{\sigma}_m \triangleright M_n : \tau_n] \\ = \quad & \langle \{\vec{x}_m : \vec{\sigma}_m \triangleright N_1 : \tau_1 | \vec{x}_m : \vec{\sigma}_m \triangleright M_1 = N_1 : \tau_1\}, \ldots, \\ & \qquad \{\vec{x}_m : \vec{\sigma}_m \triangleright N_n : \tau_n | \vec{x}_m : \vec{\sigma}_m \triangleright M_n = N_n : \tau_n\} \rangle \end{aligned}$$

Compositions in $\mathcal{C}$ is defined by substitution. Given a morphism from $\vec{\sigma}_l$ to $\vec{\tau}_m$ and another one from $\vec{\tau}_m$ to $\vec{\rho}_n$, their composition is

$$\begin{aligned} & [\vec{y}_m : \vec{\tau}_m \triangleright N_i : \rho_i | i = 1, \ldots, n] \circ [\vec{x}_l : \vec{\sigma}_l \triangleright M_i : \tau_i | i = 1, \ldots, m] \\ = \quad & [\vec{x}_l : \vec{\sigma}_l \triangleright [\vec{y}_m / \vec{M}_m] N_i : \rho_i | i = 1, \ldots, n] : \vec{\sigma}_l \rightarrow \vec{\rho}_n \end{aligned}$$

The identity for $\vec{\sigma}_n$ is the morphism $[\vec{x}_n : \vec{\sigma}_n \triangleright x_i : \sigma_i | i = 1, \ldots, n] : \vec{\sigma}_n \rightarrow \vec{\sigma}_n$.

After defining the category $\mathcal{C}$, we need to show that it is cartesian closed and then that two terms are $\alpha\beta\eta$-equivalent if they have the same interpretation in $\mathcal{C}$.

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

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

(i) Terminal object *unit* with morphism One

The empty sequence of types is the terminal object *unit*. According to the above definition, a morphism from an object to $\vec{\tau}_n$ is given by a n-tuple of terms. Then the morphism from an object to *unit*, the empty sequence, should be given by the empty tuple, i.e. $\text{One}^{\vec{\sigma}_k} = [\ ] : \vec{\sigma}_k \rightarrow unit$, for every object $\vec{\sigma}_k$. Since empty tuple is unique, $\text{One}^{\vec{\sigma}_k}$ is unique for every $\vec{\sigma}_k$.

(ii) Products with projections Proj_i and pairing $\langle \cdot, \cdot \rangle$

Given two object $\vec{\sigma}_m$ and $\vec{\tau}_n$, their product $\vec{\sigma}_m \times \vec{\tau}_n$ is obtained by their concatenation $[\sigma_1, \ldots, \sigma_m, \tau_1, \ldots, \tau_n]$. Then, the projection morphisms are defined by

$$\begin{aligned} \text{Proj}_1^{\vec{\sigma}_m, \vec{\tau}_n} &= [\vec{x}_m : \vec{\sigma}_m, \vec{y}_n : \vec{\tau}_n \triangleright x_i : \sigma_i | i = 1, \ldots, m] \quad : \vec{\sigma}_m \times \vec{\tau}_n \rightarrow \vec{\sigma}_m \\ \text{Proj}_2^{\vec{\sigma}_m, \vec{\tau}_n} &= [\vec{x}_m : \vec{\sigma}_m, \vec{y}_n : \vec{\tau}_n \triangleright y_i : \tau_i | i = 1, \ldots, n] \quad : \vec{\sigma}_m \times \vec{\tau}_n \rightarrow \vec{\tau}_n \end{aligned}$$

Given $[\vec{x}_l : \vec{\rho}_l \triangleright M_i : \sigma_i | i = 1, \ldots, m] : \vec{\rho}_l \rightarrow \vec{\sigma}_m$ and $[\vec{x}_l : \vec{\rho}_l \triangleright N_i : \tau_i | i = 1, \ldots, n] : \vec{\rho}_l \rightarrow \vec{\tau}_n$, their paring is defined by the concatenation of these two tuples

$$\begin{aligned}
& \langle[\vec{x}_l:\vec{\rho}_l \triangleright M_i:\sigma_i|i=1,\dots,m],[\vec{x}_l:\vec{\rho}_l \triangleright N_i:\tau_i|i=1,\dots,n]\rangle \\
= \;& [\vec{x}_l:\vec{\rho}_l \triangleright M_1:\sigma_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright M_m:\sigma_n,\vec{x}_l:\vec{\rho}_l \triangleright N_1:\tau_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright N_n:\tau_n] \\
& \qquad\qquad : \vec{\rho}_l \to \vec{\sigma}_m \times \vec{\tau}_n
\end{aligned}$$

In order to show that the products defined above are categorical products in $\mathcal{C}$, we should prove that they satisfy the equations in definition 2.3.3. For any

$$\begin{aligned}
f_1 &= [\vec{x}_l:\vec{\rho}_l \triangleright M_i:\sigma_i|i=1,\dots,m]:\vec{\rho}_l \to \vec{\sigma}_m, \\
f_2 &= [\vec{x}_l:\vec{\rho}_l \triangleright N_i:\tau_i|i=1,\dots,n]:\vec{\rho}_l \to \vec{\tau}_n, \text{and} \\
g &= [\vec{x}_l:\vec{\rho}_l \triangleright P_1:\sigma_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright P_m:\sigma_n,\vec{x}_l:\vec{\rho}_l \triangleright Q_1:\tau_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright Q_n:\tau_n] \\
& \qquad\qquad : \vec{\rho}_l \to \vec{\sigma}_m \times \vec{\tau}_n,
\end{aligned}$$

we have

$$\begin{aligned}
& \mathrm{Proj}_1^{\vec{\sigma}_m,\vec{\tau}_n} \circ \langle f_1, f_2\rangle \\
= \;& [\vec{y}_m:\vec{\sigma}_m,\vec{z}_n:\vec{\tau}_n \triangleright y_i:\sigma_i|i=1,\dots,m]\circ \\
& \qquad \langle[\vec{x}_l:\vec{\rho}_l \triangleright M_i:\sigma_i|i=1,\dots,m],[\vec{x}_l:\vec{\rho}_l \triangleright N_i:\tau_i|i=1,\dots,n]\rangle \\
= \;& [\vec{y}_m:\vec{\sigma}_m,\vec{z}_n:\vec{\tau}_n \triangleright y_i:\sigma_i|i=1,\dots,m]\circ \\
& \qquad [\vec{x}_l:\vec{\rho}_l \triangleright M_1:\sigma_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright M_m:\sigma_n,\vec{x}_l:\vec{\rho}_l \triangleright N_1:\tau_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright N_n:\tau_n] \\
= \;& [\vec{x}_l:\vec{\rho}_l \triangleright [\vec{y}_m,\vec{z}_n/\vec{M}_m,\vec{N}_n]y_i:\sigma_i|i=1,\dots,m] \\
= \;& [\vec{x}_l:\vec{\rho}_l \triangleright M_i:\sigma_i|i=1,\dots,m] \\
= \;& f_1
\end{aligned}$$

In the same way, we can show that $\mathrm{Proj}_2^{\vec{\sigma}_m,\vec{\tau}_n} \circ \langle f_1, f_2\rangle = f_2$.

$$\begin{aligned}
& \mathrm{Proj}_1^{\vec{\sigma}_m,\vec{\tau}_n} \circ g \\
= \;& [\vec{y}_m:\vec{\sigma}_m,\vec{z}_n:\vec{\tau}_n \triangleright y_i:\sigma_i|i=1,\dots,m]\circ \\
& \qquad [\vec{x}_l:\vec{\rho}_l \triangleright P_1:\sigma_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright P_m:\sigma_n,\vec{x}_l:\vec{\rho}_l \triangleright Q_1:\tau_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright Q_n:\tau_n] \\
= \;& [\vec{x}_l:\vec{\rho}_l \triangleright [\vec{y}_m,\vec{z}_n/\vec{P}_m,\vec{Q}_n]y_i:\sigma_i|i=1,\dots,m] \\
= \;& [\vec{x}_l:\vec{\rho}_l \triangleright P_i:\sigma_i|i=1,\dots,m]
\end{aligned}$$

Similarly, we also have $\mathrm{Proj}_2^{\vec{\sigma}_m,\vec{\tau}_n} \circ g = [\vec{x}_l:\vec{\rho}_l \triangleright Q_i:\tau_i|i=1,\dots,n]$. Then,

$$\begin{aligned}
& \langle \mathrm{Proj}_1^{\vec{\sigma}_m,\vec{\tau}_n} \circ g, \mathrm{Proj}_2^{\vec{\sigma}_m,\vec{\tau}_n} \circ g\rangle \\
= \;& \langle[\vec{x}_l:\vec{\rho}_l \triangleright P_i:\sigma_i|i=1,\dots,m],[\vec{x}_l:\vec{\rho}_l \triangleright Q_i:\tau_i|i=1,\dots,n]\rangle \\
= \;& [\vec{x}_l:\vec{\rho}_l \triangleright P_1:\sigma_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright P_m:\sigma_n,\vec{x}_l:\vec{\rho}_l \triangleright Q_1:\tau_1,\dots,\vec{x}_l:\vec{\rho}_l \triangleright Q_n:\tau_n] \\
= \;& g
\end{aligned}$$

(iii) Exponentials with currying Curry and evaluation App

Given two object $\vec{\sigma}_m$ and $\vec{\tau}_n$, their exponential $\vec{\sigma}_m \to \vec{\tau}_n$ is the type sequence of the patter $[\sigma_1 \to \cdots \to \sigma_m \to \tau_1,\dots,\sigma_1 \to \cdots \to \sigma_m \to \tau_n]$.

For every $h = [\vec{x}_l:\vec{\rho}_l,\vec{y}_m:\vec{\sigma}_m \triangleright M_i:\tau_i|i=1,\dots,n]:\vec{\rho}_l \times \vec{\sigma}_m \to \vec{\tau}_n$, the morphism $\mathrm{Curry}(h):\vec{\rho}_l \to (\vec{\sigma}_m \to \vec{\tau}_n)$ is defined by

$$
\begin{aligned}
& \mathrm{Curry}(h) \\
= \quad & \mathrm{Curry}([\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright M_i : \tau_i | i = 1, \ldots, n]) \\
= \quad & [\vec{x}_l : \vec{\rho}_l \triangleright \lambda \vec{y}_m : \vec{\sigma}_m . M_i : \sigma_1 \to \cdots \to \sigma_m \to \tau_i | i = 1, \ldots, n] : \vec{\rho}_l \to (\vec{\sigma}_m \to \vec{\tau}_n)
\end{aligned}
$$

Given $\vec{\sigma}_m$ and $\vec{\tau}_n$, the morphism $\mathrm{App}^{\vec{\sigma}_m, \vec{\tau}_n} : (\vec{\sigma}_m \to \vec{\tau}_n) \times \vec{\sigma}_m \to \vec{\tau}_n$ is defined by

$$
\begin{aligned}
& \mathrm{App}^{\vec{\sigma}_m, \vec{\tau}_n} \\
= \quad & [\vec{f}_n : \vec{\sigma}_m \to \vec{\tau}_n, \vec{x}_m : \vec{\sigma}_m \triangleright f_i \ x_1 \ldots x_m : \tau_i | i = 1, \ldots, n] : (\vec{\sigma}_m \to \vec{\tau}_n) \times \vec{\sigma}_m \to \vec{\tau}_n
\end{aligned}
$$

The following shows that the equations in definition 2.3.6 hold in $\mathcal{C}$ so that the exponentials defined above are exactly the categorical exponentials in $\mathcal{C}$. For any

$$
\begin{aligned}
h \quad = \quad & [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright M_i : \tau_i | i = 1, \ldots, n] : \vec{\rho}_l \times \vec{\sigma}_m \to \vec{\tau}_n \text{ and} \\
k \quad = \quad & [\vec{x}_l : \vec{\rho}_l \triangleright \lambda \vec{y}_m : \vec{\sigma}_m . N_i : \sigma_1 \to \cdots \to \sigma_m \to \tau_i | i = 1, \ldots, n] : \vec{\rho}_l \to (\vec{\sigma}_m \to \vec{\tau}_n)
\end{aligned}
$$

we have

$$
\begin{aligned}
& \mathrm{App}^{\vec{\sigma}_m, \vec{\tau}_n} \circ \langle \mathrm{Curry}(h) \circ \mathrm{Proj}_1^{\vec{\rho}_l, \vec{\sigma}_m}, \mathrm{Proj}_2^{\vec{\rho}_l, \vec{\sigma}_m} \rangle \\
= \quad & \mathrm{App}^{\vec{\sigma}_m, \vec{\tau}_n} \circ \langle \mathrm{Curry}([\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright M_i : \tau_i | i = 1, \ldots, n]) \circ \mathrm{Proj}_1^{\vec{\rho}_l, \vec{\sigma}_m}, \mathrm{Proj}_2^{\vec{\rho}_l, \vec{\sigma}_m} \rangle \\
= \quad & \mathrm{App}^{\vec{\sigma}_m, \vec{\tau}_n} \circ \langle [\vec{x}_l : \vec{\rho}_l \triangleright \lambda \vec{y}_m : \vec{\sigma}_m . M_i : \sigma_1 \to \cdots \to \sigma_m \to \tau_i | i = 1, \ldots, n] \circ \\
& \qquad [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright x_i : \rho_i | i = 1, \ldots, l], \mathrm{Proj}_2^{\vec{\rho}_l, \vec{\sigma}_m} \rangle \\
= \quad & \mathrm{App}^{\vec{\sigma}_m, \vec{\tau}_n} \circ \langle [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright [\vec{x}_l / \vec{x}_l](\lambda \vec{y}_m : \vec{\sigma}_m . M_i) : \sigma_1 \to \cdots \to \sigma_m \to \tau_i \\
& \qquad | i = 1, \ldots, n], [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright y_i : \sigma_i | i = 1, \ldots, m] \rangle \\
= \quad & \mathrm{App}^{\vec{\sigma}_m, \vec{\tau}_n} \circ \langle [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright \lambda \vec{y}_m : \vec{\sigma}_m . M_i : \sigma_i \to \cdots \to \sigma_m \to \tau_i, \\
& \qquad | i = 1, \ldots, n] [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright y_i : \sigma_i | i = 1, \ldots, m] \rangle \\
= \quad & [\vec{f}_n : \vec{\sigma}_m \to \vec{\tau}_n, \vec{z}_m : \vec{\sigma}_m \triangleright f_i \ z_1 \ldots z_m : \tau_i | i = 1, \ldots, n] \circ \\
& \qquad [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright \lambda \vec{y}_m : \vec{\sigma}_m . M_1 : \sigma_1 \to \cdots \to \sigma_m \to \tau_1, \ldots, \\
& \qquad \vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright \lambda \vec{y}_m : \vec{\sigma}_m . M_n : \sigma_1 \to \cdots \to \sigma_m \to \tau_n, \\
& \qquad \vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright y_1 : \sigma_1, \ldots, \vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright y_m : \sigma_m] \\
= \quad & [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright [\vec{f}_n, \vec{z}_m / \vec{N}_n, \vec{y}_m](f_i z_1 \ldots z_m) : \tau_i | i = 1, \ldots, n] \\
& (\text{where } \vec{N}_n \equiv \lambda \vec{y}_m : \vec{\tau}_m . M_1, \ldots, \lambda \vec{y}_m : \vec{\tau}_m . M_m) \\
= \quad & [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright (\lambda \vec{y}_m : \vec{\tau}_m . M_i) y_1 \ldots y_m : \tau_i | i = 1, \ldots, n] \\
= \quad & [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright [\vec{y}_m / \vec{y}_m] M_i : \tau_i | i = 1, \ldots, n] \\
= \quad & [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright M_i : \tau_i | i = 1, \ldots, n] \\
= \quad & h
\end{aligned}
$$

$$
\begin{aligned}
& \mathrm{Curry}(\mathrm{App}^{\vec{\sigma}_m,\vec{\tau}_n} \circ \langle k \circ \mathrm{Proj}_1^{\vec{\rho}_l,\vec{\sigma}_m}, \mathrm{Proj}_2^{\vec{\rho}_l,\vec{\sigma}_m}\rangle) \\
= \ & \mathrm{Curry}(\mathrm{App}^{\vec{\sigma}_m,\vec{\tau}_n} \circ \langle [\vec{x}_l : \vec{\rho}_l \triangleright \lambda \vec{y}_m : \vec{\sigma}_m.N_i : \sigma_1 \to \cdots \to \sigma_m \to \tau_i | i = 1,\ldots,n] \circ \\
& \qquad [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright x_i : \rho_i | i = 1,\ldots,l], \mathrm{Proj}_2^{\vec{\rho}_l,\vec{\sigma}_m}\rangle) \\
= \ & \mathrm{Curry}(\mathrm{App}^{\vec{\sigma}_m,\vec{\tau}_n} \circ \\
& \qquad \langle [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright [\vec{x}_l/\vec{x}_l](\lambda \vec{y}_m : \vec{\sigma}_m.N_i) : \sigma_1 \to \cdots \to \sigma_m \to \tau_i | i = 1,\ldots,n], \\
& \qquad [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright y_i : \sigma_i | i = 1,\ldots,m]\rangle) \\
= \ & \mathrm{Curry}(\mathrm{App}^{\vec{\sigma}_m,\vec{\tau}_n} \circ \\
& \qquad \langle [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright \lambda \vec{y}_m : \vec{\sigma}_m.N_i : \sigma_1 \to \cdots \to \sigma_m \to \tau_i | i = 1,\ldots,n], \\
& \qquad [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright y_i : \sigma_i | i = 1,\ldots,m]\rangle) \\
= \ & \mathrm{Curry}([\vec{f}_n : \vec{\sigma}_m \to \vec{\tau}_n, \vec{z}_m : \vec{\sigma}_m \triangleright f_i \ z_1 \ldots z_m : \tau_i | i = 1,\ldots,n] \circ \\
& \qquad [\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright \lambda \vec{y}_m : \vec{\sigma}_m.N_1 : \sigma_1 \to \cdots \to \sigma_m \to \tau_1, \ldots, \\
& \qquad \vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright \lambda \vec{y}_m : \vec{\sigma}_m.N_n : \sigma_1 \to \cdots \to \sigma_m \to \tau_n, \\
& \qquad \vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright y_1 : \sigma_1, \ldots, \vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright y_m : \sigma_m]) \\
= \ & \mathrm{Curry}([\vec{x}_l : \vec{\rho}_l, \vec{y}_m : \vec{\sigma}_m \triangleright N_i : \tau_i | i = 1,\ldots,n]) \\
= \ & [\vec{x}_l : \vec{\rho}_l \triangleright \lambda \vec{y}_m : \vec{\sigma}_m.N_i : \sigma_1 \to \cdots \to \sigma_m \to \tau_i | i = 1,\ldots,n] \\
= \ & k
\end{aligned}
$$

Therefore, the category $\mathcal{C}$ generated by $\lambda^{\to}$ is cartesian closed.

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

Given a well typed term $\Gamma \triangleright M : \tau$ in $\lambda^{\to}$, its interpretation in $\mathcal{C}$ obtained from (1) is its equivalence class.

We write Γ as $\vec{x}_n : \vec{\sigma}_n$, then its interpretation is the product of the interpretations of the types in it, according to definition 3.3.1. Products in $\mathcal{C}$ are defined as sequences of types; therefore, the interpretation of Γ in $\mathcal{C}$ is the sequence $\vec{\sigma}_n$ and the interpretation of the whole term should be a morphism from $\vec{\sigma}_n$ to $\vec{\tau}_1$ which is the single tuple of its equivalence class, i.e.

$$
\begin{aligned}
& \mathcal{C}[\![\Gamma \triangleright M : \tau]\!] \\
= \ & [\vec{x}_n : \vec{\sigma}_n \triangleright M : \tau] \\
= \ & \langle \{\vec{x}_n : \vec{\sigma}_n \triangleright N : \tau | \vec{x}_n : \vec{\sigma}_n \triangleright M = N : \tau\}\rangle : \vec{\sigma}_n \to \vec{\tau}_1
\end{aligned}
$$

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

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

□

4 Reflections

Doing a dissertation-only project was the most challenging but ambitious decision that I have made during my life in the University of Birmingham. As an overseas student, writing (in English) is definitely the most painful part of this project.

Though the correspondence seems to be straightforward, one needs to be very careful in the proofs. As far as some details are concerned, some terms may not correspond to proofs very tightly. The following is such a counter example.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x{:}\sigma \vdash x : \sigma}}{x{:}\sigma, y{:}\sigma \vdash x : \sigma}\,(add)}{x{:}\sigma \vdash \lambda y{:}\sigma.x : \sigma\to\sigma}\,(\to\mathrm{I})}{\vdash \lambda x{:}\sigma.\lambda y{:}\sigma.x : \sigma\to(\sigma\to\sigma)}\,(\to\mathrm{I}) \quad\Longrightarrow\quad \dfrac{\dfrac{\dfrac{\dfrac{}{\sigma \vdash \sigma}}{\sigma, \sigma \vdash \sigma}\,(add)}{\sigma \vdash \sigma \to \sigma}\,(\to\mathrm{I})}{\vdash \sigma \to (\sigma \to \sigma)}\,(\to\mathrm{I})$$

We erase all the terms in the derivation of $\lambda x{:}\sigma.\lambda y{:}\sigma.x : \sigma\to(\sigma\to\sigma)$. But the result does not look like a valid proof since the context, as a set, should not contain repeated propositions. One may want to adjust it to obtain a proof of "$\sigma \to (\sigma \to \sigma)$". However, we cannot rebuild the original term (or a term α-equivalent to the original one) from this proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\sigma \vdash \sigma}}{\vdash \sigma \to \sigma}\,(\to\mathrm{I})}{\sigma \vdash \sigma \to \sigma}\,(add)}{\vdash \sigma \to (\sigma \to \sigma)}\,(\to\mathrm{I}) \quad\Longrightarrow\quad \dfrac{\dfrac{\dfrac{\dfrac{}{x{:}\sigma \vdash x : \sigma}}{\vdash \lambda x{:}\sigma.x : \sigma\to\sigma}\,(\to\mathrm{I})}{y{:}\sigma \vdash \lambda x{:}\sigma.x : \sigma\to\sigma}\,(add)}{\vdash \lambda y{:}\sigma.\lambda x{:}\sigma.x : \sigma\to(\sigma\to\sigma)}\,(\to\mathrm{I})$$

One of the solutions to this problem is linear logic which is proposed as a refinement of classical and intuitionistic logic. But linear logic is not involved in this project.

During this project, most of the time was spent in learning foundations of category theory. It was difficult to show that $\mathrm{Pos}_{\perp!}$ (the category of posets with bottom preserving maps given in section 2.3.3) is not cartesian closed. Even though the supervisor allowed me to leave this tough question, I kept trying to solve it for several weeks. However, I was just able to show that the exponentials from Pos are not categorical exponentials in $\mathrm{Pos}_{\perp!}$. Finally, this question was overcome under the guidance of the supervisor.

The literature [1, 3, 9, 10] I have read, interprets $\lambda^{unit,\times,\to}$ terms in cartesian closed categories and shows the cartesian closed structure of the category generated from $\lambda^{unit,\times,\to}$. As discussed at the beginning of section 3.3, their connection seems to be obvious. People are much more interested in revealing the obscure connection; therefore, with the advice of the supervisor, it was decided to prove the correspondence between $\lambda^{\to}$ and CCCs, which is the main part of this project as well as my own contribution to this project.

Cartesian closed categories have several properties of importance, i.e. some special equations which hold in every CCC. Most categorical literature utilises diagrams to give proofs to these equations. However, beginners may find it difficult to see equations through diagrams. This dissertation provides both diagram-based and equation-based proofs to the equations. Both ways have their pros and cons. Given a complicated equation, its corresponding diagram may be too large to draw on an A4 paper. The equational ones provide proof step by step, which always makes the proof tedious and hard to remember. What is a good way to present proofs? This is another tough question I faced in this project.

All in all, it is really enjoyable to be immersed in this research-oriented project. For one thing, it increased my thirst for knowledge in theoretical computer science as well as improved my skills in research. For another, the connection between proof theory, type theory and category theory did amaze me. When looking at different mathematical subjects, I keep asking myself whether they can be linked to each other.

References

[1] A. Asperti and G. Longo. *Categories, Types, and Structures.* The MIT Press, 1991.

[2] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Elsevier, revised edition, 1985.

[3] M. Barr and C. Wells. *Category Theory for Computing Science.* Prentice Hall, 2nd edition, 1995.

[4] S. Eilenberg and S. M. Lane. General theory of natural equivalences. *Transactions of the American Mathematical Society*, 58(2), 1945.

[5] J. R. Hindley. *Basic Simple Type Theory.* Cambridge University Press, 1997.

[6] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems.* Cambridge University Press, 2nd edition, 2004.

[7] A. Jung. A short introduction to the lambda calculus, March 2004. Notes from the second year course *Models of Computation*, taught at the University of Birmingham, 98/99 to 03/04.

[8] J. Lambek. *Lecture Notes on Mathematics*, volume 274, chapter Deductive Systems and Categories III: Cartesian Closed Categories, Intuitionist Propositional Calculus, and Combinatory Logic. Springer, 1972.

[9] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*, chapter Cartesian closed categories and λ-calculus. Cambridge University Press, 1986.

[10] J. C. Mitchell. *Foundations for Programming Languages.* The MIT Press, 1996.

[11] B. C. Pierce. *Basic Category Theory for Computer Science.* The MIT Press, 1991.

[12] A. M. Pitts. *Handbook of Logic in Computer Science*, volume 5, chapter Categorical Logic. Clarendon Press, 2000.

[13] W. Ruitenburg. *Perspectives on the history of mathematical Logic*, chapter The Unintended Interpretations of Intuitionistic Logic. Birkhauser, 1991.

[14] D. E. Rydeheard and R. M. Burstall. *Computational category Theory.* Prentice Hall, 1988.

[15] M. H. B. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism.* Elsevier, 1998.

[16] D. van Dalen. *Logic and Structure.* Springer, 3rd edition, 1997.

[17] D. van Dalen. *Philosophical Logic*, chapter Intuitionistic Logic. Blackwell, 2001.