ON GEOMETRY OF INTERACTION CONSTRUCTION AND ITS APPLICATIONS

Geometry of Interaction construction とその応用について

by

Akira Yoshimizu
由水 輝

A Senior Thesis
卒業論文

Submitted to
the Department of Information Science
the Faculty of Science, the University of Tokyo
on February 7, 2012
in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science

Thesis Supervisor: Ichiro Hasuo　蓮尾 一郎
Lecturer of Information Science

ABSTRACT

Recently, a new logical system *linear logic* has been developed and come to be used to model various kinds computation. *Geometry of Interaction* (GoI) is a model of its cut-elimination, and GoI is also employed in such modelings. Since categorical interpretation (*GoI construction*) is given for GoI, it enables us to express notions in abstract and general way. Especially, there is a result that models quantum functional programming language using GoI. We survey notions on this work, supplementing proofs and propositions.

論文要旨

近年、新しい論理体系である線型論理が提案され、種々の計算のモデル化に使用されるようになった。線型論理の cut-elimination のモデルである Geometry of Interaction (GoI) に対しては圏論的な解釈が与えられており、これにより一般的かつ抽象化された議論が可能となっている。その応用の一つとして、 GoI を用いた関数型の量子計算プログラミング言語の表示的意味論の構成が挙げられる。本論文はこの研究の基礎となっている概念の一部を纏め、証明の補完および命題の追加を行ったものである。

Acknowledgements

Sincere thanks are due to our supervisor Ichiro Hasuo. We also express gratitude to Ryo Kashima and Shintaro Fukushima for providing an opportunity to refine upon a part of discussions.

Contents

Chapter 1

Introduction

Modeling of computations is a fundamental part of computer science. λ-calculus plays an important role in this area. It is known as Curry-Howard isomorphism that various notions in λ-calculus corresponds to certain notions in logic. Well-known one of such a logic is intuitionistic logic. Another logical system, *linear logic*, was introduced by Girard in his paper [9] short while ago. Linear logic has enough expressiveness to construct full part of intuitionistic or classical logic in it. Besides, its counterpart in λ-calculus side has more preferable nature for modeling some special kinds of computation than previous logical systems, or its corresponding λ-calculi. One good example is *quantum computation*. In the paper [14], it was presented that the first denotational model for quantum functional programming language with full required features, based on techniques stemming from linear logic. Those techniques, which interpret *Geometry of Interaction* categorically, were given in [1]. Geometry of Interaction (GoI for short) is a model of cut-elimination in linear logic, invented again by Girard ([11, 10, 12]). Recently another utilization of GoI, called *Geometry of Synthesis*, was presented by Ghica in [5] and subsequent papers [6, 7, 8]. It is a method to compile programs written in a certain programming language into actual digital circuits.

These two results are both based on GoI, thus they probably can be combined. Namely, there is a possibility to develop a method to compile quantum programs into quantum circuits. This is the aim of our current research.

In this paper we survey notions on which the papers [14] and [1] are based. Particularly, we study *linear combinatory algebra* and the category of *partial equivalence relations* on it, adding some propositions. Several proofs omitted in our references are also added.

Chapter 2

Preliminaries

2.1 Category Theory

In this section we recall an elementary part of category theory. A standard textbook in this area is [16].

Definition 2.1.1 (Category)**.** A *category* $\mathbf{C}$ consists of a collection of *objects* and a collection of *arrows* (also called *morphisms* or *maps*) with following operations and properties:

- There exist two operations Dom and CoDom for arrows. Given an arrow f, each $\mathrm{Dom}(f)$ and $\mathrm{CoDom}(f)$ is an object. If $\mathrm{Dom}(f) = X$ and $\mathrm{CoDom}(f) = Y$, the arrow f is written as $f : X \to Y$.
- Given two arrows $f : X \to Y$ and $g : Y \to Z$, there exists an arrow $h : X \to Z$ determined by f and g. The arrow h is written as $g \circ f$, called the *composition* of f and g.
- Given three arrows $f : X \to Y$, $g : Y \to Z$, and $h : Z \to W$, their compositions are associative, i.e. $h \circ (g \circ f) = (h \circ g) \circ f : X \to W$. Thus we can write this arrow without parentheses, as $h \circ g \circ f$.
- Given an object X, there exists an *identity arrow* $id_X : X \to X$ satisfying:
  - For any arrow $f : X \to Y$, $f \circ id_X = f$.
  - For any arrow $g : Y \to X$, $id_X \circ g = g$.

The collections of objects and arrows of $\mathbf{C}$ are sometimes written as $obj(\mathbf{C})$ and $arr(\mathbf{C})$ respectively.

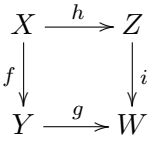
Definition 2.1.2. An arrow $e : X \to Y$ in a category $\mathbf{C}$ is called an *isomorphism* if there exists an arrow $e^{-1} : Y \to X$ that satisfies $e^{-1} \circ e = id_X$ and $e \circ e^{-1} = id_Y$. Objects X and Y are said to be *isomorphic* if there exist such isomorphisms. They are denoted by $e : X \cong Y : e^{-1}$. An identity arrow $id_X : X \to X$ itself is an isomorphism with $id_X^{-1} = id_X$.

Notation. The collection of all arrows from an object X to an object Y in a category $\mathbf{C}$ is often denoted by $\mathbf{C}(X, Y)$. The composition operator $\circ$ in a composite $f \circ g$ is often omitted, as fg.

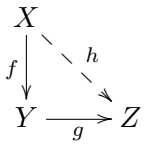
Example. The category of sets, **Sets**. Its objects are all sets, and its arrows are all maps between sets. The composition of arrows is ordinary map composition, the identity arrow for a set is the identity map.

Example. The category of sets and partial functions, **Pfn**. Its objects are all sets, its arrows are all partial functions between sets. The composition of arrows and the identity arrows are the same as in **Sets**.

Notation. Equations of arrows are frequently described as a kind of directed graphs, called *commutative diagrams*. For example, the diagram below shows that $g \circ f = i \circ h$. Conversely, if $g \circ f = i \circ h$ holds then one say that the diagram *commutes*.



Dashed arrow used in a commutative diagram designates the uniqueness of the arrow. Seeing the diagram below, we can tell that the arrow $X \to Z$ that makes the diagram commute is unique.



The following two definitions are generalized notions of Cartesian product and disjoint union of sets, respectively.

Definition 2.1.3. Let X and Y be objects of a category $\mathbf{C}$. A *binary product* of X and Y, denoted by $X \times Y$, is an object of $\mathbf{C}$ that satisfies following properties:

- There exist two arrows $\pi_1 : X \times Y \to X$, $\pi_2 : X \times Y \to Y$.
- Given an object Z and two arrows $f : Z \to X$ and $g : Z \to Y$, there uniquely exists an arrow $\langle f, g \rangle : Z \to X \times Y$ that satisfies $\pi_1 \circ \langle f, g \rangle = f$ and $\pi_2 \circ \langle f, g \rangle = g$.

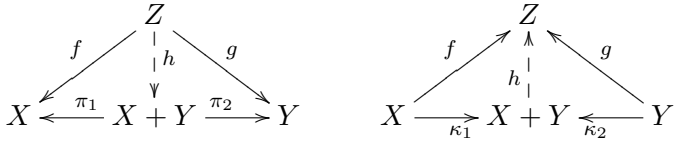
A category $\mathbf{C}$ is said to *have binary products* if there exists a binary product for arbitrary objects X and Y.

Definition 2.1.4. Let X and Y be objects of a category $\mathbf{C}$. The *binary coproduct* of X and Y, denoted by $X + Y$, is an object of $\mathbf{C}$ which satisfies following properties:

- There exist two arrows $\kappa_1 : X \to X + Y$, $\kappa_2 : Y \to X \times Y$.
- Given an object Z and two arrows $f : X \to Z$ and $g : Y \to Z$, there uniquely exists an arrow $[f, g] : X + Y \to Z$ that satisfies $[f, g] \circ \kappa_1 = f$ and $[f, g] \circ \kappa_2 = g$.

A category $\mathbf{C}$ is said to *have binary coproducts* if there exists a binary coproduct for arbitrary objects X and Y.

These two definitions are very similar. Actually, if we reverse all arrows of the one, we get the other notion. This is clearer in the following two commutative diagrams; the left one is the condition required for a binary product, and the right is one for a binary coproduct.



Such two notions are said to be *dual* to each other. Let us see one more pair of dual notions: *initial object* and *terminal object*.

Definition 2.1.5. An object A of a category $\mathbf{C}$ is called an *initial object* of $\mathbf{C}$ if for any object X the arrow $A \to X$ uniquely exists.

Definition 2.1.6. An object Z of a category $\mathbf{C}$ is called a *terminal object* of $\mathbf{C}$ if for any object X the arrow $X \to A$ uniquely exists.

In the category **Sets**, the initial object is the empty set $\emptyset$, the terminal object is any singleton set $\{x\}$.

A relation between two categories can be described by a *functor*, a kind of map from a category to another.

Definition 2.1.7. Let $\mathbf{C}$ and $\mathbf{D}$ be categories. A *functor* F from $\mathbf{C}$ to $\mathbf{D}$ (denoted by $F : \mathbf{C} \to \mathbf{D}$) is a map that satisfies following properties:

- For any object X in $\mathbf{C}$, FX is an object in $\mathbf{D}$.
- For any arrow $f : X \to Y$ in $\mathbf{C}$, $Ff : FX \to FY$ is an arrow in $\mathbf{D}$
- For any arrows $f : X \to Y$ and $g : Y \to Z$ in $\mathbf{C}$, $F(g \circ f) = Fg \circ Ff$ in $\mathbf{D}$.
- For any identity arrow $id_X : X \to X$ in $\mathbf{C}$, $F(id_X) = id_{FX}$ in $\mathbf{D}$.

Given another functor $G : \mathbf{D} \to \mathbf{E}$, the *composition* of F and G can be defined: $(GF)X = G(FX)$ for objects, $(GF)f = G(Ff)$ for arrows. Indeed, categories as objects and functors as arrows also form a category.

In a similar manner, a *bifunctor* $\square$ from two categories $\mathbf{C}$ and $\mathbf{D}$ to a category $\mathbf{E}$ can be defined. Infix notation is often used for a bifunctor as follows:

$$\begin{array}{ccc} X & Z & X \square Z \\ {\scriptstyle f}\downarrow & {\scriptstyle g}\downarrow & {\scriptstyle f \square g}\downarrow \\ Y & W & Y \square W \end{array}$$

Example. Identity functor, $Id_{\mathbf{C}}$. Every object and every arrow in $\mathbf{C}$ is sent to itself by $Id_{\mathbf{C}}$.

Example. Constant X functor, also written as X. It arrows every object to the object X, every arrow to the identity arrow Id_X.

Moreover, the notion of map from a functor to a functor is defined.

Definition 2.1.8. Let $\mathbf{C}$ and $\mathbf{D}$ be categories, and $F, G : \mathbf{C} \to \mathbf{D}$ be functors. A *natural transformation* α from F to G, denoted by $\alpha : F \Rightarrow G$, is a family of maps that satisfies following properties:

- For any object X in $\mathbf{C}$, there exists an arrow $\alpha_X : FX \to GX$.
- For any objects X, Yin $\mathbf{C}$ and any arrow $f : X \to Y$, arrows α_X and α_Y satisfy $Gf \circ \alpha_X = \alpha_Y \circ Ff$.

$$\begin{array}{ccc} FX & \xrightarrow{Ff} & FY \\ {\scriptstyle \alpha_X}\downarrow & & \downarrow{\scriptstyle \alpha_Y} \\ GX & \xrightarrow{Gf} & GY \end{array}$$

Each arrow α_X is called a *component* of α. If every component α_X is an isomorphism, the natural transformation α is called a *natural isomorphism*.

With natural transformations, we can discuss the *equivalence* of categories.

Definition 2.1.9. Let $\mathbf{C}$ and $\mathbf{D}$ be categories, $F : \mathbf{C} \to \mathbf{D}$ and $F : \mathbf{D} \to \mathbf{C}$ be functors. Categories $\mathbf{C}$ and $\mathbf{D}$ are said to be *equivalent* if there exist two natural transformations $FG \Rightarrow Id_{\mathbf{D}}$ and $GF \Rightarrow Id_{\mathbf{C}}$. If $FG = Id_{\mathbf{D}}$ and $GF = Id_{\mathbf{C}}$, categories $\mathbf{C}$ and $\mathbf{D}$ are said to be *isomorphic*.

2.2 Linear Logic

Linear logic is a logical system introduced by Girard, in his paper [9]. It is, as he said, "logic behind logic." Both classical and intuitionistic logic can be redefined in linear logic. In those definitions, each of classical/intuitionistic primitives is decomposed into two or more primitives of linear logic. For example, a classical logical formula $A \Rightarrow B$ (where A and B are atomic) is defined in linear logic as $(!\,A) \multimap (!\,B)$, where $!$ and $\multimap$ are linear primitives.

A formula $!\,A$, with the modality $!$ (called *of course* modality or *bang* modality), represents that "the formula A can be used as many times as we would like to". So the formula $(!\,A) \multimap (!\,B)$ means "we can use many A to derive B, and B also can be used repeatedly", which coincides to the meaning of the classical formula $A \Rightarrow B$. Similarly, a formula $A \Rightarrow B$ in intuitionistic logic is decomposed into $(!\,A) \multimap B$ in linear logic. These translations can be found in [4].

On the other hand, a formula A, without the $!$ modality, expresses that "the formula A must be used just once, neither zero times nor multiple times". Thus we can control assumptions that will be used only once (said to be *linear*) and that many times (*non-linear*), by using the $!$ modality.

Another important property of linear logic is its duality. Negation in intuitionistic logic is not involutive , i.e. $\neg\neg A \neq A$. Although intuitionistic logic can be reconstructed in linear logic, negation in linear logic (*linear negation*, denoted by superscripting $\perp$) is fully involutive, i.e. $A^{\perp\perp} = A$.

These properties were originally obtained from logical investigation. However, it has been recognized that these duality and controllability of linearity are suitable for modeling fairly many other areas, such as parallel computation, game semantics, database, and quantum computation. The distinction between linear and nonlinear expressions also leads to discussion on efficiency of computation. An interesting example is *light linear logic*, a fragment of linear logic introduced in [13], of which all cut-eliminations are in polynomial time.

There is another interesting viewpoint. Logic and λ-calculi are related each other, in the well-known Curry-Howard isomorphism. For instance, a proof of intuitionistic logic corresponds to a term of typed λ-calculus, normalization of proofs of intuitionistic logic is equivalent to reduction of terms of typed λ-calculus, etc. Similarly, there exists a variant of λ-calculus that corresponds to linear logic – *linear λ-calculus*. In linear λ-calculus, the abstraction is split into two, linear one and non-linear one. A formalization of it is in [21].

Combinatory algebra (or combinatory logic) is also equivalent to λ-calculus (detailed examination can be found in [3]), and a typed combinatory algebra corresponds to a Hilbert-style system of logic. There in turn exists linear variant of combinatory algebra, called *linear combinatory algebra*, which corresponds to a linear lambda calculus and a Hilbert-style system of linear logic.

Linear combinatory algebra appears in a process of modeling some kind of computation. That framework is called *Geometry of Interaction* (abbreviated as

GoI), originally invented by Girard [11, 10, 12] in order to give a new model of cut-elimination. Since cut-elimination in logic corresponds to β-reduction in λ-calculus via Curry-Howard isomorphism, GoI is also useful tool for analysis of computation. Categorical interpretation of GoI was first given by Abramsky and Jagadeesan in [2], and later in [1] Abramsky et al. showed that traced symmetric monoidal categories under certain conditions give rise to a categorical model with linear combinatory algebra. In the next chapter we see this categorical construction.

Chapter 3

GoI Construction

3.1 Monoidal Categories

An important example of categories are monoids. A monoid can be regarded as a category, and there is a generalized notion of monoid in category theory: *monoidal category*. In [1], Abramsky et al. presented a categorical framework of GoI with one of variations of monoidal categories, *traced symmetric monoidal monoidal category*. Thus we first follow the definitions.

Definition 3.1.1 (Monoidal Category)**.** A *(relaxed) monoidal category* is a category $\mathbf{C}$ equipped with:

- A bifunctor $\otimes : \mathbf{C} \times \mathbf{C} \to \mathbf{C}$. This corresponds to a binary operation on a monoid. It is sometimes called the *tensor*.
- An object I (called *unit*). This corresponds to monoid unit.
- A natural isomorphism $\alpha = \{\alpha_{X,Y,Z} : X \otimes (Y \otimes Z) \cong (X \otimes Y) \otimes Z \,|\, X, Y, Z \in obj(\mathbf{C})\}$ that satisfies the following coherence condition for all $X, Y, Z, W \in obj(\mathbf{C})$:

$$\begin{array}{ccc} X \otimes (Y \otimes (Z \otimes W)) & \xrightarrow{id_X \otimes \alpha_{Y,Z,W}} & X \otimes ((Y \otimes Z) \otimes W) \\ \downarrow{\scriptstyle \alpha_{X,Y,(Z \otimes W)}} & & \\ (X \otimes Y) \otimes (Z \otimes W) & & \downarrow{\scriptstyle \alpha_{X,(Y \otimes Z),W}} \\ \downarrow{\scriptstyle \alpha_{(X \otimes Y),Z,W}} & & \\ ((X \otimes Y) \otimes Z) \otimes W & \xleftarrow{\alpha_{X,Y,Z} \otimes id_W} & (X \otimes (Y \otimes Z)) \otimes W \end{array}$$

  This property represents associativity.

- A natural isomorphism $\lambda = \{\lambda_X : I \otimes X \cong X \,|\, X \in obj(\mathbf{C})\}$ and another natural isomorphism $\rho = \{\rho_X : X \otimes I \cong X \,|\, X \in obj(\mathbf{C})\}$ that satisfies $\lambda_I = \rho_I : I \otimes I \cong I$ and the following commutativity for all objects X, Y:

$$\begin{array}{ccc} X \otimes (I \otimes Y) & \xrightarrow{\alpha_{X,I,Y}} & (X \otimes I) \otimes Y \\ \downarrow{\scriptstyle id_X \otimes \lambda_Y} & & \downarrow{\scriptstyle \rho_X \otimes id_Y} \\ X \otimes Y & = & X \otimes Y \end{array}$$

  The existence of these two natural isomorphisms λ and ρ corresponds to left and right unit law of monoid.

A monoidal category $\mathbf{C}$ is called a *strict monoidal category* if all $\alpha_{X,Y,Z}, \lambda_X, \rho_X$ are identities for arbitrary objects X, Y, Z.

There are extensions of this basic notion. Functors and natural transformations are also extended to monoidal ones.

Definition 3.1.2 (Braided Monoidal Category and Symmetric Monoidal Category)**.** A monoidal category $\mathbf{C}$ is called a *braided monoidal category* if it is equipped with a natural isomorphism $c = \{c_{X,Y} : X \otimes Y \to Y \otimes X \,|\, X, Y \in obj(\mathbf{C})\}$, which makes the following two diagrams commute.

$$\begin{array}{ccc}
(X \otimes Y) \otimes Z & \xrightarrow{\alpha_{X,Y,Z}} & X \otimes (Y \otimes Z) \\
\downarrow{\scriptstyle c_{X,Y} \otimes id_Z} & & \downarrow{\scriptstyle c_{X,Y \otimes Z}} \\
(Y \otimes X) \otimes Z & & (Y \otimes Z) \otimes X \\
\downarrow{\scriptstyle \alpha_{Y,X,Z}} & & \downarrow{\scriptstyle \alpha_{Y,Z,X}} \\
Y \otimes (X \otimes Z) & \xrightarrow{id_Y \otimes c_{X,Z}} & Y \otimes (Z \otimes X)
\end{array}$$

$$\begin{array}{ccc}
X \otimes (Y \otimes Z) & \xrightarrow{\alpha^{-1}_{X,Y,Z}} & (X \otimes Y) \otimes Z \\
\downarrow{\scriptstyle id_X \otimes c_{Y,Z}} & & \downarrow{\scriptstyle c_{X \otimes Y,Z}} \\
X \otimes (Z \otimes Y) & & Z \otimes (X \otimes Y) \\
\downarrow{\scriptstyle \alpha^{-1}_{X,Z,Y}} & & \downarrow{\scriptstyle \alpha^{-1}_{Z,X,Y}} \\
(X \otimes Z) \otimes Y & \xrightarrow{c_{X,Z} \otimes id_Y} & (Z \otimes X) \otimes Y
\end{array}$$

The natural isomorphism c is called a *braiding* or a *commutativity constraint*. The existence of c corresponds to the property that the binary operation $\otimes$ is commutative. If $c_{Y,X} \circ c_{X,Y} = id_{X \otimes Y}$ holds for arbitrary objects X, Y, a braided monoidal category $\mathbf{C}$ is called a *symmetric monoidal category*.

Definition 3.1.3 (Monoidal Functor)**.** A *monoidal functor* from a monoidal category $(\mathbf{C}, \otimes_{\mathbf{C}}, I_{\mathbf{C}})$ to a monoidal category $(\mathbf{D}, \otimes_{\mathbf{D}}, I_{\mathbf{D}})$ is a functor $F : \mathbf{C} \to \mathbf{D}$ with a natural transformation $\phi = \{\phi_{X,Y} : FX \otimes_{\mathbf{D}} FY \to F(X \otimes_{\mathbf{C}} Y) \,|\, X, Y \in obj(\mathbf{C})\}$ and an arrow $\phi_I : I_{\mathbf{D}} \to FI_{\mathbf{C}}$ in $\mathbf{D}$, satisfying the following commutativity for all objects X, Y, Z in $\mathbf{C}$.

$$\begin{array}{ccc}
(FX \otimes_{\mathbf{D}} FY) \otimes_{\mathbf{D}} FZ & \xrightarrow{\alpha_{FX,FY,FZ}} & FX \otimes_{\mathbf{D}} (FY \otimes_{\mathbf{D}} FZ) \\
\downarrow{\scriptstyle \phi_{X,Y} \otimes_{\mathbf{D}} id_{FZ}} & & \downarrow{\scriptstyle id_{FX} \otimes \phi_{Y,Z}} \\
F(X \otimes_{\mathbf{C}} Y) \otimes_{\mathbf{D}} FZ & & FX \otimes_{\mathbf{D}} F(Y \otimes_{\mathbf{C}} Z) \\
\downarrow{\scriptstyle \phi_{X \otimes_{\mathbf{C}} Y,Z}} & & \downarrow{\scriptstyle \phi_{X,Y \otimes_{\mathbf{C}} Z}} \\
F((X \otimes_{\mathbf{C}} Y) \otimes_{\mathbf{C}} Z) & \xrightarrow{F(\alpha_{X,YZ})} & F(X \otimes_{\mathbf{C}} (Y \otimes_{\mathbf{C}} Z))
\end{array}$$

$$\begin{array}{ccc}
FX \otimes_{\mathbf{D}} I_{\mathbf{D}} & \xrightarrow{id_{FX} \otimes_{\mathbf{D}} \phi_I} & FX \otimes_{\mathbf{D}} FI_{\mathbf{C}} \\
\downarrow{\scriptstyle \rho_{\mathbf{D}}} & & \downarrow{\scriptstyle \phi_{X,I_{\mathbf{C}}}} \\
FX & \xleftarrow[F\rho_{\mathbf{C}}]{} & F(X \otimes_{\mathbf{C}} I_{\mathbf{C}})
\end{array}
\qquad
\begin{array}{ccc}
I_{\mathbf{D}} \otimes_{\mathbf{D}} FX & \xrightarrow{\phi_I \otimes_{\mathbf{D}} id_{FX}} & FI_{\mathbf{C}} \otimes_{\mathbf{D}} FX \\
\downarrow{\scriptstyle \lambda_{\mathbf{D}}} & & \downarrow{\scriptstyle \phi_{I_{\mathbf{C}},X}} \\
FX & \xleftarrow[F\lambda_{\mathbf{C}}]{} & F(I_{\mathbf{C}} \otimes_{\mathbf{C}} X)
\end{array}$$

If $\mathbf{C}$ and $\mathbf{D}$ are symmetric categories and the functor $F : \mathbf{C} \to \mathbf{D}$ makes the diagram below commute, the functor F is called a *symmetric monoidal functor*:

$$\begin{array}{ccc} FX \otimes_{\mathbf{D}} FY & \xrightarrow{c_{FX,FY}} & FY \otimes_{\mathbf{D}} FX \\ \downarrow{\scriptstyle \phi_{X,Y}} & & \downarrow{\scriptstyle \phi_{X,Y}} \\ F(X \otimes_{\mathbf{C}} Y) & \xrightarrow{F(c_{X,Y})} & F(Y \otimes_{\mathbf{C}} X) \end{array} .$$

Definition 3.1.4 (Monoidal Natural Transformation)**.** For monoidal categories $(\mathbf{C}, \otimes_{\mathbf{C}}, I_{\mathbf{C}})$ and $(\mathbf{D}, \otimes_{\mathbf{D}}, I_{\mathbf{D}})$, monoidal functors (F, ϕ, ϕ_I) and (G, ψ, ψ_I) from $\mathbf{C}$ to $\mathbf{D}$, a *monoidal natural transformation* $\theta : (F, \phi, \phi_I) \Rightarrow (G, \psi, \psi_I)$ is a natural transformation $\theta : F \Rightarrow G$ that satisfies following commutativity for arbitrary objects X and Y:

$$\begin{array}{ccc} FX \otimes_{\mathbf{D}} FY & \xrightarrow{\theta_X \otimes_{\mathbf{D}} \theta_Y} & GX \otimes_{\mathbf{D}} GY \\ \downarrow{\scriptstyle \phi_{X,Y}} & & \downarrow{\scriptstyle \psi_{X,Y}} \\ F(X \otimes_{\mathbf{C}} Y) & \xrightarrow{\theta_{X \otimes_{\mathbf{C}} Y}} & G(X \otimes_{\mathbf{C}} Y) \end{array} \qquad \begin{array}{ccc} & I_{\mathbf{D}} & \\ {\scriptstyle \phi_I} \swarrow & & \searrow {\scriptstyle \psi_I} \\ FI_{\mathbf{C}} & \xrightarrow{\theta_{I_{\mathbf{C}}}} & GI_{\mathbf{C}} \end{array}$$

A *monoidal pointwise natural transformation* $\theta' : F \Rightarrow G$ is a family of arrows that makes the following diagram commute for any arrow $f : I \to X$, not necessarily for the arrows from any other object than I:

$$\begin{array}{ccc} FI & \xrightarrow{\theta'_I} & GI \\ \downarrow{\scriptstyle Ff} & & \downarrow{\scriptstyle Gf} \\ FX & \xrightarrow{\theta'_X} & GX \end{array}$$

Definition 3.1.5 (Monoidal Equivalence)**.** Two monoidal categories $\mathbf{C}$ and $\mathbf{D}$ are said to be *monoidally equivalent* if they are equivalent via monoidal functors $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$, monoidal isomorphisms $FG \Rightarrow Id_{\mathbf{D}}$ and $GF \Rightarrow Id_{\mathbf{C}}$.

Now we can state an important theorem.

Theorem 3.1.6. *Every monoidal category is monoidally equivalent to a strict monoidal category.*

We do not show the proof here. Interested readers are referred to [19]. Due to this theorem, one may regard any monoidal caterory as strict one.

The notion of *traced monoidal category* was introduced by Joyal et al. in [15]. Here we use the definition of *traced symmetric monoidal category*, which is its symmetric version.

Definition 3.1.7 (Traced Symmetric Monoidal Category)**.** A *traced symmetric monoidal category* is a symmetric monoidal category $(\mathbf{C}, \otimes, I, \alpha, \lambda, \rho, c)$ that has an arrow $\mathsf{Tr}^U_{X,Y}(f) : X \to Y$ for any arrow $f : X \otimes U \to Y \otimes U$ satisfying the following conditions:

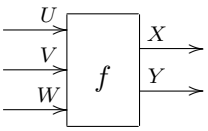
- Natural in X: $(\mathsf{Tr}^U_{X,Y}(f)) \circ g = \mathsf{Tr}^U_{X',Y}(f \circ (g \otimes id_U))$ for any $f : X \otimes U \to Y \otimes U$ and $g : X' \to X$.
- Natural in Y: $g \circ (\mathsf{Tr}^U_{X,Y}(f)) = \mathsf{Tr}^U_{X,Y'}((g \otimes id_U) \circ f)$ for any $f : X \otimes U \to Y \otimes U$ and $g : Y \to Y'$.
- Dinatural in U: $\mathsf{Tr}^U_{X,Y}((id_Y \otimes g) \circ f) = \mathsf{Tr}^{U'}_{X,Y}(f \circ (id_X \otimes g))$ for any $f : X \otimes U \to Y \otimes U'$ and $g : U' \to U$.

- Vanishing: $\mathsf{Tr}^{I}_{X,Y}(f) = f$ and $\mathsf{Tr}^{U\otimes V}_{X,Y}(g) = \mathsf{Tr}^{U}_{X,Y}(Tr^{V}_{X\otimes U, Y\otimes V}(g))$ for any $f : X \otimes I \to Y \otimes I$ and $g : X \otimes U \otimes V \to Y \otimes U \otimes V$.
- Superposing: $g \otimes \mathsf{Tr}^{U}_{X,Y}(f) = \mathsf{Tr}^{U}_{W\otimes X, Z\otimes Y}(g \otimes f)$ for any $f : X \otimes U \to Y \otimes U$ and $g : W \to Z$.
- Yanking: $\mathsf{Tr}^{U}_{U,U}(c_{U,U}) = id_U$.

A *traced monoidal functor* is a symmetric monoidal functor that satisfies:

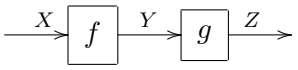
$$\mathsf{Tr}^{FU}_{FX,FY}(\phi^{-1}_{Y,U} \circ (Ff) \circ \phi_{X,U}) = F(\mathsf{Tr}^{U}_{X,Y}(f)) \ .$$

Arrows of a traced symmetric monoidal category can be depicted graphically, without changing or losing any property. More generally, monoidal category and its variants have such graphical representation respectively. A survey on those notations is [20]. In this notation, an arrow is depicted as a box, with in/out ports representing its domain/codomain. Tensor object corresponds to a bundle of ports. For example, an arrow $f : U \otimes V \otimes W \to X \otimes Y$ is described as follows:



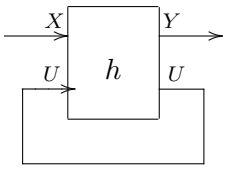
.

Composition is drawn as connection of ports. A composite of $f : X \to Y$ and $g : Y \to Z$ is:



.

And taking a trace can be regarded as taking a feedback. For an arrow $h : X \otimes U \to Y \otimes U$, its trace $\mathsf{Tr}^{U}_{X,Y}(f) : X \to Y$ is depicted as:



3.2 GoI Construction

Abramsky et al. presented a categorical model of Geometry of Interaction in [1].

Definition 3.2.1 (The Geometry of Interaction Construction)**.** Let $\mathbf{C}$ be a traced symmetric monoidal category. A new category $\mathcal{G}(\mathbf{C})$ is constructed as follows:

- Objects: A object of $\mathcal{G}(\mathbf{C})$ is a pair of objects in $\mathbf{C}$.
- Arrows: An arrow $f : (A^+, A^-) \to (B^+, B^-)$ in $\mathcal{G}(\mathbf{C})$ is an arrow $f : A^+ \otimes B^- \to A^- \otimes B^+$ in $\mathcal{G}(\mathbf{C})$, where A^+, A^-, B^+, B^- are objects in $\mathbf{C}$.
- Identity: An identity arrow $id_{(A^+,A^-)}$ is c_{A^+,A^-}, a component of the braiding c.
- Composition: For any $f : (A^+, A^-) \to (B^+, B^-)$ and $g : (B^+, B^-) \to (C^+, C^-)$, their composition $g \circ_{\mathcal{G}} f : (A^+, A^-) \to (C^+, C^-)$ is given by the trace:

$$g \circ_{\mathcal{G}} f := \mathsf{Tr}^{B^-\otimes B^+}_{A^+\otimes C^-, A^-\otimes C^+}\big(\beta \circ (f \otimes g) \circ \alpha\big)$$

where

$\alpha = (id_{A^+} \otimes id_{B^-} \otimes c_{C^-,B^+}) \circ (id_{A^+} \otimes c_{C^-,B^-} \otimes id_{B^+})$,

$\beta = (id_{A^-} \otimes id_{C^+} \otimes c_{B^+,B^-}) \circ (id_{A^-} \otimes c_{B^+,C^+} \otimes id_{B^-}) \circ (id_{A^-} \otimes id_{B^+} \otimes c_{B^-,C^-})$.

- Tensor: For objects, $(A^+, A^-) \otimes_{\mathcal{G}} (B^+, B^-)$ is $(A^+ \otimes B^+, A^- \otimes B^-)$. For arrows $f : (A^+, A^-) \to (B^+, B^-)$ and $g : (C^+, C^-) \to (D^+, D^-)$, their tensor $g \otimes_{\mathcal{G}} f : (A^+, A^-) \otimes_{\mathcal{G}} (C^+, C^-) \to (B^+, B^-) \otimes_{\mathcal{G}} (D^+, D^-)$ is:

$$f \otimes_{\mathcal{G}} g := (id_{A^-} \otimes c_{B^+,C^-} \otimes id_{D^+}) \circ (f \otimes g) \circ (id_{A^+} \otimes c_{C^-,B^-} \otimes id_{B^+}) \ .$$

- Unit: The unit in $\mathcal{G}(\mathbf{C})$ is (I, I) .

There is a special condition on this construction: *GoI situation*.

Notation. If two arrows $f : A \to B$ and $g : B \to A$ satisfy $g \circ f = id_A$, they are denoted by $f : A \lhd B : g$. Such arrows f and g are called a *section* and a *retraction* respectively, and such a object A is called a *retract*. For two objects A and B, if there exist such arrows then they are written as $A \lhd B$. The same notation is used for functors or natural transformations.

Definition 3.2.2 (GoI Situation)**.** A *GoI situation* is a triple $(\mathbf{C}, T, U)$ where:

- $\mathbf{C}$ is a traced symmetric monoidal category.
- $T : \mathbf{C} \to \mathbf{C}$ is a traced symmetric monoidal functor with the following monoidal natural transformations:
  - $e : TT \lhd T : e'$
  - $d : Id \lhd T : d'$
  - $c : T \otimes T \lhd T : c'$
  - $w : I \lhd T : w'$ where I is the constant functor to I.
- U, called a *reflexive object* is an object of $\mathbf{C}$ with retracts:
  - $j : U \otimes U \lhd U : k$
  - $I \lhd U$
  - $u : TU \lhd U : v$.

Given this condition, the collection of arrows $\mathcal{G}(\mathbf{C})(I', V)$ comes to have importance, where $I' = (I, I)$ and $V = (U, U)$. One point is that the set $\mathcal{G}(\mathbf{C})(I', V)$ is equal to $\mathbf{C}(U, U)$:

$$\begin{array}{rcll} \mathcal{G}(\mathbf{C})(I', V) & = & \{f \mid f : (I, I) \to (U, U) \text{ in } \mathcal{G}(\mathbf{C})\} & \\ & = & \{f \mid f : I \otimes U \to I \otimes U \text{ in } \mathbf{C}\} & \text{by definition} \\ & = & \{f \mid f : U \to U \text{ in } \mathbf{C}\} & \text{by Theorem 3.1.6} \\ & = & \mathbf{C}(U, U) \ . & \end{array}$$

The main role of $\mathcal{G}(\mathbf{C})(I', V) = \mathbf{C}(U, U)$ is the following proposition; it forms a linear combinatory algebra:

Proposition 3.2.3. *Given a GoI situation $(\mathbf{C}, T, U)$ with retracts $j : U \otimes U \lhd U : k$ and $u : TU \lhd U : v$, a triple $(\mathcal{G}(\mathbf{C})(I, V), \cdot, !)$ forms a linear combinatory algebra where a binary operation $\cdot$ and unary operation $!$ are defined as:*

$$\begin{array}{rcl} f \cdot g & := & \mathsf{Tr}^U_{U,U}((id_U \otimes g) \circ k \circ f \circ j) \\ !f & := & u(Tf)v \ . \end{array}$$

Linear combinatory algebra is a variant of combinatory algebra that corresponds to linear logic. In next chapter we have a discussion on its definition and properties. As we see later, its properties are convenient to handle. Thus once we can find a GoI situation on a certain category $\mathbf{C}$, this construction provides us a good means to investigate the arrows $\mathbf{C}(U, U)$.

Chapter 4

Linear Combinatory Algebra

4.1 Definitions

First we list definitions about *linear combinatory algebra*.

Definition 4.1.1. $(A, \cdot, !)$ is a !-*applicative structure* if A is a fixed set, $\cdot$ is a binary operation on A, and ! is a unary operation on A.

Definition 4.1.2. Let $(A, \cdot, !)$ be a !-applicative structure. An expression e is a *term* of A if e is a syntactic expression built up using elements of A, variables $x, y, \ldots$, operations $\cdot$ and !. A term including no variables is called a *closed term*. A closed term is also called a *combinator*.

Definition 4.1.3. Let e be a term of A. A variable x is *linear* in e if x occurs in e exactly once, not within any scope of !.

Notation. Let e be a term. Variables and linear variables occurring in e are denoted by:
$V(e) := \{x \mid x \text{ is a variable occurring in } e\}$
$LV(e) := \{x \mid x \text{ is a linear variable occurring in } e\}$
Using this notation, e is called a closed term if and only if $V(e) = \emptyset$. Usually, $x \cdot y$ is simply written as xy. Similarly, $x \cdot (!\, y)$ is written as $x\,!\,y$. Applications of $\cdot$ is usually considered to be left-associative, i.e. $x_1 x_2 \ldots x_n$ means $((\ldots(x_1 x_2)x_3)\ldots)x_n$. $e_1 \equiv e_2$ means e_1 and e_2 are syntactically equal.

Definition 4.1.4. A !-applicative structure $(A, \cdot, !)$ is a *linear combinatory algebra* (sometimes abbreviated as LCA) if A has eight elements $\mathsf{I}, \mathsf{B}, \mathsf{C}, \mathsf{K}, \mathsf{W}, \mathsf{D}, \delta, \mathsf{F}$ that satisfy equations below:

$$\begin{array}{rclrclrcl} \mathsf{I}x & = & x & \mathsf{B}xyz & = & x(yz) & \mathsf{C}xyz & = & xzy \\ \mathsf{K}x\,!\,y & = & x & \mathsf{W}x\,!\,y & = & x\,!\,y\,!\,y & \mathsf{D}\,!\,x & = & x \\ \delta\,!\,x & = & !\,!\,x & \mathsf{F}\,!\,x\,!\,y & = & !(xy)\ . & & & \end{array}$$

If the combinator K satisfies $\mathsf{K}xy = x$, the combinator K is said to be *full* and $(A, \cdot, !)$ is called an *affine combinatory algebra* (ACA).

Definition 4.1.5. Let $(A, \cdot, !)$ be a linear combinatory algebra. Equations on $(A, \cdot, !)$ are defined by axioms for $\mathsf{I}, \mathsf{B}, \mathsf{C}, \mathsf{K}, \mathsf{W}, \mathsf{D}, \delta, \mathsf{F}$ and the following rules:

$$\begin{array}{rcl} e = e & & \\ e_1 = e_2 & \Rightarrow & e_2 = e_1 \\ e_1 = e_2,\ e_2 = e_3 & \Rightarrow & e_1 = e_3 \\ e_1 = e_2 & \Rightarrow & e_1 e_3 = e_2 e_3 \\ e_1 = e_2 & \Rightarrow & e_3 e_1 = e_3 e_2\ . \end{array}$$

4.2 Combinatory Completeness

For ordinary combinatory algebra, the properties stated in the following theorem is known as *combinatory completeness*. It is stated in [3, Theorem 9.3.3] for example.

Theorem 4.2.1 (Combinatory Completeness)**.** *For any term e and any variable x, there exists another term $\lambda^* x.e$ that satisfies:*

$$\begin{array}{rcl} V(\lambda^* x.e) & = & V(e) \setminus \{x\} \\ (\lambda^* x.e)x & = & e \\ (\lambda^* x.e)e' & = & e[x := e'] \ . \end{array}$$

where $V(e)$ is the set of variables occurring in e and $e[x := e']$ is the term obtained by substituting e' to each x in e.

Similar properties hold for linear combinatory algebra. The following theorem is shown in [21] without proof. [1] Here we give an explicit proof for it.

Theorem 4.2.2 (Linear Combinatory Completeness)**.** *For any* !*-applicative structure $(A, \cdot, !)$,* (I) *and* (II) *below are equivalent.*

(I) *$(A, \cdot, !)$ is a linear combinatory algebra.*

(II) *For every term e of A,* (a) *and* (b) *below hold.*

(a) *Let x be a variable. If $x \in LV(e)$ then there exists a term $\lambda^* x.e$ that satisfies*

$$\begin{array}{rcl} LV(\lambda^* x.e) & = & LV(e) \setminus \{x\} \\ V(\lambda^* x.e) & = & V(e) \setminus \{x\} \\ (\lambda^* x.e)x & = & e \ . \end{array}$$

(b) *Let x be a variable. There exists a term $\lambda!^* x.e$ that satisfies*

$$\begin{array}{rcl} LV(\lambda!^* x.e) & = & LV(e) \setminus \{x\} \\ V(\lambda!^* x.e) & = & V(e) \setminus \{x\} \\ (\lambda!^* x.e)\,!\,x & = & e \ . \end{array}$$

Proof. (I) $\Rightarrow$ (II-a) Suppose $x \in LV(e)$. We can construct $\lambda^* x.e$ inductively:

$$\begin{array}{rcll} \lambda^* x.x & \equiv & \mathsf{I} & \\ \lambda^* x.e_1 e_2 & \equiv & \mathsf{C}(\lambda^* x.e_1)e_2 & \text{if } x \in LV(e_1) \\ \lambda^* x.e_1 e_2 & \equiv & \mathsf{B}\, e_1(\lambda^* x.e_2) & \text{if } x \in LV(e_2) \end{array}$$

We do not have to consider the other cases. Indeed, if $e \equiv !\, e'$ then $x \notin LV(e) = \emptyset$. Cases of $e \equiv y$ (y is a variable other than x) or $e \equiv \mathsf{X}$ (X is a combinator) also do not satisfy $x \in LV(e)$. Thus $e \equiv x$ or $e \equiv e_1 e_2$. In the case of $e \equiv e_1 e_2$, if both $x \in LV(e_1)$ and $x \in LV(e_2)$ hold, the situation implies x occurs twice in $e_1 e_2 (\equiv e)$. Thus x is not linear in e, i.e. $x \notin LV(e)$, in this case.

With this construction, each property of $\lambda^* x.e$ follows, shown by structural induction on $\lambda^* x.e$.

1) $LV(\lambda^* x.e) = LV(e) \setminus \{x\}$

If $e \equiv x$,

$$\begin{array}{rcl} LV(\lambda^* x.e) & = & LV(\lambda^* x.x) \\ & = & LV(I) \\ & = & \emptyset \\ & = & \{x\} \setminus \{x\} \\ & = & LV(x) \setminus \{x\} \\ & = & LV(e) \setminus \{x\} \ . \end{array}$$

[1] In [21], a term of A is called a !*-applicative polynomial.*

If $e \equiv e_1 e_2$ and $x \in LV(e_1)$,

$$\begin{aligned} LV(\lambda^* x.e) &= LV(\lambda^* x.e_1 e_2) \\ &= LV(\mathsf{C}(\lambda^* x.e_1)e_2) \\ &= LV(\lambda^* x.e_1) \cup LV(e_2) \\ &= (LV(e_1) \setminus \{x\}) \cup LV(e_2) \text{ (induction hypothesis)} \\ &= (LV(e_1) \setminus \{x\}) \cup (LV(e_2) \setminus \{x\}) \text{ (since } x \notin LV(e_2)) \\ &= (LV(e_1) \cup LV(e_2)) \setminus \{x\} \\ &= LV(e_1 e_2) \setminus \{x\} \\ &= LV(e) \setminus \{x\} \ . \end{aligned}$$

If $e \equiv e_1 e_2$ and $x \in LV(e_2)$, the proof is similar to the above.

2) $V(\lambda^* x.e) = V(e) \setminus \{x\}$

The proof is similar to the case of LV.

3) $(\lambda^* x.e)x = e$

If $e \equiv x$,

$$\begin{aligned} (\lambda^* x.e)x &\equiv (\lambda^* x.x)x \\ &\equiv Ix \\ &= x \\ &\equiv e \ . \end{aligned}$$

If $e \equiv e_1 e_2$ and $x \in LV(e_1)$,

$$\begin{aligned} (\lambda^* x.e)x &\equiv (\lambda^* x.e_1 e_2)x \\ &\equiv \mathsf{C}(\lambda^* x.e_1)e_2 x \\ &= (\lambda^* x.e_1)xe_2 \\ &= e_1 e_2 \text{ (induction hypothesis)} \\ &\equiv e \ . \end{aligned}$$

If $e \equiv e_1 e_2$ and $x \in LV(e_2)$,

$$\begin{aligned} (\lambda^* x.e)x &\equiv (\lambda^* x.e_1 e_2)x \\ &\equiv \mathsf{B}e_1(\lambda^* x.e_2)x \\ &= e_1((\lambda^* x.e_2)x) \\ &= e_1 e_2 \text{ (induction hypothesis)} \\ &\equiv e \ . \end{aligned}$$

(I) $\Rightarrow$ (Ⅱ-b) First we define an auxiliary combinator S by

$$\mathsf{S} \equiv \mathsf{B}\big(\mathsf{B}(\mathsf{BW})\mathsf{B}\big)\mathsf{C} \ . \tag{4.1}$$

This combinator satisfies $\mathsf{S}xy\,!\,z = x\,!\,z(y\,!\,z)$ for all x, y, and z. Then we can construct $\lambda!^* x.e$ inductively:

$$\begin{aligned} \lambda!^* x.e &\equiv \mathsf{K}e \quad \text{if } x \notin V(e) \\ \lambda!^* x.x &\equiv \mathsf{D} \\ \lambda!^* x.\,!\,x &\equiv \mathsf{I} \\ \lambda!^* x.\,!\,!\,e' &\equiv \mathsf{B}\,\delta(\lambda!^* x.\,!\,e') \\ \lambda!^* x.e_1 e_2 &\equiv \mathsf{S}(\lambda!^* x.e_1)(\lambda!^* x.e_2) \\ \lambda!^* x.\,!(e_1 e_2) &\equiv \mathsf{S}\big(\mathsf{BF}(\lambda!^* x.\,!\,e_1)\big)(\lambda!^* x.\,!\,e_2) \end{aligned}$$

The proof for each property is similar to that of $\lambda^* x.e$.

1) $LV(\lambda!^* x.e) = LV(e) \setminus \{x\}$

If $x \notin V(e)$,

$$\begin{aligned} LV(\lambda!^*x.e) &= LV(\mathsf{K}e) \\ &= LV(e) \\ &= LV(e) \setminus \{x\} \text{ (since } x \notin V(e)) \ . \end{aligned}$$

If $e \equiv x$,

$$\begin{aligned} LV(\lambda!^*x.e) &= LV(\lambda!^*x.x) \\ &= LV(\mathsf{D}) \\ &= \emptyset \\ &= \{x\} \setminus \{x\} \\ &= LV(e) \setminus \{x\} \ . \end{aligned}$$

If $e \equiv !\,x$,

$$\begin{aligned} LV(\lambda!^*x.e) &= LV(\lambda!^*x.!\,x) \\ &= \emptyset \\ &= \emptyset \setminus \{x\} \\ &= LV(!\,x) \setminus \{x\} \\ &= LV(e) \setminus \{x\} \ . \end{aligned}$$

If $e \equiv !\,!\,e'$,

$$\begin{aligned} LV(\lambda!^*x.e) &= LV(\lambda!^*x.!\,!\,e') \\ &= \emptyset \\ &= \emptyset \setminus \{x\} \\ &= LV(!\,!\,e') \setminus \{x\} \\ &= LV(e) \setminus \{x\} \ . \end{aligned}$$

If $e \equiv e_1 e_2$,

$$\begin{aligned} LV(\lambda!^*x.e) &= LV(\lambda!^*x.e_1e_2) \\ &= LV\big(\mathsf{S}(\lambda!^*x.e_1)(\lambda!^*x.e_2)\big) \\ &= LV(\lambda!^*x.e_1) \cup LV(\lambda!^*x.e_2) \\ &= (LV(e_1) \setminus \{x\}) \cup (LV(e_2) \setminus \{x\}) \text{ (induction hypothesis)} \\ &= \big(LV(e_1) \cup LV(e_2)\big) \setminus \{x\} \\ &= LV(e_1e_2) \setminus \{x\} \\ &= LV(e) \setminus \{x\} \ . \end{aligned}$$

If $e \equiv !(e_1e_2)$,

$$\begin{aligned} LV(\lambda!^*x.e) &= LV\big(\lambda!^*x.!(e_1e_2)\big) \\ &= LV\big(\mathsf{S}(\mathsf{BF}(\lambda!^*x.!\,e_1))(\lambda!^*x.!\,e_2)\big) \\ &= LV(\lambda!^*x.!\,e_1) \cup LV(\lambda!^*x.!\,e_2) \\ &= \big(LV(!\,e_1) \setminus \{x\}\big) \cup \big(LV(!\,e_2) \setminus \{x\}\big) \text{ (induction hypothesis)} \\ &= \big(\emptyset \setminus \{x\}\big) \cup \big(\emptyset \setminus \{x\}\big) \\ &= \emptyset \setminus \{x\} \\ &= LV(!(e_1e_2)) \setminus \{x\} \\ &= LV(e) \setminus \{x\} \ . \end{aligned}$$

2) $V(\lambda!^*x.e) = V(e) \setminus \{x\}$
The proof is similar to the case of LV.

3) $(\lambda!^*x.e)\,!\,x = e$
If $x \notin V(e)$,

$$\begin{aligned} (\lambda!^*x.e)\,!\,x &\equiv (\mathsf{K}e)\,!\,x \\ &= e \ . \end{aligned}$$

If $e \equiv x$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,x & \equiv & (\lambda!^*x.x)\,!\,x \\ & \equiv & \mathsf{D}\,!\,x \\ & = & x \\ & \equiv & e\ .\end{array}$$

If $e \equiv !\,x$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,x & \equiv & (\lambda!^*x.!\,x)\,!\,x \\ & \equiv & \mathsf{I}\,!\,x \\ & = & !\,x \\ & \equiv & e\ .\end{array}$$

If $e \equiv !\,!\,e'$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,x & \equiv & (\lambda!^*x.!\,!\,e')\,!\,x \\ & \equiv & \mathsf{B}\delta(\lambda!^*x.!\,e')\,!\,x \\ & = & \delta((\lambda!^*x.!\,e')\,!\,x) \\ & = & \delta\,!\,e' \text{ (induction hypothesis)} \\ & = & !\,!\,e' \\ & \equiv & e\ .\end{array}$$

If $e \equiv e_1e_2$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,x & \equiv & (\lambda!^*x.e_1e_2)\,!\,x \\ & \equiv & \mathsf{S}(\lambda!^*x.e_1)(\lambda!^*x.e_2)\,!\,x \\ & = & \big((\lambda!^*x.e_1)\,!\,x\big)\big((\lambda!^*x.e_2)\,!\,x\big) \\ & = & e_1e_2 \text{ (induction hypothesis)} \\ & \equiv & e\ .\end{array}$$

If $e \equiv !(e_1e_2)$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,x & \equiv & \big(\lambda!^*x.!(e_1e_2)\big)\,!\,x \\ & \equiv & \mathsf{S}\big(\mathsf{BF}(\lambda!^*x.!\,e_1)\big)(\lambda!^*x.!\,e_2)\,!\,x \\ & = & \mathsf{BF}(\lambda!^*x.!\,e_1)\,!\,x\big((\lambda!^*x.!\,e_2)\,!\,x\big) \\ & = & \mathsf{F}\big((\lambda!^*x.!\,e_1)\,!\,x\big)\big((\lambda!^*x.!\,e_2)\,!\,x\big) \\ & = & \mathsf{F}\,!\,e_1\,!\,e_2 \text{ (induction hypothesis)} \\ & = & !(e_1e_2) \\ & \equiv & e\ .\end{array}$$

(I) $\Leftarrow$ (II) For any term e and variable x, suppose (a) and (b) hold:
(a) If $x \in LV(e)$, there exists a term $\lambda^*x.e$ that satisfies

$$\begin{array}{rcll}LV(\lambda^*x.e) & = & LV(e) \setminus \{x\} & \text{(a-i)} \\ V(\lambda^*x.e) & = & V(e) \setminus \{x\} & \text{(a-ii)} \\ (\lambda^*x.e)x & = & e & \text{(a-iii)}\end{array}$$

(b) There exists a term $\lambda!^*x.e$ that satisfies

$$\begin{array}{rcll}LV(\lambda!^*x.e) & = & LV(e) \setminus \{x\} & \text{(b-i)} \\ V(\lambda!^*x.e) & = & V(e) \setminus \{x\} & \text{(b-ii)} \\ (\lambda!^*x.e)\,!\,x & = & e & \text{(b-iii)}\ .\end{array}$$

Take eight terms $I, B, C, K, W, D, \delta, F$ as

$$\begin{array}{ll}I \equiv \lambda^*x.x & W \equiv \lambda^*x.(\lambda!^*y.\,x\,!\,y\,!\,y) \\ B \equiv \lambda^*xyz.x(yz) & D \equiv \lambda!^*x.x \\ C \equiv \lambda^*xyz.\,xzy & \delta \equiv \lambda!^*x.!\,!\,x \\ K \equiv \lambda^*x.(\lambda!^*y.x) & F \equiv \lambda!^*xy.!(xy)\ .\end{array}$$

By assumptions (a-ii) and (b-ii), each of the terms has no variable. Thus the terms are indeed combinators of A. By assumptions (a-iii) and (b-iii), these combinators satisfy equations

$$\begin{array}{rclcrcl} Ix & = & x & & Wx\,!\,y & = & x\,!\,y\,!\,y \\ Bxyz & = & x(yz) & & D\,!\,x & = & x \\ Cxyz & = & xzy & & \delta\,!\,x & = & !\,!\,x \\ Kx\,!\,y & = & x & & F\,!\,x\,!\,y & = & !(xy)\ . \end{array}$$

Hence $(A, \cdot, !)$ is a linear combinatory algebra. □

The λ-abstraction-like terms in Theorem 4.2.2 also satisfy another property corresponding to the third one in Theorem 4.2.1. To verify this, we define substitutions on LCA and show a corollary of the theorem.

Definition 4.2.3. Let $(A, \cdot, !)$ be an LCA , e and e' be terms of A, and x be a variable. A term $e[x := e']$ is inductively defined as follows:

$$\begin{array}{rcl} e[x := e'] & \equiv & e \ \text{ if } x \notin V(e) \\ x[x := e'] & \equiv & e' \\ (!\,e)[x := e'] & \equiv & !(e[x := e']) \\ (e_1e_2)[x := e'] & \equiv & (e_1[x := e'])(e_2[x := e']) \end{array}$$

Corollary 4.2.4. *For the terms $\lambda^*x.e$ and $\lambda!^*x.e$ in Theorem 4.2.2, the following equations hold:*

$$\begin{array}{rcl} (\lambda^*x.e)e' & = & e[x := e'] \\ (\lambda!^*x.e)\,!\,e' & = & e[x := e']\ . \end{array}$$

Proof. 1) $(\lambda^*x.e)e' = e[x := e']$

If $e \equiv x$,

$$\begin{array}{rcl} (\lambda^*x.e)e' & \equiv & (\lambda^*x.x)e' \\ & \equiv & \mathsf{I}e' \\ & = & e' \\ & \equiv & x[x := e'] \\ & \equiv & e[x := e']\ . \end{array}$$

If $e \equiv e_1e_2$ and $x \in LV(e_1)$,

$$\begin{array}{rcl} (\lambda^*x.e)e' & \equiv & (\lambda^*x.e_1e_2)e' \\ & \equiv & \mathsf{C}(\lambda^*x.e_1)e_2e' \\ & = & (\lambda^*x.e_1)e'e_2 \\ & = & (e_1[x := e'])e_2 \text{ (induction hypothesis)} \\ & \equiv & (e_1[x := e'])(e_2[x := e']) \text{ (since } x \notin LV(e_2)) \\ & \equiv & (e_1e_2)[x := e'] \\ & \equiv & e[x := e']\ . \end{array}$$

If $e \equiv e_1e_2$ and $x \in LV(e_2)$, the proof is similar to the above.

2) $(\lambda!^*x.e)\,!\,e' = e[x := e']$

If $x \notin V(e)$,

$$\begin{array}{rcl} (\lambda!^*x.e)\,!\,e' & \equiv & \mathsf{K}e\,!\,e' \\ & = & e \\ & \equiv & e[x := e'] \quad (\text{since } x \notin V(e))\ . \end{array}$$

If $e \equiv x$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,e' & \equiv & (\lambda!^*x.x)\,!\,e' \\ & \equiv & \mathsf{D}\,!\,e' \\ & = & e' \\ & \equiv & x[x := e'] \\ & \equiv & e[x := e'] \ .\end{array}$$

If $e \equiv !\,x$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,e' & \equiv & (\lambda!^*x.!\,x)\,!\,e' \\ & \equiv & \mathsf{I}\,!\,e' \\ & = & !\,e' \\ & \equiv & (!\,x)[x := e'] \\ & \equiv & e[x := e'] \ .\end{array}$$

If $e \equiv !\,!\,e_1$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,e' & \equiv & (\lambda!^*x.!\,!\,e_1)\,!\,e' \\ & \equiv & \mathsf{B}\delta(\lambda!^*x.!\,e_1)\,!\,e' \\ & = & \delta((\lambda!^*x.!\,e_1)\,!\,e') \\ & = & \delta((!\,e_1)[x := e']) \text{ (induction hypothesis)} \\ & \equiv & \delta\,!(e_1[x := e']) \\ & = & !\,!(e_1[x := e']) \\ & \equiv & (!\,!\,e_1)[x := e'] \\ & \equiv & e[x := e'] \ .\end{array}$$

If $e \equiv e_1e_2$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,e' & \equiv & (\lambda!^*x.e_1e_2)\,!\,e' \\ & \equiv & \mathsf{S}(\lambda!^*x.e_1)(\lambda!^*x.e_2)\,!\,e' \\ & = & (\lambda!^*x.e_1)\,!\,e'((\lambda!^*x.e_2)\,!\,e') \\ & = & (e_1[x := e'])(e_2[x := e']) \text{ (induction hypothesis)} \\ & \equiv & (e_1e_2)[x := e'] \\ & \equiv & e[x := e'] \ .\end{array}$$

If $e \equiv !(e_1e_2)$,

$$\begin{array}{rcl}(\lambda!^*x.e)\,!\,e' & \equiv & (\lambda!^*x.!(e_1e_2))\,!\,e' \\ & \equiv & \mathsf{S}\big(\mathsf{BF}(\lambda!^*x.!\,e_1)\big)(\lambda!^*x.!\,e_2)\,!\,e' \\ & = & \mathsf{BF}(\lambda!^*x.!\,e_1)\,!\,e'((\lambda!^*x.!\,e_2)\,!\,e') \\ & = & \mathsf{F}((\lambda!^*x.!\,e_1)\,!\,e')((\lambda!^*x.!\,e_2)\,!\,e') \\ & = & \mathsf{F}((!\,e_1)[x := e'])((!\,e_2)[x := e']) \text{ (induction hypothesis)} \\ & \equiv & \mathsf{F}\,!(e_1[x := e'])\,!(e_2[x := e']) \\ & = & !\,\big((e_1[x := e'])(e_2[x := e'])\big) \\ & \equiv & !\,\big((e_1e_2)[x := e']\big) \\ & \equiv & (!(e_1e_2))[x := e'] \\ & \equiv & e[x := e'] \ .\end{array}$$

□

The theorem and corollary above allow us to handle linear combinatory terms as if they are closed linear λ-calculus terms. When we define a linear combinatory term in such a λ-abstraction-like form, it satisfies the properties in the theorem, without necessity of writing down its explicit form. We will take advantage of this fact in the next chapter.

Chapter 5

Partial Equivalence Relations on LCA

5.1 Definitions

We follow the standard construction of the category of *partial equivalence relations* (abbreviated as *PER*) on an LCA A, denoted by $\mathbf{PER}_A$. It appears as a category that models a computation with branching, via GoI construction. Thus a study on $\mathbf{PER}_A$ plays an important role when we study computations that can be represented as such models.

Definition 5.1.1 (Partial Equivalence Relation)**.** A *partial equivalence relation* R on a set X is a set of pairs of elements of X, i.e. $R \subseteq X \times X$, which satisfies following conditions:

$$\begin{array}{lll} \forall x, y \in X. & (x,y) \in R \Rightarrow (y,x) \in R & \text{(symmetry)} \\ \forall x, y, z \in X. & (x,y) \in R \wedge (y,z) \in R \Rightarrow (x,z) \in R & \text{(transitivity)} \end{array}$$

Much like equivalence classes of (ordinary) equivalence relations, an *equivalence class* of an element $x \in X$ is defined as $\{x' \,|\, (x,x') \in R\}$. It is denoted by $[x]$.
A *domain* of a PER R, denoted by $|R|$, is defined by $\{x \,|\, \exists y.(x,y) \in R\}$. The domain R is equal to $\{x \,|\, (x,x) \in R\}$, since for an arbitrary $x \in X$ if there exists $y \in X$ such that $(x,y) \in R$, $(y,x) \in R$ holds by symmetry, and then $(x,x) \in R$ holds by transitivity from $(x,y) \in R$ and $(y,x) \in R$.

Note that it is not the same notion as equivalence relations. A simple example is as follows:

Example. Let X and Y be sets, $f : X \rightharpoonup Y$ be a partial function. Define a relation R on X by

$R := \{(x,x') \in X \times X \,|\, \text{both } f(x) \text{ and } f(x') \text{ are defined} \wedge\ f(x) = f(x')\}$.

Then symmetry and transitivity clearly hold, but $(x,x) \in R$ is not guaranteed for every x since there can be an element x' such that $f(x')$ is undefined. Thus R is not an equivalence relation but a partial equivalence relation.

Definition 5.1.2 (Category $\mathbf{PER}_A$)**.** Let A be an LCA. The category $\mathbf{PER}_A$ consists of:

- An object of $\mathbf{PER}_A$ is a PER on A.
- An arrow $X \to Y$ of $\mathbf{PER}_A$ is an equivalence class of a PER $X \multimap Y$, where

$$\begin{array}{lcl} |X \multimap Y| & := & \{c \in A \,|\, (x,x') \in X \Rightarrow (cx, cx') \in Y\} \\ X \multimap Y & := & \left\{(c,c') \in |X \multimap Y| \times |X \multimap Y| \;\middle|\; \forall x \in |X|.\, (cx, c'x) \in Y\right\} \end{array}$$

The PER $X \multimap Y$ is the same as $\{(c, c') \in A \times A \mid (x, x') \in X \Rightarrow (cx, c'x') \in Y\}$. For an equivalence class $[c]$ with its representative c, the arrow $[c]$ is said to be *realized by the code* c.([14]) The identity arrow id_X is $[\mathsf{I}]$. For arrows $[f] : X \to Y$ and $[g] : Y \to Z$, the composition $[g] \circ [f] : X \to Z$ is $[\mathsf{B}gf]$.

From the computational perspective, a PER X can be regarded as a "data type." For each (x, x') if $(x, x') \in X$ they represent "the same datum." A term $c \in |X \multimap Y|$ also can be seen as a "function from type X to type Y." Then the condition $(x, x') \in X \Rightarrow (cx, cx') \in Y$ implies "for the same input, the function c returns the same output,". If $(c, c') \in X \multimap Y$, the condition $\forall x \in |X|.\, (cx, c'x) \in Y$ means "for every input x, functions c and c' return the same output." The fact that they are different closed terms of an LCA corresponds to that they are different as "actual programs," e.g. a bubble sort program and quick sort one, and the equivalence class $[c]$ expresses the equivalence of them in more abstract level (both of them do sort data). From this viewpoint, the phrase "the code c realizes the arrow $[c]$" can be understood more clearly.

5.2 Structures of $\mathbf{PER}_A$

Let A be an affine combinatory algebra. In this section we show some properties of $\mathbf{PER}_A$. From Section 5.2.1 to 5.2.3 are already shown in [14]. Here we follow them in a more explicit and detailed way.

5.2.1 Binary Coproducts

The category $\mathbf{PER}_A$ has binary coproducts. To construct them, we prepare two combinators in A.

Definition 5.2.1. The *pairing combinator* P and *second weakening combinator* $\bar{\mathsf{K}}$ are defined as follows:

$$\begin{aligned} \mathsf{P} &:= \lambda^* xyz.zxy \\ \bar{\mathsf{K}} &:= \mathsf{KI} \end{aligned}$$

They satisfy $\mathsf{P}xyz = zxy$ and $\bar{\mathsf{K}}xy = y$ for all x, y, z.

Let X, Y be arbitrary PERs. Their binary coproduct $X + Y$ can be defined by:

$$X + Y := \{(\mathsf{PK}x, \mathsf{PK}x') \mid (x, x') \in X\} \cup \{(\mathsf{P}\bar{\mathsf{K}}y, \mathsf{P}\bar{\mathsf{K}}y') \mid (y, y') \in Y\} \ .$$

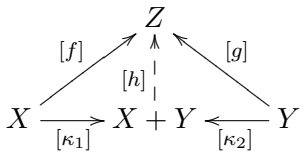
Coprojections $[\kappa_1] : X \to X + Y$ and $[\kappa_2] : Y \to X + Y$ are as follows:

$$\begin{aligned} \kappa_1 &:= \mathsf{PK} \\ \kappa_2 &:= \mathsf{P}\bar{\mathsf{K}} \ . \end{aligned}$$

Take a PER Z and arrows $[f] : X \to Z$, $[g] : Y \to Z$, where $f, g \in A$. We define the mediating arrow $[h] : X + Y \to Z$ by:

$$h := \lambda^* w.w(\lambda^* uv.ufgv) \ .$$

Their required commutativity can be checked as follows:



$$\begin{array}{rcl} h(\kappa_1 x) & \equiv & (\lambda^* w.w(\lambda^* uv.ufgv))(\mathsf{PK}x) \\ & = & \mathsf{PK}x(\lambda^* uv.ufgv) \\ & = & (\lambda^* uv.ufgv)\mathsf{K}x \\ & = & \mathsf{K}fgx \\ & = & fx \end{array}$$

By definition $f \in |X \multimap Z|$ satisfies $(fx, fx) \in Z$ for all $x \in |X|$. Thus $(h(\mathsf{PK}x), fx) = (fx, fx) \in Z$ holds, which means $(\mathsf{B}h(\mathsf{PK}), f) \in X \multimap Z = \{(c, c') \,|\, \forall x \in |X|.\, (cx, c'x) \in Z\}$. Therefore $[h] \circ [\mathsf{PK}] = [\mathsf{B}h(\mathsf{PK})] = [f]$.
The other commutation $[h] \circ [\mathsf{P}\bar{\mathsf{K}}] = [g]$ can be checked similarly.

Next we verify the uniqueness of $[h]$. Assume there exists another arrow $[h'] : X + Y \to Z$ satisfying required commutativity. Then the following equations hold:

$$\begin{array}{l} [h'] \circ [\kappa_1] = [f] = [h] \circ [\kappa_1] \\ [h'] \circ [\kappa_2] = [g] = [h] \circ [\kappa_2] \ . \end{array}$$

Thus $(h(\mathsf{PK}x), h'(\mathsf{PK}x)) \in Z$ for all $x \in |X|$ and $(h(\mathsf{P}\bar{\mathsf{K}}y), h'(\mathsf{P}\bar{\mathsf{K}}y)) \in Z$ for all $y \in |Y|$. Considering the definition of $X + Y$, it implies

$$(h, h') \in X + Y \multimap Z = \{(c, c') \,|\, \forall w \in |X + Y|.\, (cz, c'z) \in Z\} \ .$$

Hence $[h] = [h']$, i.e. $[h]$ is unique.

5.2.2 Binary Products

The category $\mathbf{PER}_A$ has binary products. Below the combinators P and $\bar{\mathsf{K}}$ are the same one as defined in Section 5.2.1. Let X, Y be arbitrary PERs. Their binary product $X \times Y$ can be defined as:

$$X \times Y := \{(\mathsf{P}k_1(\mathsf{P}k_2 u),\, \mathsf{P}k_1'(\mathsf{P}k_2' u')) \,|\, (k_1 u, k_1' u') \in X \wedge (k_2 u, k_2' u') \in Y\} \ .$$

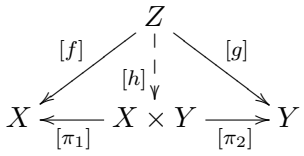
Projections $[\pi_1] : X \times Y \to X$ and $[\pi_2] : X \times Y \to Y$ are defined by using combinatory completeness:

$$\begin{array}{l} \pi_1 := \lambda^* l.\, l\big(\lambda^* mn.\, m(n\bar{\mathsf{K}})\big) \\ \pi_2 := \lambda^* l.\, l\big(\lambda^* mn.\, n(\bar{\mathsf{K}}m)\big) \ . \end{array}$$

Take a PER Z and arrows $[f] : Z \to X$, $[g] : Z \to Y$, with $f, g \in A$. We define the mediating arrow $[h] : Z \to X \times Y$ as:

$$h := \mathsf{B}(\mathsf{P}f)(\mathsf{P}g) \ .$$

Their commutativity can be checked as follows:



$$\begin{array}{rcl} \pi_1(hz) & \equiv & \big(\lambda^* l.\, l(\lambda^* mn.\, m(n\bar{\mathsf{K}}))\big)\big(\mathsf{B}(\mathsf{P}f)(\mathsf{P}g)z\big) \\ & = & \mathsf{B}(\mathsf{P}f)(\mathsf{P}g)z\big(\lambda^* mn.\, m(n\bar{\mathsf{K}})\big) \\ & = & \mathsf{P}f\,(\mathsf{P}gz)\big(\lambda^* mn.\, m(n\bar{\mathsf{K}})\big) \\ & = & \big(\lambda^* mn.\, m(n\bar{\mathsf{K}})\big) f\big(\mathsf{P}gz\big) \\ & = & f(\mathsf{P}gz\bar{\mathsf{K}}) \\ & = & f(\bar{\mathsf{K}}gz) \\ & = & fz \end{array}$$

Thus $[\pi_1] \circ [h] = [f]$. The other equation $[\pi_2] \circ [h] = [g]$ can be shown similarly.

Below we check the uniqueness of $[h]$. Suppose there exists another arrow $[h'] : Z \to X \times Y$ satisfying the required conditions, with $h' \in A$. Take an arbitrary element $z \in |Z|$. Since $[h']$ is an arrow from Z to $X \times Y$,

$$\begin{array}{rcl} (h'z, h'z) & \in & X \times Y \\ & = & \{(\mathsf{P}k_1(\mathsf{P}k_2 u), \mathsf{P}k_1'(\mathsf{P}k_2' u')) \,|\, (k_1 u, k_1' u') \in X \wedge (k_2 u, k_2' u') \in Y\} \ . \end{array}$$

Thus there exist three terms $l_1, l_2, v \in A$ that satisfy $l_1 v \in |X|$, $l_2 v \in |Y|$, and $h'z = \mathsf{P}l_1(\mathsf{P}l_2 v)$. For such l_1, l_2, v, we have $\pi_1(h'z) = l_1 v$ and $\pi_2(h'z) = l_2 v$. By assumptions $[\pi_1] \circ [h'] = [f]$ and $[\pi_2] \circ [h'] = [g]$, we also have $(\pi_1(h'z), fz) \in X$ and $(\pi_2(h'z), gz) \in Y$, i.e. $(l_1 v, fz) \in X$ and $(l_2 v, gz) \in Y$. Therefore $(h'z, hz) = (\mathsf{P}l_1(\mathsf{P}l_2 v), \mathsf{P}f(\mathsf{P}gz))$ satisfies $(l_1 v, fz) \in X$ and $(l_2 v, gz) \in Y$, which means that $(h'z, hz) \in X \times Y$.
Hence $(h', h) \in Z \multimap X \times Y = \{(c, c') \,|\, \forall z \in |Z|.(cz, c'z) \in X \times Y\}$, that means $[h'] = [h]$. Thus $[h]$ is unique.

5.2.3 Terminal Object

The category $\mathbf{PER}_A$ has the terminal object $1 := \{(\mathsf{I}, \mathsf{I})\}$. For any PER X, the unique arrow $X \to 1$ is $[\mathsf{KI}]$.
Any PER $\{(a, a)\}$ is also a terminal object, i.e. isomorphic to 1.

5.2.4 Initial Object

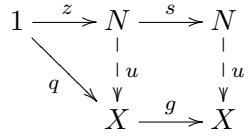
The category $\mathbf{PER}_A$ has the initial object $0 := \emptyset$, which is the empty PER. For any PER X, the unique arrow $0 \to X$ is the entire A.
There does not exist any other initial object in $\mathbf{PER}_A$. For any nonempty PER O, take a PER $X = \{(a, a), (b, b)\}$ where $a \neq b$. Such a, b give rise to arrows $[\mathsf{K}a], [\mathsf{K}b] : O \to X$. For an arbitrary element $o \in |O|$, $(\mathsf{K}ao, \mathsf{K}bo) = (a, b) \notin X$. Thus $[\mathsf{K}a] \neq [\mathsf{K}b]$, which means arrows from O to X are not unique. Therefore the uniqueness of arrow only holds for $\emptyset$.

5.2.5 Natural Number Object

A *natural number object* is an object that has a certain recursive structure on it. The formal definition is:

Definition 5.2.2 (Natural Number Object)**.** Let $\mathbf{C}$ be a category with a terminal object 1. A *natural number object* of $\mathbf{C}$ is an object N with two arrows $z : 1 \to N$ and $s : N \to N$ that satisfy:

> for all object X, arrows $q : 1 \to X$ and $g : X \to X$, there uniquely exists an arrow $u : N \to X$ that makes following diagram commute.



We construct a natural number object in $\mathbf{PER}_A$ as follows:

Proposition 5.2.3. *Let f and x be distinct variables.* $\mathbf{PER}_A$ *has a natural number object N with arrows $[z] : 1 \to N$ and $[s] : N \to N$:*

$$\begin{aligned} N &:= \{\big(\lambda!^*f.(\lambda x.f^m x),\ \lambda!^*f.(\lambda x.f^m x)\big) \mid m \in \mathbb{N}\} \\ z &\equiv \mathsf{K} \\ s &\equiv \mathsf{S}(\mathsf{S}(\mathsf{K}\mathsf{B})\mathsf{D}) \end{aligned}$$

where S *is the same combinator as (4.1),* $f^0 x \equiv x$, $f^{m+1}x \equiv f(f^m x)$.

Proof. By using the linear combinatory completeness (Theorem 4.2.2), we obtain Church numerals-like terms in this construction. By Theorem 4.2.2, the term $\lambda^*!f.(\lambda^* x.f^n x)$ is indeed a closed term of A, thus N is an object of $\mathbf{PER}_A$. The linear combinatory terms s and z satisfy

$$\begin{aligned} z\mathsf{I} &= \mathsf{K}\mathsf{I} = \lambda!^*f.(\lambda^* x.x) \\ s(\lambda!^*f.(\lambda x.f^n x)) &= \lambda!^*f.(\lambda x.f^{n+1}x)\ . \end{aligned}$$

Thus $(z\mathsf{I},\, z\mathsf{I}) \in N$ for $(\mathsf{I},\, \mathsf{I}) \in 1$ and $(sn,\, sn) \in N$ for all $(n,\, n) \in N$, i.e. $[z] : 1 \to N$ and $[s] : N \to N$ are arrows of $\mathbf{PER}_A$.

Let X be an object of $\mathbf{PER}_A$, $[q] : 1 \to X$ and $[g] : 1 \to X$ be arrows of $\mathbf{PER}_A$, with q and g terms of A. We can take an arrow $[u] : N \to X$ with

$$u := \lambda^* n.n\,!\,g\,(q\mathsf{I})\ .$$

First we check that $[u]$ is actually an arrow $N \to X$. For an arbitrary partial function $\lambda!^*f.(\lambda x.f^m x) \in |N|$,

$$\begin{aligned} u(\lambda!^*f.(\lambda x.f^m x)) &= (\lambda!^*f.(\lambda x.f^m x))\,!\,g\,(q\mathsf{I}) \\ &= g^m(q\mathsf{I})\ . \end{aligned}$$

By definition of arrows, $g^0(q\mathsf{I}) = q\mathsf{I} \in |X|$ and $g^{m+1}(q\mathsf{I}) = g(g^m(q\mathsf{I})) \in |X|$ (using mathematical induction). Thus $um \in |X|$ holds for every $m \in |N|$. Since all elements of N is in the form (m, m), the partial function u satisfies the condition $(m, m) \in N \Rightarrow (um, um) \in X$. Therefore $[u]$ is an arrow $N \to X$.
Then required commutativity $[u] \circ [z] = [q]$ and $[u] \circ [s] = [g] \circ [u]$ can be checked as follows:

$$\begin{aligned} u(z\mathsf{I}) &\equiv u(\mathsf{K}\mathsf{I}) \\ &= (\mathsf{K}\mathsf{I})\,!\,g\,(q\mathsf{I}) \\ &= \mathsf{I}(q\mathsf{I}) \\ &= q\mathsf{I} \end{aligned}$$

$$\begin{aligned} u\,(s(\lambda!^*f.(\lambda x.f^m x))) &= u(\lambda!^*f.(\lambda x.f^{m+1}x)) \\ &= (\lambda!^*f.(\lambda x.f^{m+1}x))\,!\,g\,(q\mathsf{I}) \\ &= g^{m+1}(q\mathsf{I}) \\ &\equiv g(g^m(q\mathsf{I})) \\ &= g\big(\lambda!^*f.(\lambda x.f^m x)\,!\,g\,(q\mathsf{I})\big) \\ &= g\Big(\big(\lambda^* n.n\,!\,g\,(q\mathsf{I})\big)\big(\lambda!^*f.(\lambda x.f^m x)\big)\Big) \\ &\equiv g\big(u(\lambda!^*f.(\lambda x.f^m x))\big)\ . \end{aligned}$$

Finally we verify the uniqueness of $[u]$. Suppose that there exists another arrow $[u'] : N \to X$ that satisfies the commutativity. Then for all $m \in \mathbb{N}$ the following equation holds:

$$\begin{aligned} [u'] \circ [s]^m \circ [z] &= [g]^m \circ [u'] \circ [z] \\ &= [g]^m \circ [q] \\ &= [g]^m \circ [u] \circ [z] \\ &= [u] \circ [s]^m \circ [z]\ . \end{aligned}$$

By definition,

$$\begin{array}{rl} & [u'] \circ [s]^m \circ [z] = [u] \circ [s]^m \circ [z] \\ \iff & \Big(u'\big(s(s(...(z\mathsf{I}))...)\big),\, u\big(s(s(...(z\mathsf{I}))...)\big)\Big) \in X \\ \iff & \big(u'(\lambda!^* f.(\lambda x.f^m x)),\, u(\lambda!^* f.(\lambda x.f^m x))\big) \in X \ . \end{array}$$

Thus $(u', u) \in N \multimap X = \{(c, c') \,|\, (n, n) \in N \Rightarrow\ (cn, c'n) \in X\}$, which means u' and u belong to the same equivalence class, i.e. $[u'] = [u]$. Thus $[u]$ is unique. $\square$

The NNO N is similar to an NNO in $\mathbf{PER}_{\mathbb{N}}$ given in [18], which is the category of PERs on natural numbers $\mathbb{N}$.
Our definition $s \equiv \mathsf{S}(\mathsf{S}(\mathsf{K}\mathsf{B})\mathsf{D})$ is not the same as $s' := \lambda^* n.(\lambda!^* f.(\lambda^* x.f(n\ !\, f\, x)))$, which is a linear version of the successor in usual Church encoding. We found it by the following observation:

$$\begin{array}{rcl} \lambda!^* f.(\lambda x.f^{m+1}x) & \equiv & \lambda!^* f.(\lambda^* x.f(f^m x)) \\ & \equiv & \lambda!^* f.(\mathsf{B}f(\lambda^* x.f^m x)) \\ & \equiv & \mathsf{S}(\lambda!^* f.\mathsf{B}f)(\lambda!^* f.(\lambda^* x.f^m x)) \\ & \equiv & \mathsf{S}\big(\mathsf{S}(\lambda!^* f.\mathsf{B})(\lambda!^* f.f)\big)(\lambda!^* f.(\lambda^* x.f^m x)) \\ & \equiv & \mathsf{S}\big(\mathsf{S}(\mathsf{K}\mathsf{B})\mathsf{D}\big)(\lambda!^* f.(\lambda^* x.f^m x)) \end{array}$$

The combinator s' also suffices as the code of the successor arrow of NNO, i.e. $s'(\lambda!^* f.(\lambda x.f^m x)) = \lambda!^* f.(\lambda x.f^{m+1}x)$, but s is much shorter than it; in a concrete form of an LCA term,

$$s' \equiv \mathsf{B}\Big(\mathsf{S}\big(\mathsf{S}(\mathsf{K}\mathsf{B})\mathsf{D}\big)\Big)\Big(\mathsf{C}\Big(\mathsf{B}\mathsf{S}\Big(\mathsf{B}\big(\mathsf{S}(\mathsf{K}\mathsf{B})\big)\big(\mathsf{C}(\mathsf{B}\mathsf{S}(\mathsf{B}\mathsf{K}\mathsf{I}))\mathsf{I}\big)\Big)\Big)(\mathsf{K}\mathsf{I})\Big) \quad .$$

Chapter 6

Future Work

We brought together ideas used to model computations via GoI construction, adding basic proofs of a few propositions. The framework shown in [14] has just introduced recently, thus there remain room to be investigated. Especially, finding *initial algebras* and *final coalgebras* in $\mathbf{PER}_A$ leads to obtain data types available in quantum programming.

There is another way of GoI application. *Geometry of Interaction machine* was developed by Mackie. In his paper [17], a certain programming language is given a semantics in extended GoI interpretation and it is directly compiled into assembly language. Recently another method, which is closer to machine level, was presented by Ghica et al. in [5] and subsequent papers [6, 7, 8]. This technique, called *Geometry of Synthesis*, enables us to implement a program written in a certain programming language as an actual circuit on FPGA. Putting together with the existence of GoI model of quantum programming language, we have a possibility to develop a method to compile a quantum program into corresponding quantum circuit directly. Since the model is general enough to be applied to other computations with branching, this possibility also holds for such computations. We intend to pursue studies in this direction.

Additionally, it may be useful to develop a tool to convert linear λ-calculus terms into string diagrams of category. As we saw, linear λ-calculus terms can be translated into arrows of a certain category via linear combinatory terms. However, the process to make corresponding arrows easily become so complex for human to write by hand. Thus such a tool will help us to obtain concrete examples.

References

[1] Samson Abramsky, Esfandiar Haghverdi, and Philip Scott. Geometry of interaction and linear combinatory algebras, 2000.

[2] Samson Abramsky and Radha Jagadeesan. New foundations for the geometry of interaction. *Information and Computation*, 111:53–119, 1993.

[3] Hendrik Pieter Barendregt. *The Lambda Calculus – Its Syntax and Semantics*, volume 103. North-Holland, 1984.

[4] Vincent Danos and Roberto Di Cosmo. The linear logic primer. http://www.dicosmo.org/CourseNotes/LinLog/CorsoPisa.pdf, 2009 last modified.

[5] Dan Razvan Ghica. Geometry of synthesis: a structured approach to vlsi design. In Martin Hofmann and Matthias Felleisen, editors, *POPL*, pages 363–375. ACM, 2007.

[6] Dan Razvan Ghica and Alex Smith. Geometry of synthesis ii: From games to delay-insensitive circuits. *Electr. Notes Theor. Comput. Sci.*, 265:301–324, 2010.

[7] Dan Razvan Ghica and Alex Smith. Geometry of synthesis iii: resource management through type inference. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 345–356. ACM, 2011.

[8] Dan Razvan Ghica, Alex Smith, and Satnam Singh. Geometry of synthesis iv: compiling affine recursion into static hardware. In Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy, editors, *ICFP*, pages 221–233. ACM, 2011.

[9] Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

[10] Jean-Yves Girard. Geometry of interaction 2: deadlock-free algorithms. In Per Martin-Löf and Grigori Mints, editors, *Conference on Computer Logic*, volume 417 of *Lecture Notes in Computer Science*, pages 76–93. Springer, 1988.

[11] Jean-Yves Girard. Geometry of interaction 1: Interpretation of system f. *Logic Colloquium 88*, 1989.

[12] Jean-Yves Girard. Geometry of interaction iii: Accommodating the additives. In *In: Advances in Linear Logic, LNS 222,CUP, 329389*, pages 329–389. Cambridge University Press, 1995.

[13] Jean-Yves Girard. Light linear logic. *Inf. Comput.*, 143(2):175–204, 1998.

[14] Ichiro Hasuo and Naohiko Hoshino. Semantics of higher-order quantum computation via geometry of interaction. In *LICS*, pages 237–246, 2011.

[15] André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Math. Proc. Cambridge Philos. Soc.*, 119(3):447–468, 1996.

[16] Saunders Mac Lane. *Categories for the Working Mathematician (Graduate Texts in Mathematics)*. Springer, 2nd edition, 1998.

[17] Ian Mackie. The geometry of interaction machine. In *POPL*, pages 198–208, 1995.

[18] Wesley Phoa. An introduction to fibrations, topos theory, the effective topos and modest sets. http://www.lfcs.inf.ed.ac.uk/reports/92/ECS-LFCS-92-208/, 2006 last modified.

[19] Peter Schauenburg. Turning monoidal categories into strict ones. *New York J. Math.*, 7:257–265, 2001.

[20] Peter Selinger. A survey of graphical languages for monoidal categories, 2009. http://www.mscs.dal.ca/ selinger/papers/graphical.pdf, 2011 last modified.

[21] Alex Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In *RTA*, pages 219–234, 2005.