Logical Relation Categories and Lambda Calculi

Liqun Yang

submitted to the School of Graduate Studies and Research
in partial fulfillment of the requirements for
the Master's degree in Mathematics

University of Ottawa
Ottawa, Ontario
Canada

17 February, 1996

The M.Sc. Program is a joint program with
Carleton University, administered by the Ottawa-Carleton
Institute of Mathematics and Statistics

ISBN 0-612-15779-2

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Acknowledgments

I would like to express my deepest gratitude to my supervisor Professor Philip Scott. His continuous support, interesting discussions and encouragement made my research project possible. I would have done a better job if I had followed his guidance.

I would like to thank Dr. Hongde Hu, Dr. Nax Mendler and Professor Michael Makkai for providing valuable time and useful information.

Finally, I would like to thank the University of Ottawa and the Mathematics Department for their financial support.

Logical Relation Categories and Lambda Calculi

(Abstract)

Liqun Yang

February 18, 1996

There has recently been a lot of activity in theoretical computer science on the connection between logic, category theory and foundations of programming languages.

The typed lambda calculus has been extensively used in describing, analyzing and implementing programming languages. In fact, the family of so-called *functional programming languages*, has been inspired directly from λ-calculus. Untyped lambda calculus itself is a universal programming language. Because of its simple syntax many problems concerning programming languages are seen through the λ-calculus in a pure form: for example, Dana Scott's construction of models of the λ-calculus led to the theory of denotational semantics for programming languages.

Another aspect of programming languages is the study of the operational semantics, which, in the case of a lambda calculus, is based on a directed form of equational reasoning called *reduction*. In computer science terminology, reduction may be regarded as a form of symbolic evaluation. It models a sequential computation process step by step. The crucial properties for a rewriting system are *confluence*, also called the *Church-Rosser property* and *termination*, the *(Strong) normalization property*, respectively. These are studied in depth in Chapter 2 and 6. The problem whether all λ-terms satisfy termination corresponds to the *halting problem*. From a different point of view, studying the problem of both Church-Rosser and strong normalization corresponds to, in a particular field, studying the *word problem*.

The notion of *categories* is familiar to mathematicians as a branch of algebra. It has been developed quite rapidly in less than 40 years. In particular, recently more and more computer scientists are using categories for their own purposes.

It is the connection between categories and λ-calculi that interests computer scientists.

This connection is a thread that goes through the whole thesis. In particular, it is investigated and studied in Chapter 2.

From a logical point of view, there are two aspects to study a calculus system: a *model theory* to study the static behavior; a *proof theory* to study the dynamic behavior. In computer science terminology, the first one corresponds to *denotational semantics*, and the second one to *axiomatic* and *operational semantics* depending on how one wants to model the *dynamics*.

The so called Curry-Howard-Lambek isomorphisms transform lambda calculus, proof theory and category theory into a single subject which is at the center of programming language theory. In particular, the concept of cartesian closed category plays a crucial role.

In this thesis we study logical relations, which are a kind of analog of "homomorphism" in the lambda calculus. The primary objective of this work is to understand further how logical relations, cartesian closed categories and lambda calculi connect, affect and complement each other.

This thesis is divided into six chapters. The first chapter is an introduction to the background and a little bit of history of the subject. The second and third chapters present the main subjects, i.e., lambda calculi, cartesian closed categories and logical relations.

We give a rapid survey of known results (some are folklore at the present time). We have tried to get the history correct, with approprite credits and references. Chapter 4 is a look at logical relation categories and their properties. As far as we know, much of this work is new. In our view, most of the important results in this field are organized very well in a single environment. The previous chapters mostly focus on the model theoretic part of the simply typed lambda calculus. In Chapter 5, as an example of logical relation categories, we briefly introduce Lambek's $C^{\mathcal{R}}$ construction. In Chapter 6 we investigate the dynamic part. We apply Dougherty's reduction rules on simply typed lambda calculus with coproduct types. By using this theory as our working model, a very effective and yet very natural form for representation is developed. For example, using this argument, Cubric's theorem and similar results can be easily proved. Although most of the results are known from the literature, our approach is different.

Finally we discuss some unfinished work and propose some interesting future work.

Logical Relation Categories and Lambda Calculi

Liqun Yang

February 18, 1996

Contents

Chapter 1

Introduction

This thesis is about lambda calculi and categories. We focus on cartesian closed categories and logical relations. Some important recent metatheorems by Friedman, Statman, Plotkin, and Cubric are studied.

1.1 What Is a λ-calculus?

The λ-calculus and its variable free equivalent, combinatory logic, were developed between 1925 - 1930 by logicians Schönfinkel, Church and Curry, independently. The intention of the founders of the subject was to study *rules*; in other words to study the old-fashioned notion of *function* in the sense of *definition*, i.e., as a process of correspondence from the *active variables* to the *passive variables*. This is in contrast to Dirichlet's notion of *graph*, that is the set of pairs of argument and associated value.

The λ-calculus is a calculus of functions. The main constructs are *lambda abstraction*, which we use to write function expressions, and *application* of a function to an argument, which allows us to make use of the functions we define. The domain of a function is specified by giving a type to the *formal parameter*. We read the term $\lambda x : \sigma.M$ as "the function defined by treating the expression M as a function of the variable x". Informally, it corresponds to the function $x \mapsto M(x)$ where $M(x)$ is some expression in x. The notation "$x : \sigma$" explicitly specifies that the formal parameter x varies over type σ, so the domain of $\lambda x.M$ is σ. The range of it is determined from the form of M using the typing rules of the language. A simple example is the following expression

$$\lambda x : int.x$$

for the identity function on integers, $x \mapsto x$. Since the declaration $x : int$ guarantees that the function body x has type int, the range of this function is int. To say that the integer identity function is a mapping from integers to integers, we write

$$\lambda x : int.x \ : int \longrightarrow int$$

The λ-calculus studies functions and their applicative behavior, and not, as in category theory, just their behavior under composition. Application is the primitive operation of the λ-calculus. The function f applied to the argument a is denoted by fa.

Example 1 Consider the polynomial $P(x) \equiv x^2 + bx + c$. Using the λ-notation, we can express the function $x \mapsto P(x)$ by $\lambda x : int.P(x)$ or $\lambda x : real.P(x)$ (depending on the range of the above polynomial). The ordinary instantiation $P(a)$, for some constant a, is simply the application $(\lambda x.P(x))a$. A basic rule of the λ-calculus is *the* (β)*-rule:*

$$(\beta) \qquad (\lambda x.P(x))a = P(a) = \text{ the substitution } P(a/x) = a^2 + ba + c$$

Notice that substitution of a for x corresponds to evaluating this polynominal function at a.

An important aspect of lambda abstraction is that the symbol λ in $\lambda x : \sigma.M$, like $\int$ in $\int_b^a f(x)dx$, is a binding operator. The variable x has no particular meaning at all. It is a local variable being bound in the range from a to b, in the latter, in the range of σ, in the former. Just as $\int_b^a f(x)dx$ and $\int_b^a f(y/x)dy$ are two different ways of writing the same integral (the same value), $\lambda x : \sigma.M$ and $\lambda y : \sigma.M[y/x]$ are different ways

of writing the same function. The notation y/x or $[y := x]$ reads " substitute the free occurrence of x by y". One should not be surprised to realize that *substitution* is more complex in the study of lambda calculi than in the study of integration. We will see this in Chapter 2.

Lambda calculus has been extensively used in describing, analyzing and implementing programming languages. In fact, a big family of programming languages, so called *functional programming languages*, see e.g. [4], has been inspired directly by λ-calculus. On the other hand, untyped lambda calculus itself is a universal programming language. Because it is such a simple language, many problems concerning programming languages are seen through the λ-calculus in a pure form. As one aspect of this phenomenon, Dana Scott's construction of models of the λ-calculus led to the theory of denotational semantics for programming languages, see e.g. [49].

The second aspect of programming languages is the study of the operational semantics, which, in the case of a lambda calculus, is based on a directed form of equational reasoning called *reduction* or term rewriting. In computer science terminology, reduction may be regarded as a form of symbolic evaluation. Equations are evaluated by replacing the left hand side of an equality by the right hand side. It models a sequential computation process step by step. The crucial properties for a rewriting system (a group of reduction rules) to satisfy are *confluence* and *termination*, respectively. They are also called the *Church-Rosser property* and the *(Strong) normalization property*. For example, roughly, the problem that all λ-terms satisfy termination corresponds to the *halting problem*. From a different point of view, studying the problem of both Church-Rosser and strong normalizability corresponds to, in this particular field, studying the decidability of the *word problem* for the (free) term algebra.

1.2 What about Categories?

The subject of *category theory* is more familiar to mathematicians as a branch of algebra. It has been developed quite rapidly in less than 40 years. In particular, recently more and more theoretical computer scientists are looking at different categories for their own interests and purposes. It seems that today more computer scientists than mathematicians are concerned with categories.

Why do computer scientists pay such close attention to categories? One reason is the connection between categories , λ-calculi, and the foundations of functional programming languages.

From a logical point of view, there are two aspects to studying a formal system: a *model theory* to study the static behavior; a *proof theory* to study the dynamic behavior. In computer science terminology, the first one corresponds to *denotational semantics*, and the second one to *axiomatic* and *operational semantics* depending on how one wants to model the *dynamics*. Roughly speaking, for a given lambda calculus, the study of the dynamic part is to look into the rewriting theory, to study the inference rules; the study of the static part is to look into the underlying algebraic structures, to find the internal meanings. Certain categories (here, cartesian closed ones), as abstract algebraic structures, serve as the candidates to be models of the lambda calculi.

In fact, lambda calculi and cartesian closed categories attempt to describe axiomatically (though from a different point of view) a theory of functionals. The so-called Curry-Howard-Lambek isomorphisms connect lambda calculi, proof theory and category theory in such a way that none of them can be ignored if one wants to know the subject. For a detailed investigation of this relation see, e.g.,[24].

1.3 What Is a Logical Relation?

In the study of an algebraic structure, many properties can only be seen via a "dynamic" environment, or from another point of view, via an interaction of it with others. This can be done through homomorphisms. Logical relations are a kind of analog of homomorphisms in the study of λ-calculus. In Chapter 3 and 4, we investigate, in detail, this observation from different views.

The study of *logical relations* originated from the study of *computability predicates*. The first published work, as far as we know, is Tait [50], which is influenced by, among other things, Kreisel and Howard's work. The "Brouwer-Heyting-Kolmogorov" (BHK) interpretation of the *intuitionistic logic* was very influential on these early works. See e.g. [51], [24].

The BHK view is basically a *proof-oriented*, or "functional" interpretation compared to the old *truth-oriented* classical view of Tarski. Under this interpretation, for example, a proof of $A \wedge B$ is a pair, the first component is a proof of A, and the second is a proof of B; a proof of $A \longrightarrow B$ is a function which when applied to any proof of A, gives a proof of B; and so on. Everything is fine, except for the atomic formulas. As Heyting said: *the proof of an atomic formula is just a verification of its truth.* For example, in arithmetic, this can be done by knowing that say, $2 = 2$ but $2 \neq 3$. Consequently, this depends on how people *understand* truth at the atomic level. In other words, one has the freedom to interpret *the basic truth.*

We will see that the construction of *logical relations* (or *predicates* for the unary case) is deeply related to this interpretation: there is an atomic part and a compound higher-order part.

The higher-order part is defined functionally. For example, for a logical predicate (following Tait, we call it *computability*), we say a function $f : A \longrightarrow B$ is computable if for all computable arguments $a \in A$, the value $f(a)$ is computable.

1.4 Organization of This Thesis

The primary objective of this work is to understand further how logical relations, categories and lambda calculi connect, affect and complement each other. Here lambda calculi and cartesian closed categories are the main objects: logical relations are described both as a tool, and also for their own sake.

This thesis is divided into six chapters. In the second chapter we introduce the necessary background. Various typed lambda calculi are defined, with some essential properties developed. We introduce cartesian closed and bicartesian closed categories and mention the fundamental connections between the syntax of lambda calculi and categories.

Logical relations (predicates) are the subject of Chapter 3. Logical relations are defined on the right models, *Henkin models*, as well as on some weaker structures: the bare terms and even the applicative structures (which are a kind of raw candidate for models). We include some material (e.g. Volger's method of adding indeterminates in Chapter 2.5) which are not well-known. We also have tried to be historically accurate, attributing quoted results to the the appropriate persons (as far as we know). Chapter 4 is the main chapter, where we develop categories of logical relations and compare them with different models in the literature. As far as we know, this is new material. In our view, most of the important results in this field are organized clearly in a single framework. The previous three chapters mostly focus on the model theoretic part of the simply typed lambda calculus. In Chapter 5, as an example of logical categories, we briefly introduce Lambek's $C^{\mathcal{R}}$ construction. In Chapter 6 we investigate the dynamical part of typed lambda calculus. We apply Dougherty's reduction rules on simply typed lambda calculus with coproduct types. By using this theory as our working model, a very effective and yet very natural way of arguing about representations is developed. For example, using this argument, Cubric's theorem and similar results can be easily proved.

In the last part of this thesis, we discuss some unfinished work and propose some interesting future work.

1.5 Prerequisites for Reading This Thesis

We assume the reader is familiar with the standard literature of typed lambda calculus, cartesian closed categories and proof theory, and logical relations. In particular, this includes:

1. Lambek-Scott [24], Part 0, I and the Section 20 of Part II.
2. Mitchell [33].
3. Mitchell-Scott [36].
4. Girard-Lafont-Taylor [13].

We use standard notation from logic.

Chapter 2

λ-calculi and Categories

2.1 λ-calculi

2.1.1 Simply Typed Lambda-Calculus

A typed λ-calculus is a formal theory consisting of types, terms and equations. To each term t, there corresponds a type τ, called the type of t. They are subject to the following rules.

Types: The set $\mathbf{T}$ of types contains

1. A collection of base types, including at least 1.
2. If $\sigma, \tau \in \mathbf{T}$, then $\sigma \to \tau, \sigma \times \tau \in \mathbf{T}$.
3. There may be other base types or type-forming operations not indicated by 1, 2 above, and there may be identifications between types.

Terms: The class of terms is freely generated from variables and certain basic constants, including the constant $*$, by certain term-forming operations as follows.

Notation: In what follows we will use the notion of type assignment [33]. We write $x : \tau$ to mean "x is a variable of type τ" . A type assignment Γ is a finite set of expressions of the form $x_i : \tau_i$ associating $\tau_i \in \mathbf{T}$ to variables x_i, with no variable x_i occurring twice. We write $\Gamma \rhd t : \tau$ to say "t is a term of type τ under type assignment Γ".

For each type τ there are countably many variables of type τ. Besides that there are also the following freely generated classes of terms .

I-terms (Introduction Terms)

(I1) $\Gamma \rhd x : \tau$ if $x : \tau$ is in Γ

(I2) $\Gamma \rhd * : 1$ ($*$ the distinguished constant)

(I3) $$\frac{\Gamma, x : \sigma \rhd t : \tau}{\Gamma \rhd \lambda x : \sigma . t : \sigma \to \tau}$$ (λ–abstraction)

(I4) $$\frac{\Gamma \rhd s : \sigma \quad \Gamma \rhd t : \tau}{\Gamma \rhd (s, t) : \sigma \times \tau}$$ (pairing)

E-terms (Elimination Terms)

(E1) $$\frac{\Gamma \rhd f : \sigma \to \tau \quad \Gamma \rhd t : \sigma}{\Gamma \rhd ft : \tau}$$ (application)

(E2) $$\frac{\Gamma \rhd x : \sigma \times \tau}{\Gamma \rhd \pi_1(x) : \sigma}$$

(E3) $$\frac{\Gamma \rhd x : \sigma \times \tau}{\Gamma \rhd \pi_2(x) : \tau}$$

Remark 1 *Immediate subterms* of a term t are defined inductively as follows:

1. t is an immediate subterm of $\lambda x.t$ and $\pi_1(t), \pi_2(t)$,
2. t_i is an immediate subterm of $\langle t_1, t_2 \rangle$,
3. t, s are immediate subterms of ts.

Remark 2 For a term $\Gamma \rhd t : \tau$ we define the sets of free and bound variables FV(t), BV(t) respectively, as usual. The basic idea is $x \in \mathrm{FV}(t)$ iff x occurs in t but is not bound by any λ's. Note: FV(t)$\subseteq$ the variables in Γ.

Remark 3 For simplicity, we will write $\pi_i t$ in place of $\pi_i(t)$ from now on. This makes the π_i look like terms instead of operators as they are here. However, this confusion will do no harm at all. In fact, an alternative notation is really to introduce $\rhd \pi_i : \sigma_1 \times \sigma_2 \longrightarrow \sigma_i$ as term constants for any types σ_1, σ_2. In that way, $\pi_i t$ is again an elimination term but is an instance of application (E1).

From now on, we will use Λ^σ for the set of terms of type σ, V^σ for the set of variables of type σ, and Λ for the set of all terms, hence

$$\Lambda = \bigcup_{\sigma \in \mathbf{T}} \Lambda^\sigma$$

Equations: Suppose $\Gamma \rhd t : \tau, \Gamma \rhd s : \tau$. A *provable equation* on Λ^τ is an expression of the form

$$\Gamma \rhd t = s : \tau$$

It reads as "t equals to s of type τ, relative to type-assignment Γ." The binary relation $=$ is imposed on Λ to be reflexive, symmetric and transitive, and also satisfy the usual substitution rules for all term forming operations. Thus it satisfies the following equations:

Equivalence Rules

$$\Gamma \rhd s = s : \tau$$

$$\frac{\Gamma \rhd s = t : \tau}{\Gamma \rhd t = s : \tau}$$

$$\frac{\Gamma \rhd t = s : \tau \quad \Gamma \rhd s = r : \tau}{\Gamma \rhd t = r : \tau}$$

Congruence Rules

$$\frac{\Gamma \rhd t = s : \sigma \quad \Gamma \rhd f : \sigma \to \tau}{\Gamma \rhd ft = fs : \tau}$$

$$\frac{x : \sigma, \quad \Gamma \rhd u = v : \tau}{\Gamma \rhd \lambda x : \sigma.u = \lambda x : \sigma.v : \sigma \to \tau}$$

$$\frac{\Gamma \rhd t = s : \sigma \quad \Gamma \rhd t' = s' : \tau}{\Gamma \rhd \langle t, t' \rangle = \langle s, s' \rangle : \sigma \times \tau}$$

$$\frac{\Gamma \rhd t = s : \sigma_1 \times \sigma_2}{\Gamma \rhd \pi_i t = \pi_i s : \sigma_i}$$

Computational Rules: Corresponding to the two main operations, the computational rules are:

(β) $$\Gamma \rhd (\lambda x : \sigma.t)s = t[x := s] : \tau$$

(π) $$\Gamma \rhd \pi_i(t_1, t_2) = t_i : \tau_i \quad i = 1, 2$$

Extensional Rules: As a complement to the computational rules, we have the following:

$(1!)$ $\quad \Gamma \rhd t = * : 1$

$(\times!)$ $\quad \Gamma \rhd t = \langle \pi_1 t, \pi_2 t \rangle : \sigma \times \tau$

(η) $\quad \Gamma \rhd f = \lambda x : \sigma.(fx) : \sigma \to \tau \quad$ if $x \notin \mathrm{FV}(f)$.

Remark 4 1. The first two congruence rules entail the other two congruence rules, in the presence of the Computational and Extensional equations (See [24], p.73).

2. To compare the treatment of lambda calculi here with Lambek-Scott [24], (for the purpose of this thesis) $\Gamma \rhd t = s : \tau$ is identified with the notation $t =_\Gamma s$ in [24].

Remark 5 The *renaming bound variable rule* (α) is the following

$$\lambda x.t = \lambda y.t[x := y] \text{ if } y \text{ does not occur in } t \text{ already.}$$

This will be assumed and used without further notice, i.e., we will always identify terms s and s' if they differ only in their choice of bound variables.

2.1.2 Rewriting Theory

Rewriting theory is very effective for studying λ-calculi [13], [24]. The idea is to *orient* the equations of λ-calculus: we think of the *computation rules* as rewriting the left hand side to the right hand side of an equation. Under the familiar Curry-Howard isomorphism ([13]), λ-terms correspond to natural deduction proofs, and computation by rewriting corresponds to *normalization* of proofs. For example, the rule

$$\Gamma \rhd \pi_1\langle t_1, t_2 \rangle = t_1 : \tau_1$$

when oriented left to right (see rule (π) below) corresponds to the following normalization:

$$\dfrac{\dfrac{\vdots \quad \vdots}{C \vdash A \quad C \vdash B}}{\dfrac{C \vdash A\&B \qquad A\&B \vdash A}{C \vdash A}} \quad \succ \quad \begin{array}{c}\vdots \\ C \vdash A\end{array}$$

With a little effort, we can develop two groups of rewriting rules from the specific equations for a λ-theory. In fact, the computation rules always go from left to right. The only difference occurs from orienting the extensional rules.

Rewriting Rules There are two groups of rewriting rules corresponding to the above specific equations. We start a series of *reductions* by orienting the equations from left to right.

Group 1: Let "$\Rightarrow$" denote basic reduction:

(β) $\quad (\lambda x : \sigma.t)s \Rightarrow t[x := s]$

(π) $\quad \pi_i\langle t_1, t_2 \rangle \Rightarrow t_i : \tau_i \quad i = 1, 2$

$(1!)$ $\quad t \Rightarrow * \text{ for } t : 1 \text{ and } t \neq *$

$(\times - r)$ $\quad \langle \pi_1 t, \pi_2 t \rangle \Rightarrow t$

$(\eta - r)$ $\qquad\qquad$ $\lambda x.(fx) \Rightarrow f$

The rules "$\times - r$" and "$\eta - r$" denote *contraction* rules: the RHS is genuinely simpler than the LHS.

In general, for a λ-term t if any subterm of it can be reduced by applying one $\Rightarrow$ we will say that t can be reduced by applying $\Rightarrow$. More precisely, we will introduce the following two notations

- $t \overset{1}{\Rightarrow} s$ for applying $\Rightarrow$ once to a subterm of t.
- $\overset{*}{\Rightarrow}$ for finitely many compositions of $\overset{1}{\Rightarrow}$.

Group 2: These rules reverse some of orientations above. Let $\longrightarrow$ denote a basic reduction. $(\beta), (\pi)$ and $(1!)$ are the same as **Group 1**. The following orientations are postulated

$(\times - e)$ $\qquad\qquad$ $t \longrightarrow \langle \pi_1 t, \pi_2 t \rangle$

$(\eta - e)$ $\qquad\qquad$ $f \longrightarrow \lambda x.(fx)$

There are two restrictions on these two rules:

1. For using $(\times - e)$, t is neither of the form $\langle t_1, t_2 \rangle$, nor in a context like $C[(\pi_1 t)], C[(\pi_2 t)]$.
2. For using $(\eta - e)$, f is neither of the form $\lambda y.t$, nor in a context like $C[(fs)]$.

Just as for $\Rightarrow$, the "operation" $\longrightarrow$ can be generalized to $\overset{1}{\longrightarrow}$ and $\overset{*}{\longrightarrow}$ by:

- $t \overset{1}{\longrightarrow} s$ for applying $\longrightarrow$ once to a subterm of t.
- $\overset{*}{\longrightarrow}$ for finitely many composition of $\overset{1}{\longrightarrow}$.

Unlike the rules in the first group, the rules in **Group 2**, especially $(\times - e)$ and $(\eta - e)$ are *context sensitive*. Also these two rules are "expansion" rules: the RHS is expanded from the LHS. Hence, it is not *obvious* that Group 2 yields a terminating rewriting system.

Remark 6 Notice that the restrictions on $(\times - e), (\eta - e)$ are necessary and reasonable. Without them, we might have the following loops:

$$\langle t_1, t_2 \rangle \overset{(\times - e)}{\longrightarrow} \langle \pi_1 \langle t_1, t_2 \rangle, \pi_2 \langle t_1, t_2 \rangle \rangle \overset{\pi}{\longrightarrow} \langle t_1, t_2 \rangle$$

$$\pi_i t \overset{(\times - e)}{\longrightarrow} \pi_i \langle \pi_1 t, \pi_2 t \rangle \overset{\pi}{\longrightarrow} \pi_i t$$

$$\lambda y.t \overset{(\eta - e)}{\longrightarrow} \lambda x.(\lambda y.t)x \overset{\beta}{\longrightarrow} \lambda x.t[y := x] = \lambda y.t$$

$$fs \overset{(\eta - e)}{\longrightarrow} (\lambda x.fx)s \overset{\beta}{\longrightarrow} fs$$

These loops are certainly not what we want from a computational view point.

In general, if we look at the reductions or expansions as a relation, we may want to consider the closure properties of it. The smallest equivalence relation containing the rewriting rules, i.e., the symmetric and transitive closure of it, is an equational theory, called the equivalence relation *generated* from the reductions. If, furthermore, the equivalence relation is a congruence with respect to the term forming operations, we will get an algebraic structure on Λ, the *quotient algebra*. In fact, this structure, if nontrivial, serves as a model of the equational theory. We will discuss this in detail in Chapter 6. The following theorem is presented in Section 6.2.

Proposition 1 *Both groups of rules generate the same eq... ...nce relation* $\sim$, *and it is a congruence with respect to the term forming operations.*

It might seem strange from the viewpoint of computation that the rules are oriented as expansions. Let us look at an example.

Let x be a variable of type $(o \to o) \to (o \to o)$ for some base type o. Notice, x is supposed to be a functional though it does not look like one. However, let us use our rules to do some rewriting to see what will happen. First, x is not an irreducible term and we could use an η-expansion on it: this is different from the contraction rewriting rules. By doing an η-expansion, we have $x \xrightarrow{\eta} \lambda f^{(o\to o)}.xf$. Well here we have a choice, either do an inside one reducing f, or an outside one reducing (xf). Suppose we choose the outside reduction, then we have

$$\lambda f^{(o\to o)}.xf \xrightarrow{\eta} \lambda f^{(o\to o)}y^o.(xf)y$$

Then another η application on f gives us the following term

$$\lambda f^{(o\to o)}y^o.x(\lambda z^o.fz)y$$

This is irreducible and called the *normal form* of x which is, on the face of it, a functional.

If we chose to do the inside reduction first and then another outside reduction this leads to the same normal form.

To make a term of function type appear as a function (i.e., as an explicit λ-term) is one of the main features of the system with expansion rules. The same situation happens for product types.

It is Mints who first suggested this system and after that many people studied it [2], [7], [9], [16], just to name a few of them.

2.1.3 Lambda-Calculus with Coproduct Types

Another useful data type is sum, which is extensively used in daily mathematics as well as in many programming languages such as case in ML. However in general the coproduct operation is more complex than the product operation in mathematics, especially in category theory. When one wants to introduce sum types and the terms of this type there are always some extra troubles to be overcome.

We extend the previous definition of types and terms in simply typed λ-calculus as follows:

Coproduct Types We introduce two more new type forming operations:

4. If σ, τ are types so is $\sigma + \tau$.
5. 0 is a type.

Coproduct Terms Some new terms and term forming operations are added:

I-terms

1. $$\frac{\Gamma' \rhd f_1 : \sigma \to \rho \quad \Gamma \rhd f_2 : \tau \to \rho}{\Gamma', \Gamma \rhd [f_1, f_2] : \sigma + \tau \to \rho}$$

2. $$\Gamma, x : \sigma \rhd \kappa_i x : \sigma + \tau$$

3. $$\Gamma \rhd \Box_\sigma : 0 \to \sigma$$

E-terms There are no new E-terms.

Coproduct Equations Three new equations related to sum-type and corresponding to the ordinary rules from bicartesian closed categories [24] are added:

Computation Equations (continued)

(κ) $\qquad [f_1, f_2](\kappa_i t) = f_i t,\ i = 1, 2$

Extensional Equations(continued) Two more equations

$(+!)$ $\qquad f = [\lambda x^\sigma . f(\kappa_1 x), \lambda y^\tau . f(\kappa_2 y)]$, if $f : \sigma + \tau \to \rho$

$(0!)$ $\qquad f = \Box$, if $f : 0 \to \sigma$

We will use Λ^+ to denote the set of all terms including coproduct ones.

Coproduct Rewriting Rules There are two groups of rules obtained by orienting the above extensional equations from left to right or from right to left. The basic orientations are given below. The groups here are continued from the two rewriting groups before, respectively.

Group 1:

(κ) $\qquad [f_1, f_2](\kappa_i t) \Rightarrow f_i t,\ i = 1, 2$

$(+ - r)$ $\qquad [\lambda x^\sigma . f(\kappa_1 x), \lambda y^\tau . f(\kappa_2 y)] \Rightarrow f$

$(0!)$ $\qquad f \Rightarrow \Box$, if $f : 0 \to \sigma$, and $f \neq \Box$.

This group has no extra restrictions.

Group 2:
$(\kappa), (0)$ are as in **Group 1**

$(+ - e)$ $\qquad f \longrightarrow [\lambda x^\sigma . f(\kappa_1 x), \lambda y^\tau . f(\kappa_2 y)]$
if f is a λ-abstraction.

$(\eta - e)$ $\qquad f \longrightarrow \lambda x.(fx)$
if f is neither of the form $\lambda y.t$, or $[f_1, f_2]$, nor in a context like $C[(fs)]$.

The **immediate subterms** of $[f_1, f_2]$ and $\kappa_i t$ are f_1, f_2, and t respectively.

The above rules for sum type, first appeared in the work of Dougherty [9].
The following is a brief summary of the syntax and rewriting theory of typed λ-calculi with coproduct types.

The Syntax of Lambda Calculus	
I-Terms	$\Gamma \rhd * : 1$ $\Gamma \rhd x : \tau$ if $x : \tau$ in Γ $\Gamma \rhd \lambda x : \sigma.t : \sigma \to \tau$ $\Gamma \rhd \langle s, t \rangle : \sigma \times \tau$ $\Gamma \rhd \Box_\sigma : 0 \to \sigma$ $\Gamma, x : \sigma_i \rhd \kappa_i x : \sigma_1 + \sigma_2 \; i = 1, 2$ $\Gamma', \Gamma \rhd [f_1, f_2] : \sigma + \tau \to \rho$
E-Terms	$\dfrac{\Gamma \rhd f : \sigma \to \tau \quad \Gamma \rhd t : \sigma}{\Gamma \rhd ft : \tau}$ $\dfrac{\Gamma \rhd x : \sigma \times \tau}{\Gamma \rhd \pi_1(x) : \sigma \quad \Gamma \rhd \pi_2(x) : \tau}$

(Figure 1)

The following charts present the equations.

The Computational Equations	
(β)	$\Gamma \rhd (\lambda x : \sigma.t)s = t[x := s] : \tau$
(π)	$\Gamma \rhd \pi_i(t_1, t_2) = t_i : \tau_i \quad i = 1, 2$
(κ)	$[f_1, f_2](\kappa_i t) = f_i t, \; i = 1, 2$

The Extensional Equations	
$(+!)$	$f = [\lambda x^\sigma.f(\kappa_1 x), \lambda y^\tau.f(\kappa_2 y)]$
$(0!)$	$f = \Box$, if $f : 0 \to \sigma$
$(1!)]$	$\Gamma \rhd t = * : 1$
$(\times!)$	$\Gamma \rhd t = \langle \pi_1 t, \pi_2 t \rangle : \sigma \times \tau$
(η)	$\Gamma \rhd f = \lambda x : \sigma.(fx) : \sigma \to \tau$ if $x \notin \mathrm{FV}(f)$

The Equations of Simply Typed Lambda Calculus
(Figure 2)

The following chart is a summary of rewriting rule groups.

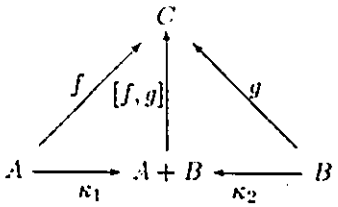
Rewriting Theories of Lambda Calculus	
R^-	$(\beta), (\pi), (\times - r), (\eta - r), (\kappa), (+ - r)$
R	$(\beta), (\pi), (\times - e), (\eta - e), (\kappa), (+ - e), (1!), (0!)$
R^∞	$(\beta), (\pi), (\times - r), (\eta - r), (\kappa), (+ - r), (1!), (0!)$

(Figure 3)

The rewriting groups R, R^- and R^∞ are generated by the basic reduction rules shown above. Furthermore, R^- is the usual reduction system for products and coproducts without units. R^∞ is the reduction system R^- with units adjoined. (To see how the units affect rewriting properties, see Lambek-Scott [24], p.81.) R is the system with Dougherty's expansion rules.

There is another widely used presentation for terms related to coproducts, which originates from natural deduction in proof theory. See the book by Girard, Lafont and Taylor [13]. We use Dougherty's notation

because of the following reason: this representation of coproducts corresponds well with coproducts in category theory:



For example, the rules on coproduct correspond exactly to the above diagram. In this way, each term has its type explicitly presented. For example, terms of the form $[-,-]$ have functional type with domain a coproduct type; terms $\kappa_i t$ have coproduct type, and so on.

2.2 Categories

The denotational semantics of lambda calculi are generally described by using categories, especially *cartesian closed categories*. This trend started with foundational work of Lambek [18] [19] and Lawvere [26] [27], then greatly expanded with Dana Scott's influential work in the 70's [42] [43]. The reader is referred to [24] and [3] for futher references. In what follows we give a short summary of the notions the reader is expected to know.

2.2.1 CCC

One of the most important observations established in categorical logic is the close relationship between λ-calculi and cartesian closed categories (ccc's) [21] [44] (see Section 2.3 below). These categories form the basic models of the lambda calculi considered here.

We begin with the usual definition of ccc in Mac Lane [30], although we only require cartesian products rather than all finite limits. Later we shall describe a still stricter notion of ccc used by categorical logicians.

Definition[Mac Lane] A *Cartesian Closed Category* is a category $\mathcal{C}$ with finite products (including terminal object 1) , such that, for each object A, the endofunctor $(-) \times A : \mathcal{C} \to \mathcal{C}$ has a specified right adjoint $(-)^A$.

This means that we have a family of natural bijections

$$\mathrm{Hom}_{\mathcal{C}}(C \times A, B) \cong \mathrm{Hom}_{\mathcal{C}}(C, B^A)$$

for any objects B and C.

In the Mac Lane definition, universal properties only determine objects *up to isomorphism*, and ccc-functors will only preserve the structure up to isomorphism. In categorical logic we need a stricter definition of ccc to model formal logical theories (here, to model lambda calculi). The following definition of ccc is similar to Lambek-Scott [24]:

Definition A *Strict Cartesian Closed Category* is a category $\mathcal{C}$ with the following additional specified structure:

CCC-1 (Terminal Object) There is a specified object $1 \in \mathcal{C}$ and, for each object $A \in \mathcal{C}$, an arrow $\bigcirc_A : A \to 1 \in \mathcal{C}$.

This data satisfies the equation:

$$f = \bigcirc_A$$

for any arrow $f : A \to 1$.

CCC-2 (Specified Products) For any two objects $A, B \in \mathcal{C}$ there is a distinguished object $A \times B \in \mathcal{C}$ and the following distinguished arrow structure:

- Projections: $\mathcal{C}$-arrows $\pi_1^{A,B} : A \times B \to A$ and $\pi_2^{A,B} : A \times B \to B$, for all objects $A, B \in \mathcal{C}$. (We usually omit writing the superscripts on the π_i)

- Pairing Rule: Given $\mathcal{C}$-arrows $f : C \to A, g : C \to B$, we have a distinguished pairing $\langle f, g \rangle : C \to A \times B \in \mathcal{C}$ as shown below:

$$\frac{f : C \to A \quad g : C \to B}{\langle f, g \rangle : C \to A \times B}$$

The above data satisfy the following equations:

1. $$\pi_1 \circ \langle f, g \rangle = f$$

$$\pi_2 \circ \langle f, g \rangle = g$$

2. $$\langle \pi_1 \circ h, \pi_2 \circ h \rangle = h$$

for $h : C \longrightarrow A \times B$

CCC-3 (Specified Function Spaces) For any two objects $A, B \in \mathcal{C}$ there is a distinguished object $B^A \in \mathcal{C}$ and the following distinguished arrows:

- Evaluation:
$$\mathbf{ev} : B^A \times A \longrightarrow B$$

- Lambda Transpose: For any $\mathcal{C}$-arrow $f : C \times A \to B$, there is an arrow $\Lambda(f) : C \to B^A$

$$\frac{f : C \times A \longrightarrow B}{\Lambda(f) : C \longrightarrow B^A}$$

satisfying the following equations. (Where if $f : A \longrightarrow B$, $g : C \longrightarrow D$, $f \times g : A \times C \longrightarrow B \times D$ abbreviates $< f \circ \pi_1, g \circ \pi_2 >$.)

1. $\mathbf{ev} \circ (\Lambda(f) \times I_A) = f$
2. $\Lambda(\mathbf{ev} \circ (k \times I_A)) = k$ for any $k : C \longrightarrow B^A$

Example 2 The first example of cartesian closed category is *Set*, the category of all sets and functions between them; a singleton set is a terminal; the product is the ordinary cartesian product, and the exponential B^A is the set of all the functions from A to B. The evaluation **ev** is the usual evaluation map, while $\Lambda(f) : C \longrightarrow B^A$ is the map $c \mapsto (a \mapsto f(c, a))$. This category, of course, is not small.

A much smaller model is the following.

Example 3 For any set B, the full type hierarchy is a ccc. Here by a full type hierarchy, we mean the smallest full subcategory T_B of *Set* with the following objects:

1. $1, B$ are objects of T_B;
2. If X, Y are objects of T_B, so are $X \times Y$, X^Y =Hom(Y, X);

The arrows are the ordinary set-theoretic functions between objects.

More generally, for U a family of sets, we can analogously define the full type hierarchy T_U, which again is cartesian closed.

The next two examples are more sophisticated and will be studied later in Chapter 4:

Example 4 For a given group G, all the G-sets form a ccc. The objects are sets with a group G acting on them. The arrows are all the equivariant functions between them. The terminal object is the terminal object in *Set*, with trivial action. Products are given by cartesian products with the pointwise action. The exponentials are the set-theoretic function space, with "conjugation" or "contragredient" action. (See [24], [14])

Example 5 The above G-set example is a generalization of *Set*. One further generalization of G-sets is *Rel*(*Set*), sets with functions invariant under certain relations.The central subject of Chapter 4 is to study this model and its variations, which we show forms a cartesian closed category.

2.3 CCC's and Simply Typed λ-calculi

An important observation is that typed lambda calculi can be interpreted in ccc's. This is known in logic as the Soundness Theorem (for this class of models). We briefly sketch how this is done. For more details, the reader can see [24] [36].

Given a typed lambda calculus $\mathcal{L}$ generated by a set of atomic types, we interpret it in a ccc $\mathcal{C}$ as follows:

1. Interpret the atomic types as fixed objects. (The type 1 is interpreted as the terminal object of $\mathcal{C}$.)
2. Given the interpretation of atomic types, the $\{\times, \rightarrow\}$-type-structure of $\mathcal{L}$ is interpreted inductively using the product and function space operations on objects of $\mathcal{C}$.
3. The terms of $\mathcal{L}$ are interpreted by arrows in $\mathcal{C}$. In particular, a term-in-context $\Gamma \rhd t : \sigma_{n+1}$, where $\Gamma = \{x_1 : \sigma_1, \ldots x_n : \sigma_n\}$, is interpreted as an arrow $[\![t]\!] : [\![\sigma_1]\!] \times \cdots [\![\sigma_n]\!] \rightarrow [\![\sigma_{n+1}]\!]$ where $[\![\sigma_i]\!]$ is the $\mathcal{C}$-interpretation of the type σ_i. The definition is given by induction on the structure of the term t. Most of this is obvious, using the ccc-structure. We give a few examples (to simplify notation for types, we write $[\![\sigma]\!]$ as σ):
   (a) $\vec{x} : \Gamma \rhd * : 1$ interprets as $[\![*]\!] = \bigcirc_{\prod_i \sigma_i} : \prod_i \sigma_i \rightarrow 1$.
   (b) $\vec{x} : \Gamma \rhd x_i : \sigma_i$ interprets as $[\![x_i]\!] = \pi_i : \prod_j \sigma_j \rightarrow \sigma_i$, the projection onto the ith component (definable in any ccc).
   (c) Lambda abstraction interprets using the Lambda Transpose Rule of ccc's: the rule
   $$\frac{\Gamma, x : \sigma \rhd t : \tau}{\Gamma \rhd \lambda x : \sigma . t : \sigma \rightarrow \tau}$$
   interprets as follows (ignoring associativity arrows): given the interpretation of $[\![t]\!] : (\prod_j \sigma_j) \times \sigma \rightarrow \tau$ we interpret $[\![\lambda x : \sigma . t]\!]$ as $\Lambda([\![t]\!]) : \prod_j \sigma_j \rightarrow \tau^\sigma$.
4. The Soundness Theorem says: Let $\mathcal{L}$ be a typed lambda calculus interpreted in a ccc $\mathcal{C}$ as above. Then if $\Gamma \vdash t_1 = t_2$ then the interpretations of the terms t_1 and t_2 denote equal arrows in $\mathcal{C}$.

Remark The above Soundness Theorem extends to larger classes of lambda calculi with additional type- and term-forming operations or additional equations, provided the ccc models $\mathcal{C}$ have the additional structure [24], [36]. For example, in these references it is shown how to interpret (i) typed lambda calculi with coproduct types (using ccc's with finite coproducts) and (ii) typed lambda calculi with natural numbers and iteration (using ccc's with natural numbers objects).

Although we shall not need the following result, we end the discussion of ccc's by mentioning a still more general theorem. It was first proved in Lambek-Scott [24]. It says that ccc's and lambda-calculi are essentially the same thing. Consider the categories CART and λ-CALC: CART is the category of ccc's, with morphisms (strict) ccc-functors that preserve the structure on the nose; λ-CALC is the category of arbitrary typed lambda calculi, not necessarily freely generated, with morphisms strict structure-preserving translations.

Theorem 2.3.1 (Lambek-Scott,11.3) *There is an equivalence of categories between CART and λ-CALC.*

The above theorem extends to more structured ccc's, by extending the lambda calculi with additional type and term-forming operations, as we mentioned earlier.

The previous Soundness Theorem is a special case of the above: start with a freely generated lambda calculus $\mathcal{L}$. This will be an initial object in λ-CALC, i.e. will have a unique translation into any other λ-calculus. Under the equivalence of categories above, this lambda calculus translates into an appropriate free ccc (free on a certain discrete graph), which will be initial in the category CART. Thus it will have a unique interpretation into any ccc (modulo an interpretation of the basic atomic types).

2.4 Applicative Structures and Henkin Models

As a semantics for lambda calculi, abstract cartesian closed categories sometimes look too vague; more concrete models of λ-theories are often studied. Among them one is a generalized version of classic set-theoretic models, the Henkin models [33], [36].

Definition 1 For a given algebraic signature Σ, a *typed applicative structure* is an algebraic structure indexed by types $\sigma, \tau \in \mathbf{T}$,

$$\mathcal{A} = (\{A^\tau\}_\tau, \{\mathrm{ap}^{\sigma,\tau}\}, \pi_1, \pi_2, *)$$

subject to the following

1. A^τ is some set, τ is a type.
2. $* \in A^1$
3. $\mathrm{ap}^{\sigma,\tau} : A^{(\sigma \to \tau)} \times A^\tau \longrightarrow A^\tau$
4. $\pi_i : A^{\tau_1 \times \tau_2} \longrightarrow A^{\tau_i}$, $i = 1, 2$.
5. $A^{\sigma \times \tau} \subseteq A^\sigma \times A^\tau$ for each type σ, τ

Remark 7 Notice that the above definition is stronger than the traditional definition of applicative structures in [36], [33]. We require that an applicative structure be *extensional* on products from the very beginning by adding axiom 5. We believe the central operations of lambda calculi are functional application and abstraction; hence, we would like to ignore the case that there may be some ill-behaved pairs in $A^{\sigma \times \tau}$. Thus Axiom 5 says an element of $A^{\sigma \times \tau}$ is a pair from $A^\sigma \times A^\tau$.

Example 6 Let L be the pure simply typed λ-calculus with only β and η, then the set of terms Λ forms an applicative structure. The operations $\mathrm{ap}^{\sigma,\tau}$ are the term applications, and π_i are the product projections.

Example 7 Let $\mathcal{A}$ be an applicative structure , $\mathcal{A}^*$ the extension of $\mathcal{A}$ by adding infinitely many indeterminates to each type, like adding indeterminates in abstract algebra. This structure will be studied later in this chapter. Then $\mathcal{A}^*$ is an applicative structure with the same signature as $\mathcal{A}$

In order to model a λ-calculus, there are two more conditions for $\mathcal{A}$ to satisfy: extensional and rich enough. To be more set-theoretic, for extensionality we replace 2 above by

2'. $A^1 = \{*\}$

and add

6'. $A^{\sigma \to \tau} \subseteq A^{\tau A^\sigma}$ for each type σ, τ

Remark 8 Why do these rules 1, 2', 3 - 6 entail extensionality? Compare them with the three extensional equations for simply typed λ-calculus in Section 1.1. The rule 2' is clearly necessary. The other two model ($\times$!) and (η). For example, for (η), by 6 every element in $A^{\sigma \to \tau}$ is indeed a set-theoretic function from A^σ to A^τ. Consequently, if $f \in A^{\sigma \to \tau}$ then f is uniquely determined by its graph, hence $f = \lambda x \in A^\sigma . f(x)$ in the usual set-theoretic semantics. So (η) is guaranteed by 6 above.

To be rich means $\mathcal{A}$ has enough "global" elements to satisfy the equations. We need to guarantee combinatory completeness [36], or functional completeness [24]. For example, by 6, to interpret λ-abstraction, according to functional completeness we need for any $\rho, \sigma, \tau \in \mathbf{T}$, elements

1. $\mathbf{K} \in A^{\tau \to (\sigma \to \tau)}$
2. $\mathbf{S} \in A^{(\rho \to \sigma \to \tau) \to (\rho \to \sigma) \to \rho \to \tau}$
3. $(\_,\_) \in A^{\sigma \to \tau \to \sigma \times \tau}$
4. $\pi_1 \in A^{\sigma \times \tau \to \sigma}$
5. $\pi_2 \in A^{\sigma \times \tau \to \tau}$

satisfying the following equations respectively [1]

1. $\mathbf{K}xy = x$
2. $\mathbf{S}xyz = (xz)(yz)$
3. $\pi_1(x, y) = x$
4. $\pi_2(x, y) = y$
5. $(\pi_1 z, \pi_2 z) = z$

By the way, the identity function on A^τ is **SKK** for some suitable types, since for any term x (of suitable type)

$$(\mathbf{SKK})x = ((\mathbf{SK})\mathbf{K})x = (\mathbf{K}x)(\mathbf{K}x) = x$$

Definition 2 A Henkin model $\mathcal{A}$ is an extensional and rich enough applicative structure. There may or may not be an $A^\tau = \emptyset$, i.e., "empty sorts".

An *interpretation* or a *meaning function*, which is a family of functions from Λ to an applicative structure $\mathcal{A}$ will be defined below. It sends every well-typed term

$$x_1 : \sigma_1, \cdots, x_n : \sigma_n \rhd M : \tau$$

to its meaning

$$[\![x_1 : \sigma_1, \cdots, x_n : \sigma_n \rhd M : \tau]\!],$$

a set-theoretic function from $A^{\sigma_1 \times \cdots \times \sigma_n}$ to A^τ.

We define meaning functions as follows:

Definition 3 (Meaning Function) Let $\mathcal{A}$ be an applicative structure, Λ be the terms of a λ-calculus. A partial function η from the variables of Λ to the universe is called an *environment* if it maps V^τ to A^τ. Let **Env** be the set of environments. A partial (meaning) function **Val**

$$\mathbf{Val}\colon \Lambda \times \mathbf{Env} \longrightarrow \bigcup \{A^\sigma \mid \sigma \in \mathbf{T}\}$$

is defined by induction on terms as follows: (for a term $\Gamma \rhd M : \sigma$ and an environment η, $\mathbf{Val}(M, \eta)$ is denoted as $[\![\Gamma \rhd M : \sigma]\!]_\eta$)

1. $[\![\Gamma \rhd x : \sigma]\!]_\eta = \eta(x)$, for a variable x.
2. $[\![\Gamma \rhd c : \sigma]\!]_\eta = c$, for the constant $c \in \mathcal{A}^\sigma$ corresponding to $c \in \Lambda$.

[1] We assume the usual conventions in λ-calculus. For example, $\mathbf{ap}(x, y)$ is abbreviated as xy and $x_1 \cdots x_n$ is associated to the left. Thus $\mathbf{K}xy$ abbreviates $(\mathbf{K}x)y$. Also, we omit type symbols when no confusion arises.

3. $[\![\Gamma \rhd MN : \sigma]\!]_\eta = \mathrm{ap}([\![\Gamma \rhd M : \sigma \to \tau]\!], [\![\Gamma \rhd N : \sigma]\!])$

4. $[\![\Gamma \rhd M : \sigma + \tau]\!]_\eta = ([\![\Gamma \rhd \pi_1 M : \sigma]\!]_\eta, [\![\Gamma \rhd \pi_2 M : \tau]\!]_\eta)$

5. $[\![\Gamma \rhd \lambda x : \sigma.M : \tau]\!]_\eta$ is the unique set-theoretic function (if it exists) $f : A^{\sigma_1 \times \cdots \times \sigma_n \times \sigma} \longrightarrow A^\tau$ such that for any $d \in A^\sigma$, $f(d) = [\![\Gamma \rhd M]\!]_{\eta'}$, where

$$\eta'(y) = \begin{cases} d & \text{if } y = x \\ \eta(y) & \text{otherwise} \end{cases}$$

6. $[\![\Gamma \rhd M : \sigma]\!]_\eta = [\![\Gamma \rhd M : \sigma]\!]_\mu$ if μ is also an environment such that for any $x \in \mathrm{FV}(M)$, $\eta(x) = \mu(x)$.

7. $[\![\Gamma, x : \sigma \rhd M : \tau]\!]_\eta = [\![\Gamma \rhd M : \tau]\!]_\eta$ if $x \notin \mathrm{FV}(M)$

Note: The clause (5) above is what causes $[\![\_]\!]$ to be a partial function. If $\mathcal{A}$ is a Henkin model then there is a unique interpretation for any given η. For the proof of the well-definedness and uniqueness of $[\![\_]\!]$ see the proof in, say, [24], [11].

For an ordinary applicative structure, both the existence and the uniqueness of the function f interpreting the term $\lambda x : \sigma.M$ in clause (5) above might cause problems. In fact, we have the following result[33]:

Proposition 2 *Let $f : A^{\sigma_1 \times \cdots \times \sigma_n \times \sigma} \longrightarrow A^\tau$ interpret $[\![\Gamma \rhd \lambda x : \sigma.M : \tau]\!]_\eta$. Then for this case*

1. *Richness corresponds to the existence of f.*
2. *Extensionality corresponds to uniqueness of f.*

Remember that **richness** corresponds to having enough "global" elements, and **extensionality** corresponds to not having too many.

If the applicative structure is a Henkin model, then the above meaning function is total, otherwise it is a proper partial function.

As we said before, the models of typed λ-calculi are ccc's, so a Henkin model is also a ccc. In fact, the objects are the A^τ, arrows from A^σ to A^τ are the elements of $A^{\sigma \to \tau}$. It is easy to check that all the axioms for ccc's are sound.

In addition, corresponding to extensionality, Henkin Models are well-pointed[2] . Conversely any well-pointed ccc can be transformed into a Henkin model by letting, for any object σ, $A^\sigma = \mathrm{Hom}(1, \sigma)$.

Basically, we have the following.

Categories	Structures
	Applicative Structures
Ccc's	Rich Applicative Structures
Well-pointed ccc's	Henkin Models(= Rich + Extensional Applicative Structures)

So far, there is no convincing categorical analog of applicative structure.

For Henkin Models the following completeness theorem holds [36] [33]

Theorem 1 *Let $\mathcal{E}$ be any typed λ-theory closed under the rule (nonempty), then there is a Henkin model $\mathcal{A}$ with no $A^\sigma = \emptyset$ satisfying precisely the equations belonging to $\mathcal{E}$.*

[2]Here by saying a category is well-pointed we mean that the category is generated by 1. That is for any two objects A, B and any arrows $f, g : A \longrightarrow B$, if for all $a : 1 \longrightarrow A$ $f \circ a = g \circ a$ then $f = g$.

Here, (nonempty) is

$$\frac{\Gamma, x:\sigma \rhd M = N : \tau}{\Gamma \rhd M = N : \tau}$$

where $x \notin \mathrm{FV}(M) \bigcup \mathrm{FV}(N)$. It is easy to see that if type σ is empty, i.e., $A^\sigma = \emptyset$, the rule is not true. A special case is that the pure theory is closed under the above rule [36].

Example 8 Following from Example 6, we know that Λ is an applicative structure. In fact, this structure is both rich and can be made extensional as follows. The relation $\sim$ is defined on Λ by

$$s \sim t \quad \text{if} \quad \Gamma \rhd s = t \text{ where } FV(s) \bigcup FV(t) \subseteq \Gamma.$$

As we mentioned in Section 2.1, $\sim$ is congruence on terms with free variables a subset of Γ.

Theorem 2 *$\mathcal{F} = \Lambda/\sim$ is a Henkin model, and is the free one.*

Proof First, we show that $\Lambda/\sim$ is an applicative structure.

In fact, let $F^\sigma = \{[t] | t \in \Lambda^\sigma\}$. Then

$$\mathbf{ap} : F^{\sigma \to \tau} \times F^\sigma \longrightarrow F^\tau$$

is defined by

$$\mathbf{ap}([M],[t]) = [\mathbf{ap}(M,t)] = [Mt].$$

The same for projections, $\pi_i([t_1],[t_2]) = [t_i]$.

The fact that $\sim$ is a congruence (see Section 2.1.2) guarantees that this is well defined.

It is clear that $\Lambda/\sim$ is a rich applicative structure, because Λ is rich. On the other hand, since $\sim$ is a congruence, $\Lambda/\sim$ is extensional.

Now we only need to show for any Henkin model $\mathcal{A}$ and any environment η, there is a unique morphism from $\mathcal{F}$ to $\mathcal{A}$. In fact, for the given η the meaning function $[\![-]\!]_\eta : \Lambda \longrightarrow \mathcal{A}$ is sound (See e.g. [11]), hence it respects $\rhd$, i.e., the meaning function is a morphism from $\mathcal{F}$ to $\mathcal{A}$. The explicit definition of $[\![-]\!]_\eta$ guarantees the uniqueness. This can be displayed by using induction on term structures. (See e.g. [33].) ⊣

Example 9 For any set B, T_B, the full type hierarchy is a Henkin model; in particular, Friedman [11] showed that for any infinite set B, $\beta\eta$-reductions are complete for T_B. This says (for a lambda calculus with single atomic type) an equation $\Gamma \rhd s = t$ is provable if and only if it is true in T_B (for infinite set B). In general, if there are several atomic types, and we assign a set, say A_i to each of them respectively, then the smallest full subccc in *Set* containing all the T_{A_i} is the full type hierarchy model.

Remark 9 A typed applicative structure is an algebraic structure. In fact, we could define it as a multisorted universal algebra. As mentioned earlier, the requirement rich can also be stated equationally (since all the combinators K, S, etc. are equationally presented). However extensionality is not an equational property, and is not closed under, say, homomorphisms. So, Henkin models do not form an algebraic variety. In fact, they should be regarded as models of the "first-order theory of typed λ-calculus". For a detailed discussion, see [36].

Remark 10 Typed λ-theories and abstract ccc's are so alike that it is hard to say the latter are semantic models of the former. On the other hand, Henkin Models are set-theoretic models, so they correspond more closely to our idea of "semantics". However, the completeness theorem fails for them (See [34], [36]). This suggests that Henkin Models might be a little bit strong.

Remark 11 We have seen that a meaning function is a functor respecting the ccc structure. The faithfulness and the fullness of this functor corresponds to two kinds of completeness properties of the theory: ***provability*** and ***definability.***

Just like fullness, faithfulness is a kind of completeness theorem. The completeness result of Friedman [11] is indeed a faithful representation theorem for certain cartesian closed categories. (See [11], [7].)

The study of provability occurs almost everywhere in logic. For example, in our context, in Lambek-Scott [24] it is shown how proofs in intuitionistic $\{\wedge, \rightarrow, \top\}$-logic correspond to arrows in ccc's. Thus *provability* of $A \vdash B$ would correspond to *non-emptiness* of the appropriate Hom-set $\mathrm{Hom}(A, B)$.

The less familiar concept *definability* is given below.

Definition 4 A functional f in a Henkin model $\mathcal{A}$ is called *definable*, if there is a closed term t such that $f = [\![t]\!]$. It is called *definable from* $a_1, \cdots, a_n \in \mathcal{A}$ if there is a term M with at most n variables and an environment η such that $f = [\![M]\!]_\eta a_1 \cdots a_n$.

We will come back to definability in the next chapter.

2.5 Indeterminates

Variables are essential in mathematics and logic. Usually, they bear different names in different situations, such as *parameters, indeterminates*, etc. In this section, we will focus on variables used as indeterminates, and investigate their algebraic behavior.

The first time that one met indeterminates in mathematics, probably, is in the study of algebraic expansions of rings and fields. We carry on the same usage and idea here. First we will see literally how to add indeterminate arrows to a category, then we will investigate the categorical structure of the expansion by adjoining indeterminates.

2.5.1 Linguistic Discussion

Let $\mathcal{A}$ be a Henkin model. Let $\{\phi_a | a \in \mathcal{A}\}$ be a set of names or constants denoting elements[3] of the $\mathcal{A}$. Let $\Omega \subseteq \{\phi_a | a \in \mathcal{A}\}$ be a set of constants, possibly empty.

We build a simply typed λ-calculus $\mathcal{L}(\Omega)$ by the following:

1. It has the same types as that of $\mathcal{A}$.
2. For any non-empty type symbol, there are countably many variables of that type. Furthermore, if $a \in \mathcal{A}_\sigma$, then ϕ_a is a constant of type σ denoting a.
3. The ordinary rules for term application and abstraction.

In particular, let $\varepsilon = Th(\mathcal{A})$, i.e., the set of all the true $\mathcal{L}(\Omega)$ sentences in $\mathcal{A}$ (including β, η rules). Then $\Lambda(\Omega)$, the set of all terms of the language, forms an applicative structure. Denote $\Lambda(\Omega)/\varepsilon$ ($\Lambda(\Omega)$ with nonlogical axioms ε) by $\mathcal{A}^*$. A special case is that when Ω is the empty set and $\mathcal{A}$ has the same signature Σ as Λ, this becomes the free term model $\Lambda/\sim$.

The new structure $\Lambda(\Omega)/\varepsilon$ has the following properties:

1. $\mathcal{A}^*$ is a Henkin Model, the free model of ε.
2. There is a faithful ccc representation of $\mathcal{A}$ into $\mathcal{A}^*$, $f \mapsto [\phi_f]$. See [24], [36], and [7].

In the rest of this chapter, we will investigate other aspects of $\mathcal{A}^*$.

[3] By definition, a Henkin model is a multisorted algebra $\{A_\sigma\}_{\sigma \in \mathbf{T}}$. However, when speaking of an element of $\mathcal{A}$ we really mean an element of $\bigcup \{A_\sigma\}_{\sigma \in \mathbf{T}}$.

2.5.2 Adjoining Finitely Many Indeterminates

To freely add finitely many indeterminates is a very useful extension method used in mathematics. For example, the algebraic expansion of fields, rings and so on.

Let $\mathcal{A}$ be a category, A is an object of it. The categorical construction of adding one indeterminate $x : 1 \longrightarrow A$, is the so-called polynomial category $\mathcal{A}[x : 1 \to A]$ [24], which turns out to be isomorphic to the Kleisli category $\mathcal{A}_t$ of a certain cotriple. A very similar but unfortunately different construction is the *slice category*. As mentioned in [24], as long as equalizers are not concerned, adjoining an indeterminate of type A is not the same as forming the slice category $\mathcal{A}/A$. However, Makkai discovered that $\mathcal{A}/A$ has a full subcategory $\mathcal{A}//A$ with objects $C \times A \xrightarrow{\pi_2} A$, which is isomorphic to $\mathcal{A}[x : 1 \to A]$. These two equivalent constructions are described, respectively, in detail in [24], [7] (also see the related references there).

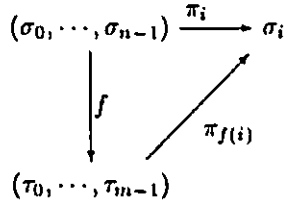
So far, we have discussed adding finitely many indeterminates $\mathcal{A}[x_1], \mathcal{A}[x_1, x_2], \cdots, \mathcal{A}[x_1, \cdots x_n]$. It is well-known ([24]) that adding finitely many indeterminates $x_i : 1 \longrightarrow A_i$, $i \leq i \leq n$, is isomorphic to adding one indeterminate of type $A_1 \times \cdots \times A_n$. But what about adding infinitely many indeterminates?

In the next section, we present a categorical construction of adding infinitely many indeterminates to a Henkin model.

2.5.3 What Is $\mathcal{A}^*$ for a Given Henkin Model $\mathcal{A}$?

Let $\mathcal{A}^*$ be obtained by adding infinitely many indeterminantes to each nonempty type of $\mathcal{A}$. In this part, we present a categorical construction for this free category. The way we present it originated from Volger's filter colimit expression for logical categories[52].

Let $\mathcal{A}$ be a well-pointed ccc with no empty types, i.e., for any object σ, there is a global arrow $t : 1 \longrightarrow \sigma$. Denote $\mathcal{A}_0$ by $\mathcal{T}$, the set of types. Let $\mathcal{T}^*$ be the category with objects all finite sequences of elements from $\mathcal{T}$, maps are induced from functions between natural numbers as follows. Let $f : n \longrightarrow m$ be a map between natural numbers, where we identify natural number n with the set $\{0, \cdots, n-1\}$. f induces a map between objects of $\mathcal{T}^*$, also denoted f, where $f : (\sigma_0, \cdots, \sigma_{n-1}) \longrightarrow (\tau_0, \cdots, \tau_{m-1})$ satisfies



So the codomain is an "extension" of the domain, and the arrow is a reindex.

Example 10 Let $f : 5 \longrightarrow 6$ be defined as $f(0) = 0$, and for $n \geq 1$ $f(n) = n - 1$. Then f induces a map from $(\sigma_0, \sigma_0, \sigma_1, \sigma_2, \sigma_3)$ to $(\sigma_0, \sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5)$.

This construction is very important for building a filtered category. (For the definition of filtered category, we suggest the reader consult [30].)

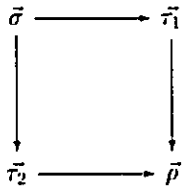
Lemma 1 *The category $\mathcal{T}^*$ is filtered.*

The following can be easily proved. For example for the first case, the vector $\vec{\rho}$ can be chosen as a kind of concatenation. We leave the proof to interested readers.

1. Every diagram of the form

$$\vec{\tau_2} \longleftarrow \vec{\sigma} \longrightarrow \vec{\tau_1}$$

has a commutative completion.



2. For every diagram

$$\vec{\sigma} \overset{f}{\underset{g}{\rightrightarrows}} \vec{\tau}$$

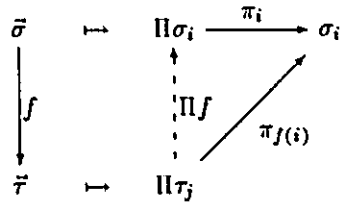
there is a morphism $h : \vec{\tau} \longrightarrow \vec{\rho}$, such that $h \circ f = h \circ g$.

3. For any two sequences $\vec{\tau}, \vec{\sigma}$, there is another sequence $\vec{\rho}$, such that

$$\vec{\tau} \longrightarrow \vec{\rho} \longleftarrow \vec{\sigma}$$

4. $\mathcal{T}^*$ has an initial object, i.e. the empty sequence.

A contravariant functor $\Pi : \mathcal{T}^* \longrightarrow \mathcal{A}^{op}$ is defined by



Using the universal property of the product, Πf exists. In particular, $\Pi\emptyset$ is the terminator of $\mathcal{A}$. For any objects σ, τ in $\mathcal{A}$, a set-valued functor

$$T_{\sigma\tau} : \mathcal{T}^* \longrightarrow \mathcal{S}et$$

is defined by

$$\begin{aligned}(\sigma_1, \cdots, \sigma_n) &\mapsto \mathcal{A}(\Pi\sigma_i \times \sigma, \tau) \\ T(f) &:= \lambda x^{\Pi\sigma_i \times \sigma \to \tau}.x \circ (\Pi(f) \times 1_\sigma)\end{aligned}$$

Finally, the category $\mathcal{A}^*$ is defined as

$$\mathrm{ob}(\mathcal{A}^*) = \mathrm{ob}(\mathcal{A}) \qquad\qquad \mathcal{A}^*(\sigma, \tau) = \varinjlim T_{\sigma\tau}$$

where

$$\varinjlim T_{\sigma\tau} = \bigsqcup T(\vec{\sigma})/\sim$$

In $\mathcal{S}et$,

$$\bigsqcup T(\vec{\sigma})/\sim = \{(f, \vec{\sigma}) | f : \Pi\sigma_i \times \sigma \longrightarrow \tau\}$$

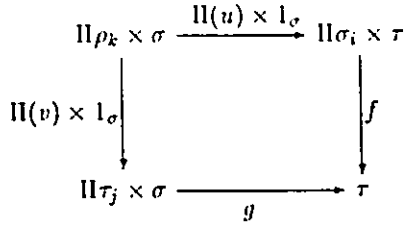
and $(f, \vec{\sigma}) \sim (g, \vec{\tau})$ iff there exists

$$\vec{\tau} \xrightarrow{u} \vec{\rho} \xleftarrow{v} \vec{\sigma}$$

such that

$$f \circ (\Pi(u) \times 1_\sigma) = g \circ (\Pi(v) \times 1_\sigma),$$

Equivalently, the following diagram commutes



Roughly speaking, $(f, \vec{\sigma}) \sim (g, \vec{\tau})$ if f, g can be identified in a later step of σ, τ.

Note that the equivalence class $[(f, \vec{\sigma})]_\sim : \sigma \longrightarrow \tau$ is an arrow in $\mathcal{A}^*$. For a pair of arrows

$$[(f, \vec{\sigma})]_\sim : \sigma \rightarrow \tau, \qquad [(g, \vec{\tau})]_\sim : \tau \longrightarrow \rho$$

the composition is defined to be $[(g \circ (\Pi\vec{\tau} \times f), \vec{\tau}\vec{\sigma})]_\sim$, which is a map of $\sigma \longrightarrow \rho$ in $\mathcal{A}^*$.

Remark 12 From now on, we use representatives of equivalence classess to denote arrows.

Theorem 3 *$\mathcal{A}^*$ is a well-pointed ccc. Furthermore, $J : \mathcal{A} \longrightarrow \mathcal{A}^*$ is a faithful representation, where the functor J maps each $f : \sigma \rightarrow \tau$ to $(f, \vec{\emptyset})$, where $\vec{\emptyset}$ is the empty sequence.*

The functor J maps each $f : \sigma \rightarrow \tau$ to $(f, \vec{\emptyset})$. For any nonempty $\vec{\sigma}$, we think of $(f, \vec{\sigma})$ as a polynomial in variables $\vec{x} : \vec{\sigma}$.

What are the indeterminantes in $\mathcal{A}^*$? The indeterminate $x_i : 1 \longrightarrow \sigma$ is represented by some arrow

$$\pi_i^n : \sigma^n \times 1 \longrightarrow \sigma$$

where $i \leq n$. In particular, we can show that for $i, j \leq n, m$,

$$(\pi_i^n : \sigma^n \times 1 \longrightarrow \sigma) \sim (\pi_i^m : \sigma^m \times 1 \longrightarrow \sigma)$$

and

$$(\pi_i^n : \sigma^n \times 1 \longrightarrow \sigma) \not\sim (\pi_j^m : \sigma^m \times 1 \longrightarrow \sigma)$$

if $i \neq j$.

Fact: Volger shows that this construction of $\mathcal{A}^*$ satisfies the universal property for adding indeterminates to $\mathcal{A}$ See [52], page 65.

This finishes our presentation of λ-calculi and categories. The study of λ-calculi has a longer history than that of categories. For an exhaustive discussion of the history of both see [5], [24] and [30].

Chapter 3

Logical Predicates

Homomorphisms in algebra play a very important role. Semantic models of λ-calculi are like algebras, with additional structure necessary to interpret "logical" properties such as "extensionality". So we expect a kind of "logical" homomorphism to arise in the study of typed λ-calculus. These are called *logical relations* and are similar to "homomorphic relations" that arise in other areas of algebra ([23]). Logical relations play an important role in the studies of proof theory and theoretical computer science. Logical relations for typed λ-calculi were developed by Howard, Tait, Friedman, Läuchli, Plotkin, Statman, Mitchell and others. See [33].

In this Chapter, we will introduce logical relations on applicative structures, and then on Henkin models. At the same time, some related properties are presented. Most of these results are from other people's work; we cite the appropriate references there. There are many interesting properties of logical relations which we will not go into. Among other things, logical relations have application to full abstraction [1], [37], [48], realizability semantics of constructive logics and strictness analysis[1].

A logical relation $\mathcal{R}$ is a family $\{R^\tau\}$ of type-indexed relations on applicative structures. The relation $R^{\sigma\to\tau}$ of type $\sigma \to \tau$ is determined from the relations R^σ and R^τ in a way that guarantees closure under application and lambda abstraction. The relation $R^{\sigma\times\tau}$ for type $\sigma \times \tau$ is similarly determined from the relations R^σ and R^τ in a way that guarantees closure under pairing and projection. The notion is so logical that such relations are closed under existential quantifications, infinite conjunctions, and infinite disjunctions when they are suitably defined [47]. On the other hand, the notion is not quite algebraic. For example, the composition of two logical relations might fail to be logical (See **Example 14**). In this thesis we will formulate the basic definition by using applicative structures.

3.1 Logical Relations on Applicative Structures

As we have said in the second chapter, applicative structures are a kind of multi-sorted algebra [6]. For algebraic structures, the notion of homomorphism is probably one of the most important concepts. In this section, we will first introduce logical relations on applicative structures, and then compare some obvious differences between these two concepts. The main references are Mitchell [33] and Statman [46].

Let $\mathcal{A} = (A^\tau, \mathrm{ap}^{\sigma,\tau})$, $\mathcal{B} = (B^\tau, \mathrm{ap}^{\sigma,\tau})$ be applicative structures for some algebraic signature Σ.

Definition 1

A (binary) *logical relation* $\mathcal{R} = \{R^\tau \mid \tau \in T\}$ over $\mathcal{A}$ and $\mathcal{B}$ is a family of (binary) relations indexed by the type expressions over Σ and satisfying

1. $R^\tau \subseteq A^\tau \times B^\tau$ for each type τ
2. $R^{\sigma\to\tau}(f,g)$ iff $\forall (x,y) \in A^\sigma \times B^\sigma . R^\sigma(x,y) \Rightarrow R^\tau(\mathrm{ap}(f,x), \mathrm{ap}(g,y))$

3. $R^{\sigma\times\tau}(s,t)$ iff $R^{\tau}(\pi_1 s, \pi_1 t), R^{\sigma}(\pi_2 s, \pi_2 t)$

If there are constants, we require they are related, i.e., $R^{\tau}(c,c)$ for any constant c of type τ.

In the same way, for any $n \geq 1$, we can define an n-ary logical relation $\mathcal{R}$ over applicative structures $\mathcal{A}_1, \cdots, \mathcal{A}_n$. For $n > 1$, we usually call them *logical relations*, while for $n = 1$, we usually call them *logical predicates*. In order to focus on logical relations themselves, we only deal with applicative structures without empty types.

Remark on notation: We use ordinary mathematical notation $f(x)$ to replace $\mathbf{ap}(f,x)$ if there is no confusion.

The most interesting part, of course, is the case of function types. The definition says: for any two functionals $f, g : \sigma \to \tau$, where $f \in \mathcal{A}^{\sigma\to\tau}$, $g \in \mathcal{B}^{\sigma\to\tau}$,

$$f R^{\sigma\to\tau} g \text{ iff } \forall x \in A^{\sigma}, y \in B^{\sigma}\ x R^{\sigma} y \Rightarrow f(x) R^{\tau} g(y)$$

The central property is that *two functions are logically related if and only if they map related arguments to related results.* $R^{\sigma\to\tau}$ is determined uniquely by R^{τ} and R^{σ}.

In particular, if R^{τ} is the graph of a function for each τ, R then is called a *logical function*. In that case, for each τ, we write $R^{\tau}(x) = y$ in place of $x R^{\tau} y$; and the above formula becomes

$$\forall x \in A^{\sigma}\ R^{\tau}(f(x)) = (R^{\sigma\to\tau}(f))(R^{\sigma}(x))$$

or simply $R^{\tau}(f(x)) = (R^{\sigma\to\tau}(f))(R^{\sigma}(x))$. So *logical functions are algebraic homomorphisms with respect to applications and types.* In general the converse is not true; we will see this later. (See the proposition below.)

Example 11 Let $\mathcal{R} \subseteq \mathcal{A}\times\mathcal{A}$ be the family of identity relations, i.e., for any type symbol σ and any elements $x, y \in A^{\sigma}$, $R^{\sigma}(x,y)$ means $x = y$. Now let us suppose R is logical. In that case

$$\begin{array}{rll} R^{\sigma\to\tau}(f,g) & \text{iff} & \forall x, y \in A^{\sigma}. R^{\sigma}(x,y) \Rightarrow R^{\tau}(f(x), g(y)) \\ & \text{iff} & \forall x \in A^{\sigma}. R^{\tau}(f(x), g(x)) \end{array}$$

This says:

$$f = g \qquad \text{iff} \qquad \forall x \in A^{\sigma}(f(x) = g(x))$$

where $f(x), g(x) \in A^{\tau}$. The last formula says $\mathcal{A}$ is extensional [33]. Thus we have proved the following:

Proposition 3 *An applicative structure is extensional if and only if the identity relation is logical.*

This shows one difference between logical relations and homomorphisms: the identity function is always a homomorphism (in fact, an isomorphism); whereas for relations the identity relation is logical only for *extensional* structures.

A notion we will mention very often is the following

Definition 5 Let $\mathcal{R}$ be a binary logical relation on Henkin models $\mathcal{A}, \mathcal{B}$. We say that *environments η_A for $\mathcal{A}$ and η_B for $\mathcal{B}$ (satisfying some context Γ) are related by $\mathcal{R}$,* and write $\mathcal{R}^{\Gamma}(\eta_A, \eta_B)$, if

$$R^{\sigma}(\eta_A(x), \eta_B(x))$$

for every $x : \sigma \in \Gamma$. In case of no confusion, we will simply write $\mathcal{R}(\eta_A, \eta_B)$.

Adjoining indeterminates to a category satisfies a familiar universal property (see Chapter 2 section 5). A similar situation arises for logical relations. For example, we have the following property.

Proposition 4 *Let $\mathcal{R}$ be a binary relation on $\mathcal{A}$, it can be extended to $\mathcal{R}^*$ on $\mathcal{A}^*$ by defining for any $M, N \in \mathcal{A}^\tau$ for any $\tau \in T$*

$$\mathcal{R}^*(M,N) \text{ iff for any environments } \mu,\nu, \mathcal{R}(\mu,\nu) \text{ implies } \mathcal{R}([\![M]\!]_\mu, [\![N]\!]_\nu) \tag{3.1}$$

Then $\mathcal{R}^*$ is logical if $\mathcal{R}$ is.

Proof Let us look at function types. We want to show that

$$\mathcal{R}^*(M,N) \text{ iff } \forall t,s\ \mathcal{R}^*(t,s) \Rightarrow \mathcal{R}^*(Mt,Ns)$$

First assume $\mathcal{R}^*(M,N)$ and $\mathcal{R}^*(t,s)$, we show that $\mathcal{R}^*(Mt,Ns)$.

It is sufficient to show

$$\forall \mu,\nu\ \ \mathcal{R}(\mu,\nu) \qquad \text{implies} \qquad \mathcal{R}([\![Mt]\!]_\mu, [\![Ns]\!]_\nu)$$

Let $\mathcal{R}(\mu,\nu)$, then from the assumption above we have

$$\mathcal{R}([\![M]\!]_\mu, [\![N]\!]_\nu) \text{ and } \mathcal{R}([\![t]\!]_\mu, [\![s]\!]_\nu) \ .$$

Since $\mathcal{R}$ is logical, these two give

$$\mathcal{R}([\![M]\!]_\mu[\![t]\!]_\mu, [\![N]\!]_\nu[\![s]\!]_\nu) \ .$$

Thus we have $\mathcal{R}([\![Mt]\!]_\mu, [\![Ns]\!]_\nu)$.

Now, let us assume that for any $t,s\ \ \mathcal{R}^*(t,s) \Rightarrow \mathcal{R}^*(Mt,Ns)$, and show $\mathcal{R}^*(M,N)$. We must show that for any environments μ,ν, if $\mathcal{R}(\mu,\nu)$, then $\mathcal{R}([\![M]\!]_\mu, [\![N]\!]_\nu)$, i.e., if $\mathcal{R}(\mu,\nu)$ then we want to show

$$\forall a,b\ \ \mathcal{R}(a,b) \Rightarrow \mathcal{R}([\![M]\!]_\mu a, [\![N]\!]_\nu b)$$

For any given a,b, let us choose a new variable x, which does not occur in M,N. Expand the environments μ,ν to μ',ν' such that

$$\mu'(y) = \begin{cases} \mu(y) & \text{if } y \neq x \\ a & y = x \end{cases}$$

$$\nu'(y) = \begin{cases} \nu(y) & \text{if } y \neq x \\ b & y = x \end{cases}$$

then $\mathcal{R}(\mu'(x),\nu'(x))$. Furthermore, we have $\mathcal{R}(\mu',\nu')$ because of $\mathcal{R}(\mu,\nu)$ and $\mathcal{R}(a,b)$.

From $\mathcal{R}^*(x,x)$ (directly from the definition of $\mathcal{R}^*$), and the first assumption, we get $\mathcal{R}^*(Mx,Nx)$. Hence, for any $\mathcal{R}$-related environments μ_1,ν_1, $\mathcal{R}([\![Mx]\!]_{\mu_1}, [\![Nx]\!]_{\nu_1})$. In particular, we have $\mathcal{R}([\![Mx]\!]_{\mu'}, [\![Nx]\!]_{\nu'})$, or $\mathcal{R}([\![M]\!]_\mu a, [\![N]\!]_\nu b)$. This finishes our proof.

Product types are easy and left to the reader. ⊣

Like most of the theorems on logical relations, this one was first proved by Statman [46], who appears to be the first person to treat logical relations as a systematic object of study.

Remark 13 A logical relation is uniquely determined by its behavior on the base types. When the signature has no term constants, one may easily construct logical relations by choosing arbitrary $R^o \subseteq A^o \times B^o$ for each atomic type constant o, and extend it to product and exponential types inductively. For example, if R^0 is symmetric and transitive then for any τ, R^τ is symmetric and transitive. However, reflexivity is not hereditary. For example, let $R^0 = I \cup \{(2,3),(3,2)\}$ on T_ω, then for any function $f \in \omega^\omega, fRf$ implies that $(f(2),f(3)) \in R^0$, hence either $f(2) = f(3)$ or $\{f(2),f(3)\} = \{2,3\}$. So lots of functions are not R related to themselves. For instance, $\lambda x.2x$ is one such.

For signatures with term constants, in particular, higher order term constants, it is more difficult to build logical relations. If, however, the constants are at base types, a logical relation can still be constructed inductively from the relations on the base types. Those relations on base types have to respect the interpretation of all term constants.

Applicative structures of the same signature are closed under products, so for any natural number n, an n-ary relation on $\mathcal{A}_1, \cdots, \mathcal{A}_n$ can always be thought of as a logical predicate on $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$.

Theoretically, abstraction and application are inverse to each other. Logical relations try to connect them consistently. If the applicative structures are models, this works perfectly (see next section). But for general applicative structures, it happens that some λ-abstractions might not exist, or on the other hand, there might be several candidates with the same desired behaviors. Ordinary logical relations cannot connect them well. So an additional condition is needed.

Definition 6 *(Admissible)* If $\mathcal{R}$ is a logical binary relation on $\mathcal{A} \times \mathcal{B}$, we say $\mathcal{R}$ is *admissible*, if for any related environments η_A, η_B and term $M : \tau$

$$\forall a, b \quad a\mathcal{R}^\sigma b \Rightarrow [\![M]\!]_{\eta_A(x:=a)} \mathcal{R}^\tau [\![M]\!]_{\eta_B(x:=b)}$$

implies

$$\forall a, b \quad a\mathcal{R}^\sigma b \Rightarrow ([\![\lambda x.M]\!]_{\eta_A} a) \mathcal{R}^\tau ([\![\lambda x.M]\!]_{\eta_B} b)$$

Remark 14 Admissiblity is a property used by Statman and Mitchell to aid in proving properties of logical relations [46], [33]. Statman used it only for lambda terms, so he formulated it as

> $\mathcal{R}$ is *admissible* if $\mathcal{R}_o$ is closed under coordinatorwise inverse reductions, i.e., if $u\mathcal{R}_o v$ and $t \overset{1}{\Rightarrow} u, s \overset{1}{\Rightarrow} v$, then $t\mathcal{R}_o s$.

This is quite like Girard's condition **CR3** in proving SN, which is :

> If t is not an I-term and for all $t \overset{1}{\Rightarrow} u$ RED(u), then RED(t).

We will come back to this connection later. See [13], and **Section 6.1**.

As we mentioned before, the well typed terms form an applicative structure

$$\Lambda = (\Lambda^\sigma, \mathrm{ap}^{\sigma,\tau})$$

As usual, for two terms $\Gamma \rhd M : \sigma \to \tau$, $\Gamma \rhd t : \sigma$, we write $\mathrm{ap}^{\sigma,\tau}(M, t)$ simply as Mt.

Let $\mathcal{R}$ be a binary logical relation defined on Λ. Define a logical predicate $\exists\mathcal{R}$ on Λ as follows: for any term M,

$$\exists\mathcal{R}(M) \text{ iff } \exists N, \mathcal{R}^\sigma(M, N)$$

Proposition 5 *$\exists\mathcal{R}$ is logical.*

Proof (only for function types) We show that

$$\exists\mathcal{R}^{\sigma\to\tau}(f) \text{ iff } \forall t\ (\exists\mathcal{R}^\sigma(t) \Rightarrow \exists\mathcal{R}^\tau(ft))$$

Suppose $\exists\mathcal{R}^{\sigma\to\tau}(f)$ and $\exists\mathcal{R}^\sigma(t)$, we prove $\exists\mathcal{R}^\tau(ft)$, i.e., $\exists u\mathcal{R}^\tau(ft, u)$. It is sufficient to show that

$$\exists u\ \ \forall \mu, \nu\ \mathcal{R}(\mu, \nu) \Rightarrow \mathcal{R}([\![ft]\!]_\mu, [\![u]\!]_\nu)$$

From the assumption $\exists\mathcal{R}^{\sigma\to\tau}(f)$ and $\exists\mathcal{R}^\sigma(t)$, there exist g, s such that

$$\mathcal{R}^\sigma(f, g),\ \mathcal{R}^\sigma(t, s)$$

Let $u = gs$, and assume environments μ, ν are $\mathcal{R}$-related, then we have

$$\mathcal{R}([\![f]\!]_\mu, [\![g]\!]_\nu),\ \mathcal{R}([\![t]\!]_\mu, [\![s]\!]_\nu)$$

Hence $\mathcal{R}([\![ft]\!]_\mu, [\![u]\!]_\nu)$. This finishes one part of the proof.

Assume that $\forall t\ \exists\mathcal{R}^\sigma(t) \Rightarrow \exists\mathcal{R}^\tau(ft)$, want to show $\exists R^{\sigma\to\tau}(f)$. To do this we need to prove there is a g such that $\mathcal{R}^*(f,g)$, i.e.,

$$\forall \mu, \nu\ \mathcal{R}(\mu,\nu) \Rightarrow R^{\sigma\to\tau}([\![f]\!]_\mu, [\![g]\!]_\nu)$$

It suffices to show that for any a, b, if $R^\sigma(a,b)$, then $\mathcal{R}^\tau([\![f]\!]_\mu a, [\![g]\!]_\nu b)$.

Notice for any variable y, we always have $\mathcal{R}^*(y,y)$. Thus $\exists\mathcal{R}^\sigma(y)$. From the assumption, we have $\exists\mathcal{R}^\tau(fy)$, i.e., there is a term u, such that $\mathcal{R}^*(fy,u)$, or $\mathcal{R}^*(fy,(\lambda y.u)y)$. Let x be a new variable, and $g = \lambda x.u$. Furthermore, define two new environments μ^+, ν^+ by

$$\begin{array}{rcll} \mu^+(y) & \equiv & \mu(y) & \text{if } y \neq x \\ & \equiv & a & \text{if } y = x \\ \nu^+(y) & \equiv & \nu(y) & \text{if } y \neq x \\ & \equiv & b & \text{if } y = x \end{array}$$

Then from $R^*(fy,(\lambda y.u)y)$ and $\mathcal{R}(\mu,\nu)$, we get $\mathcal{R}^\tau([\![fy]\!]_\mu, [\![(\lambda y.u)y]\!]_\nu)$ or

$$\mathcal{R}^\tau([\![f]\!]_\mu a, [\![g]\!]_\nu b)$$

This finishes our proof of $\exists g\mathcal{R}^*(f,g)$, and $\exists R^{\sigma\to\tau}(f)$, hence finishes the proof that $\exists\mathcal{R}$ is logical for the case of function types. (Again, product types are straightforward and left to the readers.) ⊣

Like the notion $\mathcal{R}^*$, $\exists\mathcal{R}$ can be easily introduced for ordinary Henkin models, i.e., defining them for $\Lambda[\Omega]/\varepsilon$ (see Chapter 2 Section 5). In fact, the above two properties can be proved in this context, too.

As we said before, in order to model a lambda-calculus an ordinary applicative structure should be enriched in two ways: by extensionality and functional completeness. From **Proposition 3**, we know that "*Extensionality can be characterized by saying '=' is logical*". The other condition can also be characterized by using logical relations. The following is a theorem of Statman[46]:

Theorem 4 *Let $\mathcal{A}$ be an applicative structure, then $\mathcal{A}$ is a Henkin model iff*

1. *Equality is logical, and*
2. *For any applicative structure $\mathcal{B}$ and any logical relation $\mathcal{R}$ on $\mathcal{B}\times\mathcal{A}$, $\exists\mathcal{R}$ is logical.*

3.2 The Fundamental Theorem of Logical Relations: The Basic Lemma

The fundamental theorem about logical relations is the following Basic Lemma. See [11][39][46][33]. It establishes that the meaning of a closed term in one model is always logically related to its meaning in any other model. For an open term, if the assignments of its variables in different models are related, its meanings in these models are also related. Hence, the internal properties of a term determines its meaning. On the face of it, the Basic Lemma only seems to apply to Henkin models, since the meaning of a term is not necessarily defined in arbitrary applicative structures. However, by reasonably strengthening a little on the definition of logical relations, we can also get a satisfactory version of this fundamental theorem which will be useful in proving properties of proper applicative structures.

Lemma 2 (Basic Lemma for Henkin Models) *Let $\mathcal{A}\,\mathcal{B}$ be Henkin models of the same signature, $R \subseteq \mathcal{A} \times \mathcal{B}$ a logical relation. Let η_A, η_B be environments for $\mathcal{A}$ and $\mathcal{B}$ with $\mathcal{R}^\Gamma(\eta_A, \eta_B)$. Then*

$$\mathcal{R}([\![\Gamma \rhd M : \sigma]\!]_{\eta_A}, [\![\Gamma \rhd M : \sigma]\!]_{\eta_B})$$

for every typed term $\Gamma \rhd M : \sigma$.

Proof We use induction on the construction of terms.

1. For a variable $\Gamma \rhd x : \sigma$, since $\mathcal{R}^\Gamma(\eta_A, \eta_B)$, hence $\mathcal{R}([\![\Gamma \rhd x : \sigma]\!]_{\eta_A}, [\![\Gamma \rhd x : \sigma]\!]_{\eta_B})$.

2. For an application $\Gamma \rhd MN : \tau$, we assume the theorem is true for its subterms, i.e.,

$$\mathcal{R}^{\sigma \to \tau}([\![\Gamma \rhd M : \sigma \to \tau]\!]_{\eta_A}, [\![\Gamma \rhd M : \sigma \to \tau]\!]_{\eta_B})$$
$$\mathcal{R}^{\sigma}([\![\Gamma \rhd N : \sigma]\!]_{\eta_A}, [\![\Gamma \rhd N : \sigma]\!]_{\eta_B})$$

   However, from the definition of logical relation, we get the following immediately.

$$\mathcal{R}^{\tau}(\mathbf{ap}([\![\Gamma \rhd M : \sigma \to \tau]\!]_{\eta_A}, [\![\Gamma \rhd N : \sigma]\!]_{\eta_A}), \mathbf{ap}([\![\Gamma \rhd M : \sigma \to \tau]\!]_{\eta_B}, [\![\Gamma \rhd N : \sigma]\!]_{\eta_B}))$$

   This is, from the definition of meaning function,

$$\mathcal{R}^{\tau}([\![\Gamma \rhd MN : \tau]\!]_{\eta_A}, [\![\Gamma \rhd MN : \tau]\!]_{\eta_B})$$

3. For a pair term $\Gamma \rhd (s,t) : \sigma \times \tau$, since the theorem is true for subterms, then from the definition of logical relation, we have

$$\mathcal{R}([\![\Gamma \rhd (s,t) : \sigma \times \tau]\!]_{\eta_A}, [\![\Gamma \rhd (s,t) : \sigma \times \tau]\!]_{\eta_B})$$

   immediately.

4. For an abstraction term $\Gamma \rhd \lambda x : \sigma.M : \sigma \to \tau$, we want to show that

   (*) $$\mathcal{R}([\![\Gamma \rhd \lambda x.M : \sigma \to \tau]\!]_{\eta_A}, [\![\Gamma \rhd \lambda x.M : \sigma \to \tau]\!]_{\eta_B})$$

   provided that $\mathcal{R}(\eta_A, \eta_B)$. Let $\mathcal{R}(\eta_A, \eta_B)$, then it is sufficient to show that for any $a \in A^\sigma, b \in B^\sigma$, if $\mathcal{R}(a,b)$, then

$$\mathcal{R}^{\tau}(\mathbf{ap}([\![\Gamma \rhd \lambda x.M : \sigma \to \tau]\!]_{\eta_A}, a), \mathbf{ap}([\![\Gamma \rhd \lambda x.M : \sigma \to \tau]\!]_{\eta_B}, b))$$

   Define two new environments η'_A, η'_B by the following

$$\begin{aligned} \eta'_A(y) &= \eta_A(y) \quad \text{if } y \neq x \\ &= a \quad \text{if } y = x \\ \eta'_B(y) &= \eta_B(y) \quad \text{if } y \neq x \\ &= b \quad \text{if } y = x \end{aligned}$$

   Then $\mathcal{R}(\eta_A, \eta_B)$ together with $R^\sigma(a,b)$ give $\mathcal{R}(\eta'_A, \eta'_B)$. Therefore by using the induction hypothesis, we have

$$\mathcal{R}([\![\Gamma \rhd M : \tau]\!]_{\eta'_A}, [\![\Gamma \rhd M : \tau]\!]_{\eta'_B})$$

   But, since

$$[\![\Gamma \rhd M : \tau]\!]_{\eta'_A} = \mathbf{ap}([\![\Gamma \rhd \lambda x.M : \sigma \to \tau]\!]_{\eta_A}, a)$$
$$[\![\Gamma \rhd M : \tau]\!]_{\eta'_B} = \mathbf{ap}([\![\Gamma \rhd \lambda x.M : \sigma \to \tau]\!]_{\eta_B}, b)$$

   Consequently, from the definition of logical relation, the formula (*) is proved.

⊣

An easy consequence is the following:

Corollary 1 *Let $\mathcal{A}, \mathcal{B}$ be Henkin models for the same signature, and let $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{B}$ be a logical relation. Then for any closed term M, the meaning of M in $\mathcal{A}$ is $\mathcal{R}$-related to the meaning of M in $\mathcal{B}$, i.e. $[\![M]\!]_{\mathcal{A}} \mathcal{R} [\![M]\!]_{\mathcal{B}}$.*

For admissible logical relations, we can also show the following:

Lemma 3 (Basic Lemma, general version) *Let $\mathcal{A}_1, \cdots, \mathcal{A}_n$ be applicative structures of the same signature, $\mathcal{R} \subseteq \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is an admissible logical relation. Let $\eta_1, \cdots, \eta_n$ be environments for them with $\mathcal{R}^{\Gamma}(\eta_A, \eta_B)$. Then*

$$\mathcal{R}([\![\Gamma \rhd M : \sigma]\!]_{\eta_1}, \cdots, [\![\Gamma \rhd M : \sigma]\!]_{\eta_n})$$

for every typed term $\Gamma \rhd M : \sigma$.

For a proof and further discussion see [33]. For an n-ary ($n \geq 1$) logical relation, we can always consider it as a logical predicate (i.e., unary logical relation) on the product Henkin Model (or applicative structure). For further discussion of logical predicates, see below.

If $\mathcal{R}$ is a partial logical function, the global elements[1] of $\mathcal{A}$ are in the domain of $\mathcal{R}$, and $\mathcal{R}$ maps them to the corresponding global elements of $\mathcal{B}$.

Let us now discuss logical predicates on applicative structures. For Λ, the set of all terms which is an applicative structure, the basic lemma for a logical predicate $\mathcal{R}$ is

1. If M is a closed term, then $\mathcal{R}(M)$,
2. For any term M with $\mathrm{FV}(M) \subseteq \{x_1, \cdots, x_n\}$ and any substitution[2] $\theta : V^{\tau} \longrightarrow \Lambda^{\tau}$, if $R(\theta x_i)$ for $i = 1, \cdots, n$, then $R(\theta M)$.

In the rest of this section, we will present some examples of using logical relations to study syntactical properties.

A typical (and probably the first important) example is the following [46]. Let $\mathcal{SN}(t)$ be the property saying "well-typed term t is strongly normalizable", *i.e.*, any reduction starting from t is finite, or, by using König's lemma, it is the same as saying that the reduction tree of t is finite. Then we can show that $\mathcal{SN}$ is in fact an admissible logical predicate. Then the basic lemma concludes that "every term is $\mathcal{SN}$".

Here $n = 1, \mathcal{A} = \Lambda$, and θ is the identity function. Since variables are strongly normalizable, then any term t is $\mathcal{SN}$.

More precisely, to prove strong normalization, one defines a unary relation $\mathcal{S}$ on terms of base types. Then $\mathcal{S}$ is extended to a logical relation at higher types, still denoted by $\mathcal{S}$. Finally one proves the following two steps:

(i) $\mathcal{S}$ is admissible.

(ii) $\mathcal{S}(x)$ implies $\mathcal{SN}(x)$.

From the basic lemma, it follows that every well-typed term satisfies $\mathcal{S}$ and hence from (ii), is strongly normalizing. The predicate $\mathcal{S}$ is first defined straightforwardly on the terms of base types. For example, for a well-typed term t of base type, define

$$\mathcal{S}(t) \quad \text{iff } t \text{ is strongly normalizable}$$

A related proof is Girard's proof of $\mathcal{SN}$ for simply typed λ-calculus [13], a predicate $\mathcal{S} = \mathbf{Red}$ and three rules CR1, 2, 3 are employed. The predicate **Red** is defined exactly the same way as the $\mathcal{S}$ above (at base types), and then is extended to all higher types inductively.

Girard postulates the rules

[1] If an element is definable in the pure lambda calculus, we sometimes call it a global element.

[2] A substitution $\theta : V^{\tau} \longrightarrow \Lambda^{\tau}$ maps variables to terms. The induced function, also called θ, from terms to terms is defined inductively. For a term M with $\mathrm{FV}(M) \subseteq \{x_1, \cdots, x_n\}$ θM is $M[x_1 := \theta x_1, \cdots, x_n := \theta x_n]$

(CR 1) If t is in **Red**, then t is strongly normalizable.

(CR 2) If t is in **Red**, and $t \Rightarrow t'$ [3], then t' is in **Red**, too.

(CR 3) For terms of the form

$$\pi_1 N, \qquad \pi_2 N, \qquad Mt$$

if all its one step reductions are in **Red**, so is the term itself.

We notice that CR1 is saying "**Red** implies $\mathcal{SN}$",i.e., "$\mathcal{S}$ implies $\mathcal{SN}$". CR2 is a technical lemma. CR3 is saying that the predicate **Red** is *admissible*. **Red** is proved to be logical in §6.2 of [13] and a special case of the basic lemma is given in §6.3 of the same book.

Example 12 For any $M \in \Lambda^\tau$, let $\mathrm{CR}(M)$ be the following predicate:

$$\forall P, Q \in \Lambda^\tau (M \stackrel{*}{\Rightarrow} P) \wedge (M \stackrel{*}{\Rightarrow} Q) \longrightarrow \exists t \in \Lambda^\tau . (P \stackrel{*}{\Rightarrow} t) \wedge (Q \stackrel{*}{\Rightarrow} t) \quad .$$

Then the same argument as above gives the confluence (Church-Rosser) theorem. *[46], [33].*

3.3 Logical Relations on Models

Let $\mathcal{A}$ be a Henkin Model, $\mathcal{R}$ an n-ary logical relation on $\mathcal{A}$. The element $f \in \mathcal{A}$ is said to be *invariant* under $\mathcal{R}$, if $(f, \cdots, f) \in \mathcal{R}$. The basic lemma says that all the global elements are invariant. It is interesting to compare the notion *invariant* with another notion *lambda definable*. In fact, the latter is much stronger than the former. For a given $\mathcal{R}$, in general, there are more invariant elements than definable ones. A good exercise is using the Basic Lemma to show that *definable* implies invariant.

As we said before, a Henkin model $\mathcal{A}$ can be looked at as a ccc with types as objects and functions as arrows. We define a sub-category $\mathcal{R}(\mathcal{A})$ by the following

objects:	the same as that of $\mathcal{A}$
arrows:	those invariant under $\mathcal{R}$

From the discussion above we see that $\mathcal{R}(\mathcal{A})$ is a ccc. (All global elements are invariant and the related equations are already true in $\mathcal{A}$) Is it also extensional?

Example 13 Consider T_ω and the logical predicate defined on it by $\mathcal{R}^0 = \{0, \cdots, n\}$. Then

$$R^{0 \to 0} = \{f : \omega \to \omega \mid \forall k \leq n (f(k) \leq n)\}$$

Notice that the set $R^{0 \to 0}$ is infinite whereas R^0 is finite. There are more functions than $n+1$ elements can determine: in set theory, m elements can only determine m^m functions. There are many different functions in $R^{0 \to 0}$ which agree on $\{0, \cdots, n\}$. Thus extensionality fails for this example. So in general, $\mathcal{R}(\mathcal{A})$ is not a Henkin model. We will study more about $\mathcal{R}(\mathcal{A})$ for different logical relations $\mathcal{R}$ in the next chapter.

On Henkin models, there are some easy and useful exercises for the interested reader to practice on:

Recall, a *per* (partial equivalence relation) is a symmetric and transitive relation. We shall discuss *pers* in more detail in 4.2.5 and Chapter 5.

1. The logical relation generated (on a Henkin model) from a *per*, is a *per* at each level (called a *logical per*): by this we meen that for any type symbol σ, R^σ is a *per*.

2. The logical relation generated from a permutation is a permutation at each level.

[3] It reads t is reduced to t' in one step, we have this in Chapter 2 and will discuss this more in Chapter 6.

3. The logical relation generated from a surjective partial function is partial surjective at each level.

We will come back to these interesting examples to see their categorical meaning in Chapter 4.

Let us say a few more words about logical *pers*. A logical *per* has many of the properties of a congruence relation. Since a typed applicative structure is a multi-sorted algebra, a congruence relation on it allows us to form the quotient algebra. In our case, this quotient algebra is furthermore extensional. This method to get extensional structures is usually called *the extensional collapse*. The quotient of an applicative structure $\mathcal{A}$ with respect to a logical *per* $\mathcal{R}$, $\mathcal{A}/\mathcal{R}$, has the same objects as $\mathcal{A}$ and arrows are the equivalence classes of arrows of $\mathcal{A}$. Since $\mathcal{R}$ is partial, if $a \notin \mathrm{Dom}(\mathcal{R}) = \{x | x\mathcal{R}x\}$, there would be no representative of a in $\mathcal{A}/\mathcal{R}$. As is the case in algebra, $\mathcal{A}/\mathcal{R}$ is always a "model" if $\mathcal{A}$ is[4]. In fact, for any applicative structure $\mathcal{A}$ and any logical *per* $\mathcal{A}/\mathcal{R}$ is always extensional. In particular, an equivalence relation can be defined on the hom-sets of a ccc $\mathcal{E}$ by, for all $f, g : A \longrightarrow B$,

$$f \sim g \text{ iff for any } x : 1 \longrightarrow A \ \ (f \circ x = g \circ x)$$

Then $\mathcal{E}/\sim$ is a well-pointed ccc satisfying all the equations of $\mathrm{Th}(\mathcal{E})$.

Unfortunately, for some categories $\mathcal{E}$, this quotient model might be the trivial one, i.e., all the arrows of the same type may be equal. For example, if we apply this collapse to the free ccc $\mathcal{C}$ (the closed term "model", without constants), then everything collapses.

One of the applications is the following result [45].

Let $\mathcal{A}, \mathcal{B}$ be two Henkin models, h a surjective homomorphism from $\mathcal{B}$ to $\mathcal{A}$. The fundamental homomorphism theorem of algebra might suggest the following

$$\mathrm{dom}(h)/\mathrm{K}(h) \cong \mathcal{A}$$

Here $\mathrm{K}(h) = \{(x, y) \mid h(x) = h(y)\}$ is the kernel of the homomophism h. Unfortunately, this is not true here since Henkin models do not form an algebraic variety. However, the following is a very close alternative:

Theorem 5 (Statman) *For any Henkin model $\mathcal{A}$, there is a full type hierarchy T_B and a partial homomorphism $h : T_B \longrightarrow \mathcal{A}$ such that $dom(h)/\sim \cong \mathcal{A}$, where $\sim$ is the equality relation[5] defined on $dom(h)$.*

The following example shows that the composition of two logical relations might fail to be logical. This distinguishes logical relations from algebraic homomorphisms.

Example 14 Let $D = \{0, 1\}$, and T_D be the full type hierarchy generated from D. Two binary relations $R = \{\langle 0, 0\rangle\}, S = \{\langle 1, 0\rangle\}$ are on D. They can be extended inductively to higher levels to give two binary logical relations $\mathcal{R}$, $\mathcal{S}$ on T_D. To see if the composition of $\mathcal{R}$ and $\mathcal{S}$ is logical we have to show that for any type τ, $(R \circ S)^\tau = R^\tau \circ R^\tau$. (See the Remark below).
To make the notation simple, we identify a function with its image. For example, the function f which is defined on $\{0, 1\}$ and maps 0 to a, 1 to b is denoted as $\langle a, b\rangle$. Note

$$R \circ S = \emptyset$$

Let τ be $0 \to 0$, then

$$(R \circ S)^\tau = D^D \times D^D$$

i.e., each pair of elements is related. While

$$R^\tau = \{\langle\langle 0,0\rangle, \langle 0,0\rangle\rangle, \langle\langle 0,0\rangle, \langle 0,1\rangle\rangle, \langle\langle 0,1\rangle, \langle 0,0\rangle\rangle\langle\langle 0,1\rangle, \langle 0,1\rangle\rangle\}$$

and

[4] For example, if $\mathcal{A}$ is a category, cartesian, cartesian closed, etc; then so is $\mathcal{A}/\mathcal{R}$.

[5] Since h is partial, $dom(h)$ might only be a subset of T_B^0, hence the equality relation on $dom(h)$ is a *per* on T_B^0.

$$S^\tau = \{\langle\langle 0,1\rangle,\langle 0,0\rangle\rangle, \langle\langle 0,1\rangle,\langle 0,1\rangle\rangle, \langle\langle 1,1\rangle,\langle 0,0\rangle\rangle, \langle\langle 1,1\rangle,\langle 0,1\rangle\rangle\}$$

Their composition

$$R^\tau \circ S^\tau \neq (R \circ S)^\tau = D^D \times D^D$$

For example, $\langle\langle 0,0\rangle,\langle 1,1\rangle\rangle \notin R^\tau \circ S^\tau$.

Remark 15 In general, let R, S be binary logical relations on some Henkin model. Suppose the base types are enumerated by τ_i for some index i. Then the compositions $R^{\tau_i} \circ S^{\tau_i}$ define a relation on the base types, one for each τ_i. A logical relation can be built from them, in fact, this is $(R \circ S)^\tau$, for any higher type τ. If the family $\{(R^\tau \circ S^\tau)\}_\tau$ is also logical, it must be the case that $(R \circ S)^\tau = R^\tau \circ S^\tau$ since they start from the same base relations.

It would be very interesting if one could find that there are some internal conditions implying

$$(R \circ S)^\tau = R^\tau \circ S^\tau$$

i.e., exchanging the logical operation $(\_)^\tau$ with the algebraic operation $\circ$ without changing their attributes. This seems to be an open question.

Chapter 4

Relation Categories

4.1 Relation Categories and Scones

We will use $\mathcal{C}$ for a free cartesian closed category (ccc) generated from a non-empty graph, with or without non-logical axioms; and use $\mathcal{E}$ for an arbitrary ccc or even an ordinary category.

4.1.1 Scones

One of the powerful categorical constructions is the Freyd cover [24], (also called *scone* [10]). More generally, the construction arises from general comma categories [30]. Let $\mathcal{E}, \mathcal{B}, \mathcal{D}$ be categories, F, G functors as shown:

$$\mathcal{E} \xrightarrow{\quad G \quad} \mathcal{B} \xleftarrow{\quad F \quad} \mathcal{D}$$

The construction provides a new category $F \downarrow G$ with objects of the form

$$F(X) \xrightarrow{\quad f \quad} G(A)$$

where X is from $\mathcal{D}$, A is from $\mathcal{E}$, and f a morphism in $\mathcal{B}$. A morphism from $F(X) \xrightarrow{f} G(A)$ to $F(X') \xrightarrow{f'} G(A')$ is a pair (u, v) such that the following diagram commutes in $\mathcal{B}$

$$\begin{array}{ccc} F(X) & \xrightarrow{F(u)} & F(X') \\ \Big\downarrow{\scriptstyle f} & & \Big\downarrow{\scriptstyle f'} \\ G(A) & \xrightarrow[G(v)]{} & G(A') \end{array}$$

Let $\mathcal{E}$ be a category with a terminal object. The Freyd cover of the category $\mathcal{E}$, denoted $\hat{\mathcal{E}}$, is the following special case: Γ is taken to be the global sections functor (see below), I the identity functor.

$$\mathcal{E} \xrightarrow{\quad \Gamma \quad} \mathcal{S}et \xleftarrow{\quad I \quad} \mathcal{S}et$$

The objects, hence, are of the form $(X, f, \Gamma(A))$ or simply (X, f, A), with $X \in \mathcal{S}et$, A an object of $\mathcal{E}$, and $f : X \longrightarrow \Gamma(A)$. The global section functor $\Gamma : \mathcal{E} \longrightarrow \mathcal{S}et$ maps an object A to $\mathcal{E}(1, A)$, the set of global elements of $\mathcal{A}$. There is a forgetful functor.

$$G : \hat{\mathcal{E}} \longrightarrow \mathcal{E}$$

given by

$$\begin{array}{c} (X \to \Gamma(A)) \mapsto A \\ (u, v) \mapsto v \end{array}$$

For the following results, see [24], [35].

Proposition 6

1. *Γ is left exact, in particular, it preserves finite products.*
2. *If $\mathcal{E}$ is cartesian closed, then $\hat{\mathcal{E}}$ is cartesian closed and the cartesian closed structure is given as follows. Let $U = (X, f, A)$, $V = (X', f', A')$.*
   (a) *The product is*
   $$U \times V = (X \times X', f \times f', A \times A')$$
   here $(f \times f')(a, b) = \langle f(a), f'(b)\rangle$, where $a \in X, b \in X', f(a) \in \Gamma(A), f'(b) \in \Gamma(A')$.
   (b) *Let $A \Rightarrow A'$ be the exponential in $\mathcal{E}$, then the exponential inside the scone is*
   $$U \Rightarrow V = (Z, h, A \Rightarrow A')$$
   where Z is the set of morphisms $U \longrightarrow V$ in $\hat{\mathcal{E}}$, i.e., the set of all the pairs (u, v) such that the following diagram commutes.

$$\begin{array}{ccc} X & \xrightarrow{u} & X' \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f'} \\ \Gamma(A) & \xrightarrow[\Gamma(v)]{} & \Gamma(A') \end{array}$$

   The function $h : Z \longrightarrow \Gamma(A \Rightarrow A')$ maps (u, v) to $\bar{v}$, where $\bar{v}$ is obtained from v by cartesian closedness:
   $$\frac{v : A \longrightarrow A'}{\bar{v} : 1 \longrightarrow (A \Rightarrow A')}$$
3. *The canonical forgetful functor $G : \hat{\mathcal{E}} \longrightarrow \mathcal{E}$ is a ccc-representation.*

More generally, the functor Γ in the Freyd cover can be chosen as any product preserving one. The following can be shown [35]

Proposition 7 1. *If Γ is a product preserving functor, then the scone is a ccc.*

2. *The forgetful functor $G : \hat{\mathcal{E}} \longrightarrow \mathcal{E}$ preserves cartesian closed structure.*

If $\mathcal{A}, \mathcal{B}$ are cartesian closed, then $\mathcal{A} \times \mathcal{B}$ is still cartesian closed. Its objects and arrows are pairs with the first components from $\mathcal{A}$ and the second from $\mathcal{B}$. All the operations and constructions are done componentwise. In particular, for the object (A, B) of $\mathcal{A} \times \mathcal{B}$, its global sections arise from the product of those of A and those of B, *i.e.*,
$$\Gamma(A, B) = \Gamma(A) \times \Gamma(B) \cong \Gamma(A \times B)$$

For Henkin models $\mathcal{A}, \mathcal{B}$, the scone $\mathcal{A} \hat{\times} \mathcal{B}$ is the category with objects and arrows satisfying the following diagram

$$\begin{array}{ccc} R & \xrightarrow{t} & R' \\ {\scriptstyle u}\downarrow & & \downarrow{\scriptstyle u'} \\ \Gamma A \times \Gamma B & \xrightarrow[\Gamma(f,g)]{} & \Gamma A' \times \Gamma B' \end{array}$$

The full subcategory, $\mathcal{A} \tilde{\times} \mathcal{B}$ of $\mathcal{A} \hat{\times} \mathcal{B}$, consisting of those objects for which u is injective is called *the injective scone* [35]. Hence the objects of $\mathcal{A} \tilde{\times} \mathcal{B}$ look like: $(R, A \times B)$ with R a binary relation, i.e., a subset of $\Gamma A \times \Gamma B$.

It so happens that this injective scone is a sub-ccc; in fact, for any two objects $(R, A \times B), (R', A' \times B')$ products and exponentials are given by

$$(R, A \times B) \times (R', A' \times B') = (R \times R', (A \times A') \times (B \times B'))$$
$$(R, A \times B) \Rightarrow (R', A' \times B') = (R \Rightarrow R', (A \Rightarrow A') \times (B \Rightarrow B'))$$

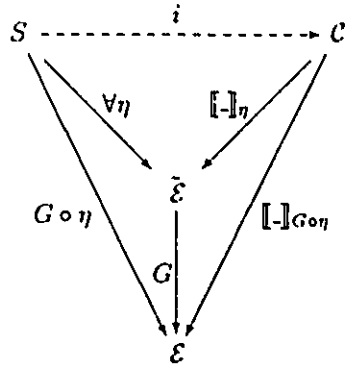
where the relations $R \times R'$ and $R \Rightarrow R'$ satisfy the following critical properties:

$$\begin{array}{rll} (a,a')(R \times R')(b,b') & \text{iff} & aRb \text{ and } a'R'b' \\ f(R \Rightarrow R')g & \text{iff} & \forall t : 1 \longrightarrow A, \forall s : 1 \longrightarrow B, tRs \text{ implies } f(t)R'g(s) \end{array}$$

Comparing this with **Definition 1** of Chapter 3, one will see immediately that *logical relations can be modeled by using injective scones.*

In this framework, the Basic Lemma (Lemma 2 in Chapter 3) translates to saying that a certain diagram commutes [35]:

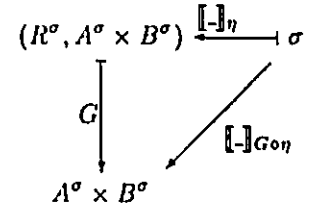
Proposition 8 (Basic Lemma) *Let $\mathcal{E}$ be the product of two Henkin models $\mathcal{A}$ and $\mathcal{B}$, $\tilde{\mathcal{E}}$ the injective scone, then the lower right triangle below commutes.*



where S is a discrete graph, $\mathcal{C}$ is the free ccc generated from S, $\eta = (\eta_A, \eta_B)$ is any type assignment, G is the forgetful functor $(R, A) \mapsto A$, $[\![-]\!]_\eta$ are ccc representations.

Before we prove it, let us explain why this is the basic lemma.

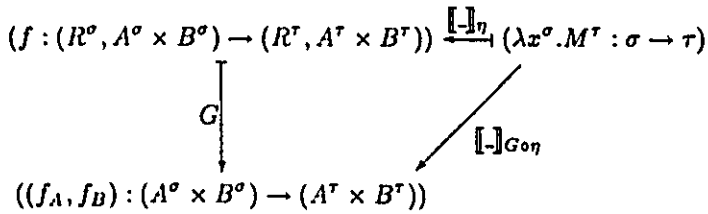
We think of $\mathcal{C}$ as syntactically presented from types and terms. Consider the above mentioned "*lower right triangle*"; for any type symbols σ, τ, we obtain the following diagram:



The above diagram is about the type part. What is a diagram for the term (or "arrow") part?

Recall the map $f : (R^\sigma, A^\sigma \times B^\sigma) \longrightarrow (R^\tau, A^\tau \times B^\tau))$ is of a form (u, v), where $u : R^\sigma \longrightarrow R^\tau$ in *Set*, and $v : (A^\sigma \times B^\sigma) \longrightarrow (A^\tau \times B^\tau)$ in the category $\mathcal{A} \times \mathcal{B}$. Hence, v is a pair itself, say, (f_A, f_B). Thus under the forgetful functor G, $f \mapsto (f_A, f_B)$.

So for the element part, we have the following:



Notice, for an injective scone, the arrow pair (u, v) can really be replaced by v. So basically, $f = (f_A, f_B)$ with $f_A : A^\sigma \longrightarrow A^\tau$, and $f_B : B^\sigma \longrightarrow B^\tau$. The proposition claims the "*lower right triangle*" commutes, with G the forgetful functor. This implies that $f_A = [\![\lambda x^\sigma.M^\tau]\!]_{\eta_A}$ and $f_B = [\![\lambda y^\sigma.M^\tau]\!]_{\eta_B}$. Notice $f \in R^{\sigma\to\tau}$ means $f_A R^{\sigma\to\tau} f_B$ or $[\![\lambda x^\sigma.M]\!]_{\eta_A}$ is related with $[\![\lambda x^\sigma.M]\!]_{\eta_B}$. This is exactly what the Basic Lemma says.

Proof Sketch:

1. The arrow $[\![\_]\!]_\eta$ is the unique functor to make the upper triangle commute from the universal property of free ccc. It is a ccc-representation.
2. The arrow $[\![\_]\!]_{G\circ\eta}$ is the unique functor to make the outside triangle commute based on the same reason as above. It is also a ccc-representation.
3. The canonical forgetful functor $G : \tilde{\mathcal{E}} \longrightarrow \mathcal{E}$ is another ccc-representation.
4. The composition of two ccc-representations G with $[\![\_]\!]_\eta$ is also a ccc-representation.

Hence from the uniqueness of $[\![\_]\!]_{G\circ\eta}$, the "*lower right triangle*" commutes. ⊣

4.2 Logical Relation Categories

In the rest of this chapter, we will assume that $\mathcal{E}$ is a cartesian closed category unless otherwise specified.

4.2.1 Introduction

Definition 7 A logical relation category $\mathcal{R}el_n(\mathcal{E})$ is defined by the following data.

Objects (R, A^n), here A is an object of $\mathcal{E}$, and R is a subset of $\Gamma(A^n)$.

Arrows For two objects (R, A^n) and (S, B^n), an arrow $f : (R, A^n) \longrightarrow (S, B^n)$ is defined to be $f : A \longrightarrow B$ in $\mathcal{E}$ such that for any $x_i : 1 \longrightarrow A$, $i = 1, \cdots, n$,

$$(x_1, \cdots, x_n) \in R \Rightarrow (fx_1, \cdots, fx_n) \in S$$

Two arrows $f, g : (R, A^n) \longrightarrow (S, B^n)$ are defined to be equal if they are equal in $\mathcal{E}$.

The composition is defined the same way as in $\mathcal{E}$. It is easy to check this is a category. Furthermore the following properties follow from that of $\mathcal{S}et$ and the nice behavior of the functor Γ.

Proposition 9 *1. If $\mathcal{E}$ has a terminator 1, so does $\mathcal{R}el_n(\mathcal{E})$, namely $(\{*\}, 1)$; here $\{*\}$ is $\Gamma(1^n)$, a singleton set. If $\mathcal{E}$ has an initial object 0, so does $\mathcal{R}el_n(\mathcal{E})$, namely $(\emptyset, 0)$, here 0 is an initial in $\mathcal{E}$.*

2. If $\mathcal{E}$ is a cartesian category, so is $\mathcal{R}el_n(\mathcal{E})$.

3. If $\mathcal{E}$ is cartesian closed, so is $\mathcal{R}el_n(\mathcal{E})$.

Proof Since $\Gamma(A \times B) \cong \Gamma A \times \Gamma B$, we only need to provide the cartesian closed structure of $\mathcal{R}el_1(\mathcal{E})$.

Recall that the objects are of the form (R, A), with R a subset of ΓA; here A is an object of $\mathcal{E}$. An arrow $f : (R, A) \longrightarrow (S, B)$ is, by definition, an arrow $f : A \longrightarrow B$ in $\mathcal{E}$, such that for any $x \in \Gamma A$, if x is in R, then the morphism $f \circ x$ is in S. Now the cartesian closed structure is given as follows:

1. $(\{*\}, 1)$ is a terminator, if 1 is one in $\mathcal{E}$. $\{*\}$ is the set $(\Gamma 1)$, a singleton.
2. For object $(R, A), (S, B)$, the product $(R, A) \times (S, B)$, is defined to be $(R \times S, A \times B)$, where for any $(x, y) \in \Gamma(A \times B) \cong (\Gamma A) \times (\Gamma B)$,

$$(x, y) \in R \times S \text{ iff } x \in R \text{ and } y \in S$$

It is easy to check that this is a product.

3. The exponential $(R, A) \Rightarrow (S, B)$ is defined to be $(R \Rightarrow S, A \Rightarrow B)$. Here $A \Rightarrow B$ is the exponential B^A of A, B in $\mathcal{E}$. The (unary)relation $R \Rightarrow S$, a subset of $\Gamma(A \Rightarrow B)$, is defined by

$$f \in (R \Rightarrow S) \quad \text{iff} \quad \forall x \in R\, (fx) \in S$$

⊣

As a practice, we check the following equation from Lambek and Scott [24] page 53.

E4a. $ev(h^* \pi, \pi') = h$ or the following

$ev(h^* \times I) = h$

here $h : (T, C) \times (R, A) \longrightarrow (S, B)$, h^* is the currying of h, i.e., $h^* : (T, C) \longrightarrow ((R, A) \Rightarrow (S, B))$, and ev is the evaluation map from $((R, A) \Rightarrow (S, B)) \times (R, A)$ to (S, B). The two projections are from $(T, C) \times (R, A)$ to (T, C) and (R, A), respectively. I is the identity arrow for (R, A).

For the given objects (R, A), (S, B) let $ev : B^A \times A \longrightarrow B$ be the evaluation map in $\mathcal{E}$. We want to show that this evaluation map in $\mathcal{E}$ gives an evaluation map in the relation category.

First, $ev : ((R, A) \Rightarrow (S, B)) \times (R, A) \longrightarrow (S, B)$ is an arrow in the latter category. By definition, $((R, A) \Rightarrow (S, B)) \times (R, A)$ is equal to $((R \Rightarrow S) \times R, B^A \times A)$. Now for any $(f, x) \in (R \Rightarrow S) \times R$, i.e., $x \in R, f \in R \Rightarrow S$, from 3 above we have $fx \in S$. Thus the evaluation arrow in $\mathcal{E}$ does induce an arrow from $((R, A) \Rightarrow (S, B)) \times (R, A)$ to (S, B) in the relation category.

Now it is easy to see that this ev is the real evaluation map in $\mathcal{Rel}(\mathcal{E})$, i.e., it comes from the identity map $I : (R, A) \Rightarrow (S, B) \longrightarrow (R, A) \Rightarrow (S, B)$ along the adjointness of the following two functors

$$(\_) \times (R, A) \dashv (R, A) \Rightarrow (\_)$$

Notice the following isomorphism

$$\mathrm{Hom}_{\mathcal{Rel}(\mathcal{E})}((T, C) \times (R, A), (S, B)) \cong \mathrm{Hom}_{\mathcal{Rel}(\mathcal{E})}((T, C), (R, A) \Rightarrow (S, B))$$

comes from

$$\mathrm{Hom}_{\mathcal{E}}(C \times A, B) \cong \mathrm{Hom}_{\mathcal{E}}(C, B^A)$$

and the definition of $\Rightarrow$. We leave this for the interested readers to check.

Finally, we have the following diagram

$$(T, C) \times (R, A) \xrightarrow{h^* \times I} ((R, A) \Rightarrow (S, B)) \times (R, A) \xrightarrow{ev} (S, B)$$

or

$$ev(h^* \times I) : (T, C) \times (R, A) \longrightarrow (S, B)$$

This map has the same domain and codomain as those of h; furthermore, $ev(h^* \times I) = h$ is true in $\mathcal{E}$. According to our definition of equality of maps in $\mathcal{Rel}(\mathcal{E})$, the equation is proved.

4.2.2 Examples of Logical Relation Categories

As we mentioned before, any (binary) relation on the atomic objects can be extended to a logical relation on the whole type structure. In what way do logical relation categories and logical relations connect? In this section we will investigate several examples to display the connection.

Example 15 Let $\mathcal{A}$ be a Henkin model. We examine $\mathcal{Rel}_2(\mathcal{A})$ from **Definition 7**.

Recall in the case of a Henkin model, 1 is a generator; hence, we identify $\Gamma(A)$ with A.

Let R be a binary relation on $\mathcal{A}_o$, for base type o. Then a logical relation $\mathcal{R} = \{R^\sigma\}_\sigma$ which extends R can be defined inductively as follows

$$\begin{array}{lll} xR^{\sigma \times \tau}y & \text{iff} & \pi_1 x R^\sigma \pi_1 y \text{ and } \pi_2 x R^\tau \pi_2 y \\ fR^{\sigma \to \tau}g & \text{iff} & \forall x, y \in \mathcal{A}^\sigma\, (xR^\sigma y \Rightarrow f(x) R^\tau g(y)) \end{array}$$

The category $\mathcal{Rel}_2(\mathcal{A})$ is as follows:

Objects (R^σ, A^σ), here A^σ is an object of $\mathcal{A}$ and R^σ is the extension of R on A^σ, so $R^\sigma \subseteq A^\sigma \times A^\sigma$.

Arrows For two objects (R^σ, A^σ) and (R^τ, A^τ), an arrow $f : (R^\sigma, A^\sigma) \longrightarrow (R^\tau, A^\tau)$ is defined if $f : A^\sigma \longrightarrow A^\tau$ in $\mathcal{A}$ and such that for any $x, y \in A^\sigma$,

$$(x R^\sigma y) \Rightarrow f(x) R^\tau f(y)$$

Notice that the last equation can be simply written as

$$f R^{\sigma \to \tau} f$$

This category is a logical relation category. It can also be thought of as a subcategory $R(\mathcal{A})$ of $\mathcal{A}$ with the same objects $\{A^\sigma \mid \sigma \in \mathcal{T}\}$ but only those arrows $f : A^\sigma \longrightarrow A^\tau$ which are *invariant* under the logical relation $\mathcal{R}$, i.e., such that $\mathcal{R}(f, f)$.

In the following we will explore more about this construction, and $R(\mathcal{A})$ will be mentioned not only for one relation R (here), also for a set of relations $\mathcal{N}(\mathcal{A}), \mathcal{S}(\mathcal{A}), etc.$ Another related construction [22] will be left until the next chapter.

The next example is quite interesting. It relates to one of the fundamental results in this field [11]. Recall, for a category $\mathcal{C}$, $\mathcal{C}^*$ is the well-pointed category obtained by adjoining infinitely many variables to the nonempty types. This is defined and discussed in Chapter 2.

Example 16 Consider $\mathcal{R}el_1(\mathcal{E})$, where $\mathcal{E} = \mathcal{S}et \times \mathcal{C}^*$, $\mathcal{C}$ the free *ccc* on a set of generators, $\mathcal{C}^*$ is the freely generated category obtained by adding infinitely many variables to each nonempty type. Consider the full subcategory $\mathcal{F}$ of $\mathcal{R}el_1(\mathcal{E})$ with objects of form $(R, (A, \sigma))$, where (A, σ) is an object of $\mathcal{S}et \times \mathcal{C}^*$, such that R is a subset of

$$\Gamma_{\mathcal{S}et \times \mathcal{C}^*}(A, \sigma) = \Gamma_{\mathcal{S}et}(A) \times \Gamma_{\mathcal{C}^*}(\sigma) \cong A \times \Gamma_{\mathcal{C}^*}(\sigma)$$

Furthermore, R is a partial surjective function from A onto $\Gamma(\sigma)$. To avoid some nonessential argument, we only consider the nonempty sets A.

Proposition 10 *$\mathcal{F}$ is cartesian closed.*

Proof Let $(R, (A, \sigma)), (R', (A', \sigma'))$ be two objects of $\mathcal{F}$. We will check that the product and exponential are still in $\mathcal{F}$, where these objects are (respectively)

$$(R \times R', (A \times A', \sigma \times \sigma')),\ (R \Rightarrow R', (A \Rightarrow A', \sigma \Rightarrow \sigma'))$$

Here $A \Rightarrow A', \sigma \Rightarrow \sigma'$ are exponentials in $\mathcal{S}et$ and $\mathcal{C}$, respectively.

What we want to prove is if R, R' are partial surjective functions from A to σ, A' to σ' respectively, then $R \times R', R \Rightarrow R'$ are also partial surjective from $A \times A'$ to $\sigma \times \sigma'$, (from $R \Rightarrow R'$ to $(S \Rightarrow S', \sigma \Rightarrow \sigma')$, respectively.)

First we show it is true for products.
$R \times R'$ is still a partial function. In fact, suppose

$$(f, u) R \times R' (g, v)$$
$$(f, u) R \times R' (h, w)$$

Where f, g, h are functions in $\mathcal{S}et$, and u, v, w are terms. Then we have $f R g, f R h, u R' v$, and $u R' w$. Hence, $g = h$ and $v = w$ or

$$(g, v) = (h, w)$$

Second, for any $(g, v) \in \Gamma(\sigma \times \sigma')$, there are $g \in \Gamma(\sigma), v \in \Gamma(\sigma)$. Hence, there are f, u such that

$$f R g, \quad \text{and} \quad u R v$$

Thus $(f,u)R \times R'(g,v)$, *i.e.* $R \times R'$ is partial surjective.

Now we show it is true for exponentials.
First, $R \Rightarrow R'$ is still a partial function. In fact, suppose

$$f(R \Rightarrow R')v$$
$$f(R \Rightarrow R')w$$

then for any $(x,t) \in A \times \Gamma(\sigma)$

$$xRt \quad \Rightarrow \quad f(x)R'ev(v,t)$$
$$f(x)R'ev(w,t)$$

Since R' is a function, so $ev(v,t) = ev(w,t)$, for any t in $\Gamma(\sigma)$. From extensionality, we have $v = w$. Thus $R \Rightarrow R'$ is a partial function.

Second, we show $R \Rightarrow R'$ is surjective. For any $v : 1 \longrightarrow (\sigma \Rightarrow \sigma')$, we want to find a $f : A \longrightarrow A'$ in $\mathcal{S}et$ such that $f(R \Rightarrow R')v$. For any $x \in A$, if $\exists t : 1 \longrightarrow \sigma$, such that xRt, then for this t, $ev(v,t) : 1 \longrightarrow \sigma'$. From the surjectivity of R', we know there is an $a_x \in A'$ such that

$$a_x R'ev(v,t)$$

If there is no such t, let a_x be any element in A' (recall, A' is nonempty). By using the Axiom of Choice, we get a function $f : A \longrightarrow A'$, which sends each x from A to a_x in A', and satisfies $f(R \Rightarrow R')v$.

This finishes our proof. ⊣

One of the first successful uses of partial surjective functions as logical relations was in Friedman[11]. Among the most recent applications is Cubric [7]. Together with [11], Cubric proved that the partial surjective functions form a structure preserving map from $\mathcal{S}et$ to an extension of $\mathcal{C}$, namely $\mathcal{C}^*$. Furthermore this connection can be extended to $\mathcal{C}$ also with structure preserved. Thus there is a structure preserving map from $\mathcal{S}et$ onto $\mathcal{C}$. Finally, this map has a "retract" which turns out to be a faithful representation. Notice that the map mentioned here is not a functor, hence the retract (a genuine functor) is only formal. In the following diagram, the injection from $\mathcal{C}$ to $\mathcal{C}^*$ is done by Cubric, and the other one by Friedman.

$$\mathcal{C} \hookrightarrow \mathcal{C}^* \hookrightarrow \mathcal{S}et$$

Theorem 6 (Cubric) *[7] There is a faithful representation from a free ccc into Set.*

The next example is $\mathcal{R}el_2(\mathcal{S}et)$, or simply $\mathcal{R}el(\mathcal{S}et)$, a binary relation category.

Example 17 $\mathcal{R}el(\mathcal{S}et)$ is a category whose objects are sets with a binary relation on each of them. An arrow

$$f : (R,A) \longrightarrow (S,B)$$

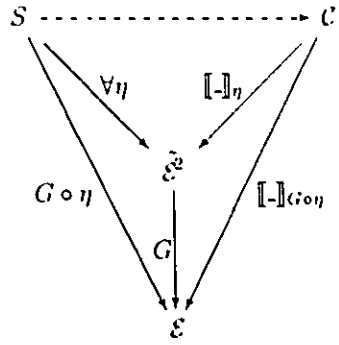
is a set-theoretic function $f : A \longrightarrow B$ between the two underlying sets such that for any $x,y \in A$, if xRy, then $f(x)Sf(y)$. Two arrows $f,g : A \longrightarrow B$ are said to be equal if they are equal in $\mathcal{S}et$.

It is easy to see that $\mathcal{R}el(\mathcal{S}et)$ is cartesian closed. In fact it is a sub-ccc of $\tilde{\mathcal{S}et}^2$.

$$\begin{array}{ccc} \mathcal{R}el(\mathcal{S}et) & \longrightarrow & \tilde{\mathcal{S}et}^2 \\ (R,A) & \longmapsto & (R, A \times A) \end{array}$$

This identifies (R,A) in $Rel(Set)$ with the object $(R, A \times A)$ in $\tilde{\mathcal{S}et}^2$.

Recall from the **Basic Lemma**, we have the following commutative diagram.



Here S is a discrete graph, $\mathcal{C}$ is the free ccc generated by S_o, η is any type assignment. G is the forgetful functor, i.e., $(A, R) \mapsto A$.

Pick some η such that $[\![\text{-}]\!]_\eta$ is faithful[1]. Then the representation $[\![\text{-}]\!]_{G\circ\eta}$ is also faithful.

In fact, for any t_1, t_2 in $\mathcal{C}$, if $[\![t_1]\!]_{G\circ\eta} = [\![t_2]\!]_{G\circ\eta}$, then $G([\![t_1]\!]_\eta) = G([\![t_2]\!]_\eta)$. Both G and $[\![\text{-}]\!]_\eta$ are faithful. Hence $t_1 = t_2$.

Thus, a corollary of the Basic Lemma is the following

Proposition 11 *There is a faithful representation from a free ccc into $\mathcal{R}el(\mathcal{S}et)$.*

4.2.3 Invariance Under a Family of Logical Relations

In the previous section in Example 15, the category we illustrated concerned arrows invariant under *one* logical relation or predicate. As we mentioned, one of the main purposes for which logical relations were invented was to study lambda definability. For this reason, we need to understand invariance under a *family* of logical relations. In studying definability, logical relations are used as a criterion: those arrows which are invariant under these relations form a subcategory and become candidates for further study; those relations which fail to be invariant are cut off from the category. Generally speaking, adding more relations will cut down more, since fewer elements will be invariant under more relations.

Let $\{\mathcal{A}\}_{i\in I}$ be a family of small cartesian closed categories where i ranges over an index set I. Consider the product set $\prod_{i\in I}\mathcal{A}$. A cartesian closed structure can be induced on it componentwise as follows:

- $(1)_i$ is a terminator;
- $(a_i)_i \times (b_i)_i$ is defined to be $(a_i \times b_i)_i$;
- $(a_i)_i \Rightarrow (b_i)_i$ is defined to be $(a_i \Rightarrow b_i)_i$;

where we write $(a_i)_i$ for $(a_i)_{i\in I}$.

For any (R, X) in $\mathcal{R}el(\mathcal{S}et)$, it generates a cartesian closed full subcategory $< R, X >$ inside $\mathcal{R}el(\mathcal{S}et)$. By analogy, $< R, X >$ is to $\mathcal{R}el(\mathcal{S}et)$ as the full type hierarchy T_X is to $\mathcal{S}et$.

On the other hand, the forgetful functor $G : \mathcal{R}el(\mathcal{S}et) \longrightarrow \mathcal{S}et$ when restricted to $< R, X >$ is a faithful embedding into T_X. Thus if we look at $< R, X >$ as a subcategory of T_X, *all its arrows are invariant under the logical relation $\mathcal{R}$ generated from R.*

Let $I = P(X^2)$, the set of all binary relations on the set X. Then, as we said above,

$$\mathcal{X} = \prod_{R\in I} < R, X >$$

[1] Any type assignment which sends each base type to an infinite set will do.

is still a cartesian closed category.

Let X be an infinite set, we denote $InRel(T_X)$ for the following cartesian closed category

Objects The same objects as those of T_X.

Arrows An arrow f from A to B in $InRel(T_X)$ is a function $f : A \longrightarrow B$ in T_X such that f is invariant under all the binary logical relations generated from the elements of $\mathcal{P}(X \times X)$ (all the binary relations on set X).

Theorem 7 (Plotkin) *If X is infinite, then there is a faithful representation of the one base type free ccc inside $InRel(T_X)$. Furthermore it is full for type ranks less than 3.*

4.2.4 Logical Permutations

The next example can be traced back to [25], see also [14].

Example 18 Instead of ordinary $\mathcal{R}(\mathcal{A})$, we consider the special case, $\mathcal{B}ij(\mathcal{S}et)$, a full subcategory of $\mathcal{R}el(\mathcal{S}et)$, with the objects (R, A), such that R is a permutation on A.

Let $\mathcal{Z}$ be the group of integers under addition. It is easy to verify that $\mathcal{B}ij(\mathcal{S}et)$ is indeed the topos $\mathcal{Z}$-sets $\mathcal{S}et^{\mathcal{Z}}$, [24] [14].

As a more general treatment, we can consider any permutation group $\mathcal{G}$, and the logical permutations inductively generated using the elements of $\mathcal{G}$. We might want to ask which arrows are left invariant by applying such logical permutations.

A more general description can be given for any monoid $\mathcal{M}$ as follows. An $\mathcal{M}$-set A is a functor from $\mathcal{M}$ to $\mathcal{S}et$. From another point of view, it is a monoid homomorphism $\mathcal{M} \longrightarrow A^A$ where A^A is the monoid of endomaps of A under composition. So an $\mathcal{M}$-set is a set together with a monoid $\mathcal{M}_A$, here $\mathcal{M}_A$ is a submonoid of A^A, and a homomorphic image of $\mathcal{M}$. The functoriality is given below

- $A(xy) = A(x) \circ A(y)$
- $A(Id) = Id_A$

which is exactly saying A is a homomorphism. Conversely, for any such a pair $(A, \mathcal{M}_A)$, if $\mathcal{M}_A$ is a submonoid of A^A and a homomorphic image of $\mathcal{M}$, then the pair is an $\mathcal{M}$-set, *i.e.* $(A, \mathcal{M}_A) \in \mathcal{S}et^{\mathcal{M}}$. By abusing notation, we can write

$$\mathcal{S}et^{\mathcal{M}} = \{(A, \mathcal{M}_A) | A \in \mathcal{S}et, \mathcal{M}_A \text{ is a submonoid of } A^A, \text{ and homomorphic image of } \mathcal{M}.\}$$

$\mathcal{M}$-sets form a cartesian closed category, in fact a topos[24]. A special kind of $\mathcal{M}$-set is when $\mathcal{M}$ is in fact a group G. In this case, we have a simpler expression of G-set. It is a set together with a group (namely, a subgroup of the permutations on A). The cartesian closed structure of $\mathcal{S}et^G$ can be described as

- $(1, \{e\})$ is a terminal
- $(A, G_A) \times (B, G_B) = (A \times B, G_A \times G_B)$
- $(A, G_A)^{(B, G_B)} = (A^B, G_{B \Rightarrow A})$

Here the group $G_{B \Rightarrow A}$ is defined as follows:

For any $g \in G, g_{B \Rightarrow A} \in G_{B \Rightarrow A}$ is equal to $\lambda x : A^B . g_A \circ x \circ g_B{}^{-1}$.

Notice that this description of cartesian closed structure is not the same as the ordinary topos-theoretic description (which comes via the Yoneda Lemma) [24].

4.2.5 $\mathcal{P}er(\mathcal{C})$, a Generalized *per*-like Category

Equivalence relations (reflexive, symetric and transitive binary relations) posses very nice properties among ordinary binary relations. If the reflexive requirement is removed from the above notion, we get the so called *partial equivalence relations*, or *Pers*. It turns out that *Pers* play a very important role in the study of semantics. We invite readers to [33] and the references there.

More generally, we consider a subcategory of $\mathcal{R}el(\mathcal{S}et)$ which has objects (R, A) such that R is a partial equivalence relation on A. However, an even better construction is the following.

Example 19 The category $\mathcal{P}er(\mathcal{S}et)$ is the following category

Objects (R, A), where A is a set, and R is a partial equivalence relation on A.

Arrows For two objects (R, A) and (S, B), an arrow $f : (R, A) \longrightarrow (S, B)$ is a set-theoretic function $f : A \longrightarrow B$ and such that for any $x, y \in A$,

$$xRy \Rightarrow f(x)Sf(y)$$

For any two arrows $f, g : (R, A) \longrightarrow (S, B)$, define $f = g$ if for any x, y in A,

$$xRy \Rightarrow f(x)Sg(y)$$

Remark 16 The equality here is different from the relation category $\mathcal{R}el(\mathcal{S}et)$.

Actually, the main difference between $\mathcal{P}er(\mathcal{S}et)$ and $\mathcal{R}el(\mathcal{S}et)$ is the equality on arrows. So although $\mathcal{P}er(\mathcal{S}et)$ arrows are ordinary set theoretic maps between *Pers* in $\mathcal{R}el(\mathcal{S}et)$, equality of maps in $\mathcal{P}er(\mathcal{S}et)$ is defined with the assistance of the related partial equivalence relations. It is easy to prove the following:

Proposition 12 *$\mathcal{P}er(\mathcal{S}et)$ is cartesian closed.*

Let $\mathcal{A}$ be any Henkin model, R a logical partial equivalence relation over it. As above, we can consider $\mathcal{P}er(\mathcal{A})$ for any Henkin model $\mathcal{A}$. Then the logical relation category of it is equivalent to the quotient model which turns out to be a Henkin model again. This is the method called the *extensional collapse* [12].

4.3 Other Related Categories and Interconnection

As we noted previously,

> If R is a binary relation on the set B, then $< R, B >$ is a full subcategory of $\mathcal{R}el(\mathcal{S}et)$.

In particular, if R is a permutation on the set B, then $< R, B >$ is a full subcategory of $\mathcal{S}et^{\mathcal{Z}}$. Furthermore if G is a permutation group on the set B, then $< G, B >$ is a full sub category of $\mathcal{S}et^{G}$.

Remember, we use logical relations as a kind of *sieve* to refine a category by removing "non-qualified" arrows. Two typical examples are: all the logical permutations, and all the binary logical relations. The following shows us how powerful they are.

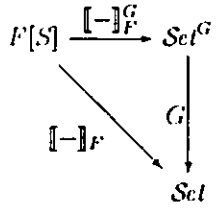
Theorem 8 *Let S be a discrete graph, say a singleton, and $\mathcal{C}$ be the freely generated ccc on S. Then for any group G, there is no non-trivial full embedding $\mathcal{C} \longrightarrow \mathcal{S}et^{G}$.*

Proof Let G be an arbitrary group. By Cayley's Theorem, suppose G is a subgroup of $\mathrm{Sym}(B)$, for some infinite set B. WLOG we let $S = \{\tau\}$, i.e., there is only one base type, hence under any interpretation of the base type, $\mathcal{C}$ goes to a T_B for some set B under a uniquely determined ccc functor F.

(It is easy to see that if B is finite, then F cannot be faithful. For example, there are infinitely many Church numerals for any type symbol σ [13], and they are all different from each other. On the other hand, there are only finitely many elements in each level of T_B.)

Nontrivial means that $|B| \geq 2$. In fact, for any such T_B, a functional EQ is defined such that $EQ(a,b) = [\![K]\!]$ if $a = b$ and $EQ(a,b) = [\![K_*]\!]$ if $a \neq b$, where $K = \lambda xy.x$, $K_* = \lambda xy.y$. We will show that EQ is invariant for any logical permutation induced from one on B, but EQ is not λ-definable.

First faithful:



If for any t, s, $[\![t]\!] \neq [\![s]\!]$ then as set-theoretic functions, they are not equal in Set^G either. So $[\![-]\!]$ is faithful.

Second the functor $[\![-]\!]_F^G : F[S] \longrightarrow Set^G$ is not full at all. The following is an example from [39]:

For any given type σ, consider the function

$$\mathbf{EQ} : (\sigma \times \sigma) \rightarrow (\sigma \rightarrow (\sigma \rightarrow \sigma)) \qquad (= \rho \longrightarrow \tau)$$

which for any $a, b \in \mathcal{A}_\sigma$,

$$\mathbf{EQ}ab = \textbf{if } \ a = b \ \textbf{ then } [\![\lambda xy.x]\!] \ \textbf{ else } [\![\lambda xy.y]\!]$$

For any permutation g, $g_{\rho\rightarrow\tau}(\mathbf{EQ}) = g_\tau \circ \mathbf{EQ} \circ (g_\rho)^{-1}$. (Here, though we write $g_{\rho\rightarrow\tau}$, it is indeed a permutation of type $(\rho \rightarrow \tau) \longrightarrow (\rho \rightarrow \tau)$, the same for the others.) For any $a, b \in \mathcal{A}_\sigma$, if $a \neq b$, then $(g_\sigma)^{-1}a \neq (g_\sigma)^{-1}b$. So

$$g_{\rho\rightarrow\tau}(\mathbf{EQ})ab = g_\tau \circ \mathbf{EQ} \circ g_\rho^{-1}ab = g_\tau \circ \mathbf{EQ}(g_\sigma^{-1}a)(g_\sigma^{-1}b)$$

which is $g_\tau \circ K_* = K_*$. On the other hand,

$$g_{\rho\rightarrow\tau}(\mathbf{EQ})aa = g_\tau \circ \mathbf{EQ} \circ (g_\rho)^{-1}aa = g_\tau \circ \mathbf{EQ}((g_\sigma)^{-1}a)((g_\sigma)^{-1}a)$$

which is $g_\tau \circ K = K$.

From extensionality, $g_{\rho\rightarrow\tau}(\mathbf{EQ}) = \mathbf{EQ}$. Hence it is invariant under any permutation.

However, it is not λ-definable, as we can see that a simple binary relation will falsify it.

Let $R = \{(a,b),(b,a),(a,a)\}$ here $a \neq b$, then $aR^\circ b, bR^\circ a$. As our notation that $\mathbf{EQ}$ is of type $(o \times o) \longrightarrow \tau (\equiv \rho \longrightarrow \tau)$. If $(\mathbf{EQ})R^{\rho\rightarrow\tau}(\mathbf{EQ})$, then we should have $((\mathbf{EQ})ab)R^{o\rightarrow(o\rightarrow o)}((\mathbf{EQ})aa)$. This gives

$$(\lambda xy.y)R(\lambda xy.x) \qquad \text{or} \qquad K_*RK$$

From aRb, bRa, we will further get

$$(\lambda xy.y)abR(\lambda xy.x)ba$$

This is simply bRb. But, for this example it is not true.

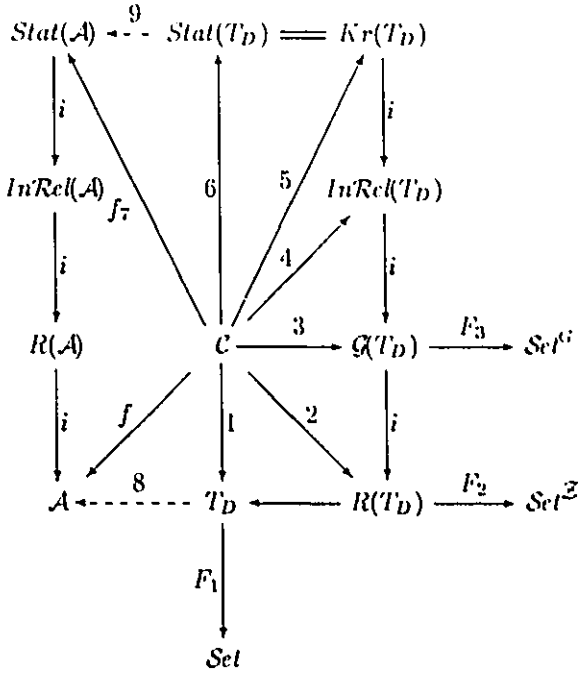
⊣

Let D be an infinite set, then the base type mentioned for the full type hierarchy T_D is D. Let $\mathcal{C}$ be the free ccc on one base type. In the following, we will investigate the different "sieves" applying to T_D.

Recall, for a Henkin model $\mathcal{A}$, $\mathcal{A}^*$ is the Henkin model generated from $\mathcal{A}$ by adjoining infinitely many variables to each nonempty type. A logical relation on $\mathcal{A}^*$ is called a Statman relation on $\mathcal{A}$. Let $Stat$ be the set of Statman logical relations on $\mathcal{A}$ [46], Kr the set of Kripke logical relations [39], $InRel$ the set of

all binary logical relations, $\mathcal{G}$ the group of all or part of the logical permutations, and R one fixed logical permutation on the base type (atomic object).

The diagram below describes the relationship of the categories mentioned above. Except for dotted arrows 8, 9 and arrows f, f_7, all the other arrows are faithful representations, and the arrows labeled F_i are also full. The behavior of the arrows f, f_7 is dependent on $\mathcal{A}$. Consider the following diagram, where i denotes an appropriate "inclusion".



The following are some remarks. The comment labelled i refers to the functor labelled i.

1. The faithfulness of arrow 1 comes from Cubric[7]. It is easy to see that there are uncountably many elements (=set-theoretic functions) in $D_{o\to o}$, since $D_{o\to o} = D^D$, and D is infinite. Hence there are uncountably many morphisms in $\mathrm{Hom}(D, D)$. On the other hand, there are only countably many morphisms of $\mathcal{C}$ in total, since the languages of typed λ-calculus is countable. So this functor is not full at all.

2. The fact that arrow 2 is weakly full is from Läuchli's Theorem [25] and [14]. In fact, what they proved was stronger than what we said here. Both of them consider coproduct as well, and Läuchli also considers predicate logic. The weak fullness of functors 3, 4 is because of the same reason.

3. The representation from $\mathcal{C}$ to $\mathcal{G}(T_D)$ is faithful but not full. For example, the term **EQ** in the proof of **Theorem 8**, in this chapter, is invariant under any permutations but not definable.

4. The representation from $\mathcal{C}$ to $InRel(T_D)$ is faithful. Furthermore, it is full for types with rank less than 3 [39]. For higher types, the question is still open.

5. Plotkin [39] proved that if a morphism from the full type hierarchy satisfies all the Kripke relations, then it is definable. Statman gave a very tricky example to verify that Plotkin's theorem is not true for the ordinary Henkin Models, i.e., it is only true for full type hierarchy [46].

6. Statman [46] claimed that for any Henkin model $\mathcal{A}$, if a morphism t satisfies all the logical predicates in $\mathcal{A}$, then t is definable, *i.e.* , equal to an image of a morphism from the free category. So if $\mathcal{A}$ is full type hierarchy, 6 is a fully faithful cartesian closed functor.

7. The functor f_7 depends on the model $\mathcal{A}$, i.e., it depends on f. Recall f is the unique ccc representation $\mathcal{C} \longrightarrow \mathcal{A}$ for a given type assignment. In general, we might want to say that $Stat(\mathcal{A})$ is a cartesian

closed category with all its arrows λ-definable, but which may also realize some unexpected equations between these arrows.

8. The dotted arrows 8 and 9 are not functors. **Theorem 5** of Chapter 3 is a version of the fundamental theorem of algebra, which characterizes a Henkin model by two logical relations h and *equality* from *Set*. So dotted arrow 8 is the partial homomorphism, and 9 could be thought as the restriction of 8 on $Stat(T_D)$. It is interesting to notice that 9 is a total function.

9. If Statman's theorem is correct, then the arrows of $Stat(\mathcal{A})$ are all definable; then $Stat(\mathcal{A})$ is, in fact, a quotient $\bar{\Lambda}/\mathrm{Th}(\mathcal{A})$. However, as far as we know, the proof of Statman's theorem has never been published.

Chapter 5

Lambek's $\mathcal{C}^R$ as a logical relation category

As an alternative example of a logical relation category, we briefly investigate Lambek's subcongruence completion category $\mathcal{C}^R$ in this chapter. The original construction is given in [22], which is as an equalizer completion of a given category $\mathcal{C}$. It turns out that the construction is too good: one also gets coequalizers. We will only use this construction to illustrate the similarity between it and our logical relation categories. For the other properties of this construction, we invite the reader to read [22].

5.1 Introduction

Let $\mathcal{C}$ be a small category. A new category $\mathcal{C}^R$ is defined as follows:

objects: The objects are tuples of the form

$$(R, i, j, A)$$

where $R = \{R_C\}_{C \in \mathcal{C}_o}$, $i = \{i_C\}_{C \in \mathcal{C}_o}$, $j = \{j_C\}_{C \in \mathcal{C}_o}$ are families indexed by objects of $\mathcal{C}$, and A is an object of $\mathcal{C}$ such that for any object $C \in \mathcal{C}_o$ we have arrows

$$\mathrm{Hom}(C, A) \xleftarrow{i_C} R_C \xrightarrow{j_C} Hom(C, A)$$

such that i_C, j_C are jointly monic from R_C to $\mathrm{Hom}(C, A)$ in $\mathcal{S}et$, and the pair (i_C, j_C) makes R_C a partial equivalence relation on $\mathrm{Hom}(C, A)$. Furthermore, for any map $h : C \longrightarrow D$ in $\mathcal{C}$, $\mathrm{Hom}(h, D) : Hom(D, A) \longrightarrow Hom(C, A)$ is a map in $\mathcal{P}er(\mathcal{S}et)$. We will write $\mathcal{R}^A$ for this object (R, i, j, A) and R^A_C for the component at C.

morphisms: $f : \mathcal{R}^A \longrightarrow \mathcal{R}^B$, if f is a map $f : A \longrightarrow B$ in $\mathcal{C}$, and for any $C \in \mathcal{C}_o$, $x R^A_C y$ then $fx R^B_C fy$.

equality: Two maps $f, g : \mathcal{R}^A \longrightarrow \mathcal{R}^B$ are called equal, if for any $C \in \mathcal{C}_o$, if $x R^A_C y$ then $fx R^B_C gy$. Equivalently, for any $x \in Dom(R^A_C)$, $fx R^B_C gx$.

If $\mathcal{C}$ is cartesian closed, $\mathcal{C}^{\mathcal{R}}$ is too. The following is the cartesian closed structure of it.

1. $1 = \{(*, Hom(C, 1))\}_C$ is a terminal object in $\mathcal{C}^{\mathcal{R}}$, here 1 is terminal in $\mathcal{C}$.
2. $\mathcal{R}^A \times \mathcal{R}^B$ is defined as $\mathcal{R}^{A \times B}$, a per on $\mathrm{Hom}(\_, A \times B)$, such that for any $C \in \mathcal{C}_o$,

$$u R^{A \times B}_C v \Leftrightarrow \pi_1(u) R^A_C \pi_1(v), \text{and} \pi_2(u) R^B_C \pi_2(v)$$

3. $\mathcal{R}^A \Rightarrow \mathcal{R}^B$ is defined to be $\mathcal{R}^{A \Rightarrow B}$ which, for any $C \in \mathcal{C}_o$, is a per on $\mathrm{Hom}(C, A \Rightarrow B)$, such that for any $x, y : C \longrightarrow A \Rightarrow B$

$$x R^{A \Rightarrow B}_C y \Leftrightarrow \forall a R^A_C b (\mathrm{ev}(x, a) R^B_C \mathrm{ev}(y, b))$$

Theorem 9 (Lambek) *There is a fully faithful embedding J_1, from $\mathcal{C}$ into $\mathcal{C}^{\mathcal{R}}$, which preserves the ccc structure.*

For any $A \in \mathcal{C}_0$, there are many $\mathcal{R}^A$'s. For example, we can choose different partial equivalence relations to get different $\mathcal{R}^A$'s. The image of A under J_1 is chosen to be r^A, for which for any $C \in \mathcal{C}_0$, the associated per is the identity relation on $\mathrm{Hom}(C, A)$. For $f : A \longrightarrow B$, and any C, $J_1 f$ is $\mathrm{Hom}(C, f)$. This is well defined since $x = y$ always implies $fx = fy$. Arguing via Yoneda embedding, one can show that J_1 is fully faithful.

5.2 $C^{\mathcal{R}}$ as a Relation Category

Recall $\mathcal{P}er(\mathcal{S}et)$, which is a logical relation category. Its objects are of the form (R, A). An arrow $f : (R, A) \longrightarrow (S, B)$ is a function $f : A \longrightarrow B$ which preserves the partitions induced on A and B: i.e., for all $x, y \in A, xRy \Rightarrow (fx)S(fy)$.

Theorem 10 *There is a fully faithful cc functor J_2 from $\mathcal{C}^{\mathcal{R}}$ into $\mathcal{P}er(\mathcal{S}et)^{\mathcal{C}^{op}}$.*

The mapping is

$$\mathcal{R}^A \mapsto \{(\mathcal{R}^A_C, Hom(C, A))\}_C \quad .$$

Then

$$\begin{array}{ccc} C & \mapsto & (\mathcal{R}^A_C, Hom(C, A)) \\ \Big\downarrow h & & \Big\uparrow Hom(h, A) = \mathcal{R}^A(h) \\ D & \mapsto & (\mathcal{R}^A_D, Hom(D, A)) \end{array}$$

and

$$Hom(\mathcal{R}^A, \mathcal{R}^B) = Nat(\{(\mathcal{R}^A_C, Hom(C, A))\}_C, \{(\mathcal{R}^B_C, Hom(C, B))\}_C)$$

First, each $f : \mathcal{R}^A \longrightarrow \mathcal{R}^B$ is a natural transformation by the following commutative diagram which depends on the associativity of composition

$$\begin{array}{ccc} (\mathcal{R}^A_C, Hom(C, A)) & \xrightarrow{fC} & (\mathcal{R}^B_C, Hom(C, B)) \\ \Big\uparrow \mathcal{R}^A(h) & & \Big\uparrow \mathcal{R}^B(h) \\ (\mathcal{R}^A_D, Hom(D, A)) & \xrightarrow[fD]{} & (\mathcal{R}^B_D, Hom(D, B)) \end{array}$$

Second, each $\eta \in Nat(\{(\mathcal{R}^A_C, Hom(C, A))\}_C, \{(\mathcal{R}^B_C, Hom(C, B))\}_C)$ is one of these. This is true from the Yoneda Lemma.

Remark 17 We can consider ordinary binary relations instead of pers, and in that way, we will get a fully faithful embedding

$$\mathcal{C}^{\mathcal{R}} \hookrightarrow \mathcal{R}el(\mathcal{S}et)^{\mathcal{C}^{op}} \quad .$$

Chapter 6

Strong Normalization and Church-Rosser

We have been talking a lot about the semantic part of lambda calculi. As we mentioned earlier, there is another aspect of a lambda calculus: the proof-theoretical part.

In this chapter, we will briefly investigate the rewrite system R (see [9], or Chapter 2) used for the λ-calculus, which corresponds to the free bi-cartesian closed category [9], [24]. Furthermore, we will discuss some interesting connections between syntax and semantics, i.e., we will see how the rewrite system affects the semantics. In particular, in section 6.3 we shall develop some new theorems for representing free categories.

Equality is a fundamental logical concept of universal importance. Equations are often a very natural way of expressing mathematical knowledge, and the properties of the equality relation allow us to reason by "replacing equals by equals", a very powerful and general technique. A key idea in equational reasoning is to treat a set of equations as rules for rewriting expressions. A *rewrite rule*, written $M \longrightarrow N$, is an equation which is used in only one direction, i.e., M is replacable by N, but not vice versa. A set of rewrite rules is called a *rewrite system*.

Recall there are three rewrite systems we have introduced in Chapter 2. They are

R^{-}:	The ordinary contraction rules for simply typed lambda calculus with coproduct types.
R:	The context sensitive expansion rules with full restrictions.
R^{∞}:	The context-free expansion rules

We will study each of them and also investigate the equivalence relations developed from them.

6.1 $\mathcal{SN}, \mathcal{CR}$ for Simply Typed Lambda-Calculus

The theory of rewriting centers around the concept of "normal form", an expression (lambda term, in our case) that cannot be rewritten any further. Rewrite systems are readily used as a programming language. Computation consists of rewriting to a normal form; when the normal form is unique, it is taken as the value of the initial expression.

Basically there are two kinds of rewriting rules: *contraction* and *expansion*. Let us use an earlier example

$$x = \lambda f^{(o \to o)} y^{o}.x(\lambda z^{o}.fz)y : (o \to o) \to (o \to o)$$

to discuss some arguments or "simplifications" behind it. From an operational point of view, the contraction rules just state possible "optimizations" of an expression, like simplification in ordinary algebra. For example, a term like $\lambda f^{(o \to o)} y^{o}.x(\lambda z^{o}.fz)y$ is more complex than x, but behaves the same way. It is convenient to replace all its occurrences by x, as this transformation will yield an equivalent, but smaller expression. This idea backs the use of contraction rules, and is fully accepted in other fields of mathematics. However, from a theoretical point of view, there is another aspect. The equation states a relation between a term and its type. It just tells us that whenever a term x has a functional type, then it must really be a function, built

by λ-abstractions. We ought to be able to replace x by $\lambda f^{(o\to o)}y^o.x(\lambda z^o.fz)y$, since x gives no information about its behavior.

Sometimes different rewrite rules are not only a matter of taste: they affect normalization and confluence (see the definition below) of the system directly[1]. Even for a given fixed rewrite system, there is still a nondeterministic problem around: what reduction strategy does one want to use? For example, the typical example in (untyped) lambda calculus that strategy affects the rewriting result is the following:

Example 20 Let Ω be the term $\lambda x.xx$, y a variable, and K be the term $\lambda xy.x$. Consider the application term

$$Ky(\Omega\Omega)$$

Notice, $Ky(\Omega\Omega) = y$ by the usual equational theory of lambda calculus. What about from a rewriting viewpoint?

If doing inner-most reduction (call-by-value, in computer science terminology), which usually always simplifies the computation (unfortunately, not this time) one will be trapped into an infinite loop

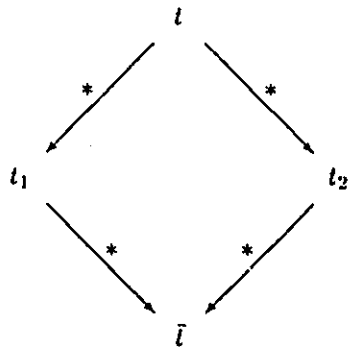
$$\Omega\Omega \xrightarrow{\beta} \Omega\Omega \xrightarrow{\beta} \cdots$$

On the other hand, if doing outer-most reduction (call-by-name, or lazy evaluation), one will luckily get $Ky(\Omega\Omega) \to y$, and y is a normal form.

We remind the reader again of two standard terminologies:

Definition 8 *1.* A term is said to satisfy $\mathcal{SN}$ (strongly normalizing), if every possible reduction sequence leads to a normal form, no matter which strategy is used. A rewrite system is said to be $\mathcal{SN}$ if all terms under this system satisfy $\mathcal{SN}$.

2. A rewrite system is said to be $\mathcal{CR}$ (Church-Rosser, or satisfy the "Diamond Property"), if any two reductions of a term are confluent: that is, for any term t if there are reduction paths such that $t_1 \overset{*}{\longleftarrow} t \overset{*}{\longrightarrow} t_2$, then there is a term $\bar{t}$ such that t_1 and t_2 can be reduced to $\bar{t}$. The picture is a "diamond":



If a theory is $\mathcal{SN}$ and $\mathcal{CR}$ it satisfies many properties. Among them is ***decidablility of the word problem.*** For example, to determine if two given terms s and t are equal or not, one only needs to do reductions and compare their normal forms. If the normal forms are the same, then they are equal, otherwise they are not. Another good property is ***consistency.*** For example, two different variables are not equal. Hence as long as there are inequivalent terms, the equational theory is consistent.

As reported in [24] the traditional contraction rules are not Church-Rosser with product types: they are only strongly normalizing. The problem occurs in dealing with extensional equations. For example in the case of simply typed λ-calculus, the extensional rules for type 1 arise in the following cases, against the other two extensional rules η and $\times!$.

[1] See also Chapter 2 Remark 6 for the evidence.

$$\begin{array}{ccccc} \lambda y.* & \xleftarrow{!!} & \lambda y^o.sy & \xrightarrow{\eta} & s \\ (*,*) & \xleftarrow{!!} & (\pi_1 t, \pi_2 t) & \xrightarrow{\times-r} & t \end{array}$$

where $s : \sigma \to 1$, and $t : 1 \times 1$. It is easy to see that since the terms on the left and right are irreducible, they are different from each other. Consequently, the confluence is destroyed. Lambek and Scott prove the confluence without type 1 [24]. A better approach is developed by G.Mints who mended this annoyance, by orienting the extensional equations as expansion rules with some restriction to context. (For a discussion about this, see [7], or Chapter 2). Prawitz from the view point of proof theory also suggested expansions as rewriting rules, see [40]. It turns out that this is not only quite reasonable, but also satisfies normalization and confluence. What is lost is the freedom from context, and hence, the reduction rules are context sensitive and rules (as schema) may not be closed under substitution.

Example 21 Consider

$$S = \lambda x^{o\to(o\to(o\to o))} y^{o\to o} z^o.(xz)(yz)$$

In the ordinary case, this is its normal form. However, if we use rewrite system $\mathcal{R}$, believe it or not, this well-known combinator is not in its normal form yet!

Consider its immediate subterms and their normal forms as below:

$$\begin{array}{c} x \to \lambda t^o.xt \to \lambda t^o u^o.(xt)u \to \lambda t^o u^o v^o.((xt)u)v \\ xz \to \lambda t^o.(xz)t \to \lambda t^o u^o.((xz)t)u \quad . \end{array}$$

We cannot substitute either of the terms which are after x into S. The same situation for xz.

What we can do is an outmost reduction to get

$$S \longrightarrow \lambda t^o xyz.((xz)(yz))t \quad .$$

The right hand side term is irreducible with respect to the rewrite system $\mathcal{R}$.

6.2 $\mathcal{SN}, \mathcal{CR}$ and λ-calculus with Coproduct Types

In this section, we consider the λ-calculus with coproduct types. The proofs of $\mathcal{SN}$ and $\mathcal{CR}$ in this section are mostly from [9], we only make some of them smoother to read.

Proposition 13 *If t is an I-term other than an abstraction whose immediate subterms are $\mathcal{SN}$, then t is $\mathcal{SN}$*
.

Remember, a non-abstraction I-term is one of the following,

$$* (: 1), \quad x \text{ (a variable)}, \quad \langle s, t\rangle, \quad \Box, \quad \kappa_i t, \quad [f_1, f_2]$$

If all its immediate subterms (provided there are some) are in normal form, there is really no more reduction to do on them.

The situation with λ-abstraction is more subtle, e.g., even if t is in its normal form, $\lambda x : \sigma + \tau.t$ can still be reduced to something like

$$[\lambda x_1{}^\sigma.t[x := (\kappa_1 x_1)], \lambda x_2{}^\tau.t[x := (\kappa_2 x_2)]]$$

and the components might be able to be separated further.

To prove normalization Dougherty uses the standard method of computability predicates **Comp**. (Remember, the predicate **Comp** (called RED by Girard [13]) is a logical predicate. In addition, Dougherty introduces the following concept.

Definition 9 For a type τ, the **pseudo-variables** $PV(\tau)$ are the variables of type τ and with, in case $\tau \equiv \tau_1 + \tau_2$, the set $\{\kappa_i x_i; x_i \in PV(\tau_i), i = 1, 2\}$.

For example, let the types $2 = 1 + 1, 3 = 1 + 2$, here, of course, 1 is the base type 1.

$$\begin{aligned} PV(2) &= \{x : 2; x \in V\} \cup \{\kappa_i x; x \in V^2\} \\ PV(3) &= \{x : 3; x \in V\} \cup \{\kappa_1 x; x \in V^1\} \cup \{\kappa_2 x; x \in PV(2)\} \\ &= \{x : 3; x \in V\} \cup \{\kappa_1 x, \kappa_2\kappa_2 x, \kappa_2\kappa_1 x; x \in V^1\} \cup \{\kappa_2 x; x \in V^2\} \end{aligned}$$

In the above equations, we use V^n to denote the set of variables of type n. Recall that **Comp** is a logical predicate, which is going to used on the set of terms, an applicative structure. As we mentioned before (Section 3.1), a kind of *admissibility* property is necessary.

The following is in Dougherty [9], a technical lemma for establishing admissibility.

Lemma 4 *For a type τ, if $PV(\tau) \subseteq \mathcal{SN}$, then $\lambda x : \tau.t$ is $\mathcal{SN}$ provided*

$$\{t[x := y]; y \in PV(\tau)\} \subseteq \mathcal{SN}$$

Finally, after showing **Comp** is an admissible logical predicate, a version of the Basic Lemma gives the following.

Theorem 11 *The following are proved in [9].*

1. *R^- and R are $\mathcal{SN}$.*
2. *R^- and R^∞ are confluent.*

This is the best one can expect as we have the following counter examples. We recall the rewrite systems R R^- and R^∞ in Figure 3 of Chapter 2.

Example 22 R is not confluent. Consider the following examples for (+!) and (0!).

$$[\lambda x^\sigma.z, \lambda y^\tau.z]u \xleftarrow{+!} (\lambda t^{\sigma+\tau}.z^b)u \xrightarrow{\beta} z$$

Both left and right terms are irreducible, here b is a base type.

As for the case of R^∞, see **Remark 6** in Chapter 2.

The following result is quite interesting which reveals the relation between $\rightarrow$ and $\Rightarrow$. We introduce some standard terminology [24],[15].

Definition 10 Let R be a binary relation on a set A. *The equivalence relation generated by R* is defined to be

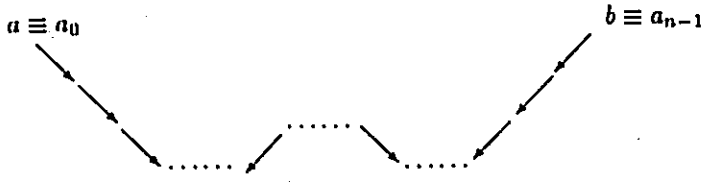
$$< R >= Id \bigcup (R \cup R^{op}) \bigcup (R \cup R^{op})^2 \bigcup \cdots \bigcup (R \cup R^{op})^n \bigcup \cdots$$

where S^n is the relational product $S \cdots S$ (n times).

Therefore $a < R > b$ means that for some number n, $a(R \bigcup R^{op})^n b$. If we write $a_i \rightarrow a_j$ for $(a_i, a_j) \in R$, and $a_i \leftarrow a_j$ for $(a_i, a_j) \in R^{op}$, then $a(R \bigcup R^{op})^n b$ means there is a sequence $a_0, a_1, \cdots, a_{n-1}$ such that

$$a \equiv a_0(R \cup R^{op})a_1(R \cup R^{op})a_2 \cdots a_{n-2}(R \cup R^{op})a_{n-1} \equiv b$$

Geometrically, there is a zigzag path connecting a to b in the following manner:



Proposition 14 *Each reduction step " $\Rightarrow$ " can be replaced by finitely many reductions " $\leftarrow$ " and " $\rightarrow$ ".*

Proof

1. There is no difference for computation rules. See **Figure 2** in Chapter 2.
2. For the application of η-expansion rule
$$f \longrightarrow \lambda x.fx$$
there are three extra conditions for the term f as well as for the context in which f lives. They are
   (a) f is not an abstraction
   (b) f is not a term of the form $[f_1, f_2]$
   (c) f is not in a context $C[(fx)]$

   We replace them respectively.
   (a) Let $f \equiv \lambda y.t \Rightarrow \lambda x.(\lambda y.t)x$, here x is not free in t. This one step can be replaced by the following simple reduction:
$$\lambda x.(\lambda y.t)x \xleftarrow{\beta} \lambda x.t[y := x] \equiv f$$
   (b) $[f_1, f_2] \Rightarrow \lambda x : \sigma + \tau.[f_1, f_2]x$ can be replaced by the following path.
$$\begin{array}{ll}
\lambda x : \sigma + \tau.[f_1, f_2]x \xrightarrow{\pm!} & \\
[\lambda x : \sigma + \tau.[f_1, f_2]x \cdot \kappa_1, \lambda x : \sigma + \tau.[f_1, f_2]x \cdot \kappa_2] \equiv & \\
[\lambda x : \sigma.[f_1, f_2](\kappa_1 x), \lambda y^\tau.[f_1, f_2](\kappa_2 y)] \xrightarrow{\kappa} & \\
[\lambda x : \sigma.f_1 x, \lambda y^\tau.f_2 y] \xleftarrow{\eta} [f_1, f_2] & \text{if neither } f_i \text{ is an abstraction;} \\
\text{or} \equiv [\lambda x : \sigma.(\lambda x : \sigma.g(x))x, \lambda y^\tau.(\lambda y^\tau.h(y))y] \xrightarrow{\beta} & \text{if both } f_i \text{ are abstractions;} \\
[\lambda x : \sigma g(x), \lambda y^\tau.h(y)] \equiv [f_1, f_2] & \text{or} \\
\equiv [\lambda x : \sigma.(\lambda x : \sigma.g(x))x, \lambda y^\tau.f_2 y] \xrightarrow{\beta} & \text{if one of them is an abstraction} \\
[\lambda x : \sigma g(x), \lambda y^\tau.f_2 y] \xleftarrow{\eta} & \\
\equiv [f_1, f_2] &
\end{array}$$
   (c) $ft \Rightarrow (\lambda x.fx)t$ could be replaced by $ft \leftarrow (\lambda x.fx)t$.
3. There are no problems with rules 0! and 1!.
4. For $f : \sigma + \tau \rightarrow \rho$, to apply the rule $f \rightarrow [f \cdot \kappa_1, f \cdot \kappa_2]$, it has to be an abstraction. We have the following.

   First, notice that
$$[f \cdot \kappa_1, f \cdot \kappa_2] \equiv [\lambda x : \sigma.f(\kappa_1 x), \lambda y^\tau.f(\kappa_2 y)]$$
   If $f \equiv [f_1, f_2]$, then
$$\begin{array}{l}
[\lambda x : \sigma.f(\kappa_1 x), \lambda y^\tau.f(\kappa_2 y)] \xrightarrow{\kappa} \\
[\lambda x : \sigma.f_1 x, \lambda y^\tau.f_2 y] \\
\leftarrow [f_1, f_2] \equiv f
\end{array}$$
   If f is not a co-pair,
$$\begin{array}{l}
[\lambda x : \sigma.f(\kappa_1 x), \lambda y^\tau.f(\kappa_2 y)] \equiv \\
[(\lambda x : \sigma + \tau.fx) \cdot \kappa_1, (\lambda x : \sigma + \tau.fx) \cdot \kappa_2 \xleftarrow{\pm!} \\
\lambda x : \sigma + \tau.fx \xleftarrow{\eta} f
\end{array}$$
   Since we use η in the last step, we may need to consider in this case that there may be a context involved; in fact,
$$ft \Rightarrow [f \cdot \kappa_1, f \cdot \kappa_2]t$$
   could be replaced by the following path
$$[(\lambda x : \sigma + \tau.fx) \cdot \kappa_1, (\lambda x : \sigma + \tau.fx) \cdot \kappa_2]t \leftarrow (\lambda x : \sigma + \tau.fx)t \rightarrow ft$$

5. $f \Rightarrow \langle \pi_1 f, \pi_2 f \rangle$, in case $f \equiv \langle f_1, f_2 \rangle$, or in a context like $C[(\pi_1 f)]$, the reduction can be replaced by the following respectively.

$$f \equiv \langle f_1, f_2 \rangle \multimap \langle \pi_1 \langle f_1, f_2 \rangle, \pi_2 \langle f_1, f_2 \rangle \rangle$$

and

$$\pi_1 f \multimap \pi_1 \langle \pi_1 f, \pi_2 f \rangle$$

This finishes the representation of $\Rightarrow$ by $\multimap$ or $\mathrel{-\!\!\circ}$. ⊣

Remark 18 Notice that we almost can make the following better claim: *Each occurrence of* $\Rightarrow$ *could be replaced by* $\stackrel{\bullet}{\rightarrow}$ *or* $\stackrel{\bullet}{\leftarrow}$ *without any mixed arrow sequent.* The mixed cases occur only in 2(b) and 4 above, and it is caused by the extensional rule $(+!)$ in case 4, and (η) in 2(b). This turns out to be crucial when we talk about confluence.

What we have proved is the following: both R and R^{∞} generate the same equational theory. As we will see this theory is exactly the equational theory of free the bi-ccc [24]. The equational theory generated from R^{-} is that of a free weak bi-ccc [24].

Although R is only $\mathcal{SN}$ not $\mathcal{CR}$, the equivalence relation generated is still a congruence:

Proposition 15 *For both equational theories, if* $\vdash M = N, \vdash K = S$, *then*

$$\vdash MK = NS, \vdash [M, K] = [N, S], \vdash \langle M, K \rangle = \langle N, S \rangle, \vdash \lambda x.M = \lambda x.N$$

provided the types check. Here $\vdash$ *is for both the equational theory of* R *(or* R^{∞}*) and that of* R^{-}.

The following is a proof sketch.

Let us notice that $\vdash M = N, K = S$ means there is a zigzag path linking M and N, K and S. To prove $\vdash MK = NS$ we only need to find such a zigzag path linking these two terms. Suppose we have a zigzag path linking M and N, and a similar case for K, S (see **Fig 1**). Now consider the above path with the starting point replaced by MK instead of only M, but the reductions are the same as above, i.e., we only reduce the subterm M. This will change the above $\rightarrow$ or $\leftarrow$ by $\Rightarrow$ or $\Leftarrow$ respectively, since we have got some context. Following the new path forward and backward we get to NK, then we do reductions only on K following the path linking K and S (see **Fig 2**). Thus we get a path linking MK and NS with each step $\Rightarrow$, or $\Leftarrow$.

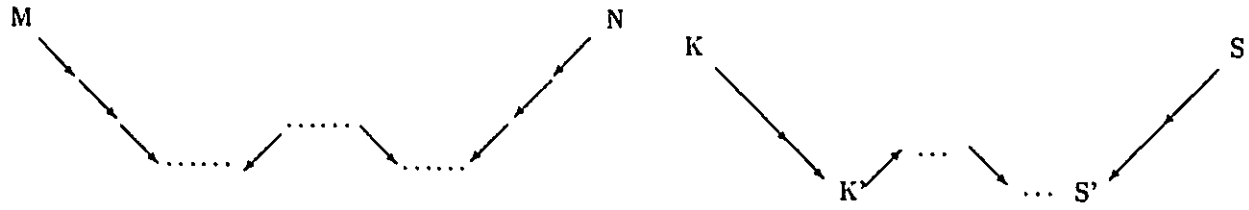


Fig 1

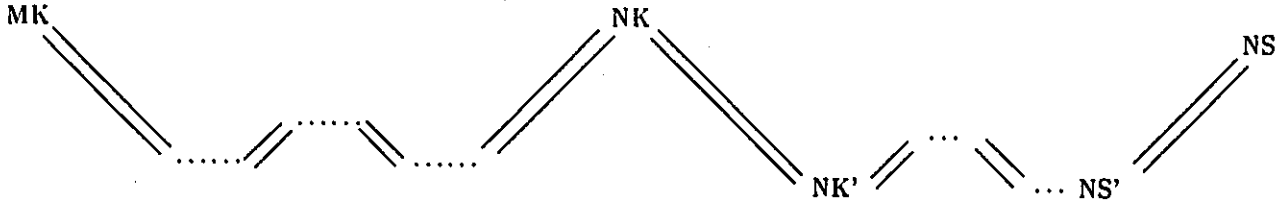


Fig 2

As we have shown every such step could be replaced by several restricted steps (i.e., replacing $\Rightarrow$ by $\rightarrow$ or $\leftarrow$). Hence, there is a restricted path linking MK and NS.

The same argument goes through for the other equalities. For the proof of the following theorem , we ask the readers to see [24].

Theorem 12 · *1. The equational theory generated from R is that of a free bi-ccc.*

2. The equational theory generated from R^- is that of a free connectionally closed category. *(For a precise definition, see [14]).*

For further related ideas, see [9][24][14].

6.3 $\mathcal{SN}, \mathcal{CR}$ and Faithful Representations

Let $\mathcal{L}$ be a typed calculus and R_L be a set of term rewriting rules, E_L the congruence relation generated from it,(hence an equational theory), and $\mathcal{C}_L^*$ the free open term model (hence a well-pointed category). If R_L satisfies $\mathcal{SN}$ and $\mathcal{CR}$, an easy argument shows that there is a faithful embedding from $\mathcal{C}_L$, the free category of closed terms into $\mathcal{C}_L^*$. In this section we study this and some related questions in detail. We will omit the subscripts below.

As we have said several times, for a lambda calculus, a rewrite system and a cartesian closed category describes its two aspects (syntax and semantics) respectively. It is natural to expect some nice property from one if the other has some "good behavior".

The following theorems are our contribution to this area. As far as we know, these results are new. We recall Section 2.5.3 and the category $\mathcal{C}^*$. Here $\mathcal{C}$ can be free (bi-)ccc or connectionally closed [14] .

Theorem 13 *If a rewriting theory satisfies $\mathcal{SN}$ and $\mathcal{CR}$, then the free category $\mathcal{C}$ can be faithfully embedded into the free model category $\mathcal{C}^*$.*

Proof We argue this by considering the correspondence between arrows and terms. See **Fig 3** for the idea. Let f, and g be two arrows and suppose they are equal in $\mathcal{C}$. This is equivalent to saying that there are two equivalence classes of terms and they are equal as sets. In particular, we can pick two closed terms t and s one from each of them. Since they are in the same class they are provably equal, i.e., they have the same normal form which is a closed term, of course. Now let us consider the behavior of these two terms in the class of all terms including open ones and the associated "open term" rewriting theory. Any rewriting theory for all terms includes, of course, those rules applicable to closed terms. Since there is at least one path linking these two terms (qua closed terms) via a normal form as we mentioned above, they are in the same equivalence class again. Under the correspondence there is only one arrow in the category $\mathcal{C}^*$ for them. So f and g are equal in $\mathcal{C}^*$, too.

Let $F : \mathcal{C} \longrightarrow \mathcal{C}^*$ be the canonical injection functor, we want to show that if closed terms t, s are not equal in $\mathcal{C}$, they are not equal in $\mathcal{C}^*$ either. In fact, since t, s are not equal in $\mathcal{C}$, there are reduction paths S, T which lead t to its normal form $\bar{t}$; and s to $\bar{s}$. Both $\bar{t}$ and $\bar{s}$ are irreducible, and distinct. Now inside $\mathcal{C}^*$, we perform T and S respectively, in particular, $\bar{t}$ and $\bar{s}$ are still normal forms and distinct since open term rewriting rules do not introduce new free variables, so they have no impact on closed terms. Thus t, s are not equal in $\mathcal{C}^*$ either. This gives a faithful embedding of the category $\mathcal{C}$ into $\mathcal{C}^*$.

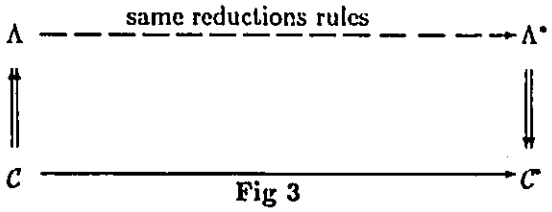


Fig 3

Example 23 Let $\mathcal{C}$ be the free ccc, and consider Mints' reduction rules. Cubric, in [7], showed this theory is $\mathcal{SN}$ and $\mathcal{CR}$. Hence $\mathcal{C}$ can be faithfuly embedded into $\mathcal{C}^*$.

Example 24 The theorem can also be applied on some weak λ-calculi provided the rewrite system is Church-Rosser and SN. For example

1. The simply typed lambda calculus with type 1 but without product, and the ordinary contraction rules.
2. The simply typed lambda calculus with product type, without type 1, and the ordinary contraction rules.
3. The calculus corresponding to the free connectionally closed category [14] but without type 1, and the expansion rewriting rules R^-.

As in the above theorem, let $\mathcal{S}$ be the set of all closed terms of a lambda calculus, e.g., simply typed lambda-calculus with or without (weak) coproduct types. Let $\mathcal{S}^*$ be the set of all terms (including variables and open ones) of the same calculus. Let R be the set of corresponding reduction rules applied to the above theory. Furthermore we suppose R satisfies $\mathcal{SN}$ only. Let $=$ be the equations generated from R. Let R^- be a subset of R, and E the equations generated by R^-. We assume R^- is $\mathcal{CR}$ (obviously, R^- is $\mathcal{SN}$) . We hope to complete the following diagram by another faithful embedding $\mathcal{S}/= \hookrightarrow \mathcal{S}^*/=$, where $\mathcal{S}/=$ means $=$ is restricted to the closed terms.

$$\begin{array}{ccc} \mathcal{S}/E & \hookrightarrow & \mathcal{S}^*/E \\ \downarrow & & \downarrow \\ \mathcal{S}/= & & \mathcal{S}^*/= \end{array}$$

Fig 4

Think of both $\mathcal{S}/=$ and $\mathcal{S}^*/=$ as categories, and the equivalence class $[t]_=$, as an arrow in $\mathcal{S}/=$ for any closed term t. Any two terms in $[t]_=$ are equal under $=$ (restricted to closed terms), i.e., there is a zigzag R-path linking them; any step in it is an occurrence of some rule from R. Now let us perform the rules R^- on the terms in $[t]_=$. There may no longer be R^- zigzag paths between each pair of terms in $[t]_=$. Hence the class $[t]_=$ splits into a set of E-classes. (Recall that E is the equational theory generated by R^-.) Thus, we get a set of normal forms (in the meaning of R^-) or arrows in $\mathcal{S}/E$. So $[t]_=$ equals $[t_1]_E \bigcup \cdots \bigcup [t_i]_E \bigcup \cdots$. This is because $R^- \subseteq R$. In category $\mathcal{S}/E$, the t_i are distinct arrows, so they are still distinct in $\mathcal{S}^*/E$ by Theorem 13 (because R^- is $\mathcal{SN}$ and $\mathcal{CR}$). Expand each closed-term class $[t_i]_E$ in $\mathcal{S}^*$ to the class $[t_i]^*_E$ (including open terms) such that it is an equivalence class with respect to the equivalence relation generated from E on $\mathcal{S}^*$. Thus we get an arrow $[t_i]^*_E$ for each t_i in $\mathcal{S}^*/=$. Mapping all these arrows $[t_i]^*_E$ down to $\mathcal{S}^*/=$, we get one $[t]^*$. We show such a correspondence $F : [t] \mapsto [t]^*$ is, in fact, a faithful embedding.

First, we need to show that $[t]^*$ is an equivalence class, i.e., all the terms in it represent the same arrow in $\mathcal{C}^*/=$; i.e., they are all equal with respect to the equations $=$. This is true since the same zigzag lines from $[t]_=$ and $[t_i]^*_E$ are still there.

It is an ordinary and tedious procedure to check that $F : \mathcal{S}/= \longrightarrow \mathcal{S}^*/=$ is a well-defined functor.

Now for a closed term t, we show $F : [t] \mapsto [t]_*$ is faithful from $\mathcal{S}/=$ to $\mathcal{S}^*/=$. Perform the reduction rules of R^- on each closed term of $[t]_*$. We get a set of closed terms $\{t_1, \cdots, t_i, \cdots\}$, all of them are in normal form. For any two of them there is a zigzag path (in R) connecting them since they all equivalent in $\mathcal{S}^*$. This path has each step from R, and all its nodes from $\mathcal{C}$. This says that all of them are equal in $\mathcal{S}/=$. Thus from each arrow in $\mathcal{S}^*/=$, we can only get back to one arrow in $\mathcal{S}/=$, hence F is faithful.

Let $\mathcal{C}^w$ be the free bi-ccc but with only weak co-product and weak initial, and R^- is the reduction rules as [9] or **Figure 3** in Chapter 2, which are confluent and strong normalizing. Let $=$ be the corresponding

rules saying that coproduct is strong and initial is strong (they happen to be strongly normalized, see [9]), we get the following theorem

Theorem 14 *There is a faithful embedding from the free bi-ccc into the free open term model of bi-ccc.*

This finishes our investigations in proof theory, category and λ-calculus.

Chapter 7

Some Future Work

In this thesis, we studied some properties of logical relations, logical relation categories and related results.

There are still lots of interesting subjects that have not been touched. For example, one is the full abstraction problem which is closely related to logical relations, [37]. Logical relations for the higher order (polymorphic) case is another one. One of the beautiful properties of logical relations is its hereditary nature; this, of course, depends on the hierarchical construction of simply typed structures. For higher order lambda calculus the variable types destroy the hierarchy.

I have not said much about logical relations for coproduct types. Coproduct types are very nasty to handle. It relates to understanding disjunctions. So far, the computability predicate **Comp** is the only non-trivial successful logical predicate for weak coproduct types.

Probably the main reason why people studied logical relations was to characterize lambda definability by the more familiar concepts, like functions or relations. There are only two published results which partially achieve this goal. One is the work of Plotkin [39], he used a kind of logical relation on more general Kripke models. The other is Jung and Tiuryn [17], developed from the former. In fact, only one logical relation (or predicate) is used in their works: *a term being definable*. We interpret these results to mean that definability is a logical property itself. Not much useful information is revealed. It would be better if there were a sequence of "approximating" relations, each doing more than its predecessor. Finally some kind of limit would reach the goal: *characterizing lambda definability*. This might lead people, step by step, to know something really deep about the nature of definability. However, Loader's theorem might suggest that there is no such result [28].

Statman's characterization theorem offers a ray of hope. It was stated by him in two ways; for a Henkin model $\mathcal{A}$,

1. An element of $\mathcal{A}$ is lambda definable iff it satisfies (is invariant under) all the logical relations on $\mathcal{A}$ [46].

2. An element of $\mathcal{A}$ is lambda definable iff it satisfies all the relations on any model $\mathcal{B}$ which extends $\mathcal{A}$ [47].

The first one is very interesting and promising because of the special connection between $\mathcal{A}$ and $\mathcal{A}^{\omega}$.

The soundness direction is always easy to show, even for the generalized basic lemma. Neither proof is published. We cannot make any more comment on this theorem. Finally, the problem of characterizing lambda definability (even on a full type hierarchy T_B) with ordinary relations is still open.

Bibliography

[1] Abramsky, S., Abstract Interpretation, Logical Relations and Kan Extensions., Department of Computer Science, Imperial College, London. preprint.

[2] Akama, Y., Mints Reductions and CCC-Calculus, In: *Typed Lambda Calculus and Applications 93* Vol.664, Springer Lecture Notes in Computer Science (1993)

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

[4] Backus, J, Turing Award Address In: *Comm of the ACM* No 7, 1976

[5] Barendregt, H.P., *The Lambda Calculus: Its Syntax and Semantics* (North-Holland, Amsterdam, 1984).

[6] Birkhoff, G and J.Lipson, Heterogeneous Algebras, In:*J. Combinatorial Theory*, 8 (1970), pp. 115 - 133.

[7] Cubric, D. , Some Results on Categorical Proof Theory, Ph.D Thesis, McGill University, 1993.

[8] Curry, H.B. and R. Feys, *Combinatory Logic I* (North-Holland, Amsterdam, 1958).

[9] Dougherty,D.J., Some Lambda Calculi with Categorical Sums and Products, In: Springer Lecture Notes in Computer Science, Vol.690, (1993)

[10] Freyd, P. and A. Scedrov *Allegories, Categories* North-Holland, 1991.

[11] Friedman, H., Equality between Functionals, in *Logic Colloquium*, Springer LNM 453, ed by R. Parikh, (1972), pp.22-37.

[12] Gandy, R.O., On the Axiom of Extensionality, *Journal of Symbolic Logic* **21**, (1956),

[13] Girard, J.Y., Y. Lafont and P.Taylor, *Proofs and Types*, Cambridge Tracts in *Theoretical Computer Science* Vol.7, (1989).

[14] Harnik, V. and M. Makkai, Lambek's Categorical Proof theory and Läuchli's Abstract Realizability, *Journal of Symbolic Logic* **57**, (1992), pp. 200-230.

[15] Hindley, J.R. and J.P.Seldin, *Introduction to Combinators and lambda-calculus*. London Mathematical Society Student Texts **1**. Cambridge University Press, 1986.

[16] Jay,C.B., Long $\beta\eta$-normal Forms and Confluence Technical Report, University of Edinburgh, 1992

[17] Jung, A. and J.Tiuryn, A New Characterization of Lambda Definability, *Types Lambda Calculi and Applications* Vol 664, Lecture Notes in Computer Science, pp230-244, 1993.

[18] Lambek, J., Deductive Systems and Categories I, *J. Math. Systems Theory 2*, pp 278-318, 1968

[19] Lambek, J., Deductive Systems and Categories II, Springer LNM 86, pp 76-122, 1969

[20] Lambek, J., Deductive Systems and Categories III, Springer LNM 274, pp 57-82, 1972

[21] Lambek, J., From λ-calculus to Cartesian Closed Categories in: *To H.B.Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* (Academic Press, New York, 1980), pp 375-402

[22] Lambek, J., Some Aspects of Categorical Logic, preprint, McGill University (1992)

[23] Lambek, J., Relations in Categories, preprint, McGill University (1995)

[24] Lambek, J. and P.J. Scott, *Introduction to Higher Order Categorical Logic*, Cambridge University Press, (1986).

[25] Läuchli, H., An Abstract Notion of Realizability for which Intuitionistic Predicate Calculus is Complete, in *Intuitionism and Proof Theory*, edited by A. Kino, J. Myhill, and R.E.Vesley North-Holland,(1970), pp. 227-234.

[26] Lawvere, F.W. , An Elementary Theory of the Category of Sets, *Proc. Nat. Acad. Sci*, USA **52**, pp. 1506-11

[27] Lawvere, F.W. , The Category of Categories as a Foundation for Mathematics, *Proceedings of the Conference on Categorical Algebra*, 1966

[28] R. Loader, The Undecidability of λ-definability, In *The Church Festschrift* ed by M.Zeleny 1993.

[29] Ma, Q. and J. Reynolds, Types, Abstraction and Parametric Polymorphism II, In: *Lecture Notes in Computer Science* 598, (Springer-Verlag, 1992).

[30] MacLane, S., *Categories for the Working Mathematician*, Graduate Text in Mathematics, Vol.5 (Springer, Berlin, 1971)

[31] Martin-lof, P., Constructive Mathematics and Computer Science in *Proc. 6th Internat. Congr. for Logic, Methodology, and Philosophy of Science* (North-Holland, Amsterdam, 1982) 153-175.

[32] Meyer, A.R., What is a model of the lambda calculus, *Inform. and Control* **52**(1) (1982) 87 - 122.

[33] Mitchell, J.C, Type Systems for Programming Languages, in: *Handbook of Theoretical Comp. Science*, Elsevier Publishers, (1990). pp. 366-458.

[34] Mitchell, J.C. and E. Moggi, Kripke-style models for typed lambda calculus. *Annals Pure Appl. Logic*,**51**, (1991), pp. 99-124.

[35] Mitchell, J.C. and A. Scedrov, Notes on Sconing and Relators, In: *Proceedings of Computer Science Logic*, Springer Lecture Notes in Computer Science, Vol 702, (1993)

[36] Mitchell, J.C. and P.J. Scott, Typed Lambda Models and Cartesian Closed Categories, in: *Contemporary Mathematics* Vol.92, (1989), pp. 301-316.

[37] Mulmuley, K., *Full Abstraction and Semantic Equivalence*. The MIT Press, Cambridge, Mass., 1987.

[38] Plotkin, G.D., LCF Considered as a Programming Language, in: *Theoretical Computer Sci* **5** (1977) pp.223-255.

[39] Plotkin, G.D., Lambda Definability in the Full Type Hierarchy, in: *To H.B. Curry, Essays on Combinatory Logic, Lambda Calculus and Formalism*, (1980), pp. 363-373.

[40] Prawitz, D., *Natural Deduction*(Almquist and Wiksell, Stockholm, 1965).

[41] Reynolds, J., Types, Abstraction and Parametric Polymorphism, *Information Processing '83*, North-Holland, (1983).

[42] Scott, D.S., Continuous Lattices, Springer LNM 274, pp.97-136, 1972

[43] Scott, D.S., Data Types as Lattices, SIAM J. Computing 5, pp. 522-87, 1976

[44] Scott, D.S., Relating Theories of the Lambda Calculus, in: *To H.B.Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* (Academic Press, New York, 1980) pp.403-450.

[45] Statman, R., Completeness, Invariance and λ-definability, in: *Journal of Symbolic Logic* **47**(1) (1982), pp. 17-26.

[46] Statman, R., Logical Relations and the Typed Lambda Calculus, *Information and Control* **65**, (1985), pp. 85-97.

[47] Statman, R., Equality between functionals revisited, in: *Harrey Friedman's Research in Mathematics Fundation*, (1985), pp331-338.

[48] Stoughton, A., Mechanizing Logical Relations, in: *Proc. Mathematical Foundations of Programming Semantics Lecture Notes in Computer Science* 802, (Springer-Verlag, 1994).

[49] Stoy, J.E., *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory* (MIT Press, Cambridge, MA, 1977)

[50] Tait,W.W., Intensional Interpretation of Functionals of Finite Type, *Journal of Symbolic Logic* **32**, (1967), pp. 198-212.

[51] Troelstra, A.S., *Mathematical Investigation of Intuitionistic Arithmetic and Analysis*, Lecture Notes in Mathematics, Vol. 344 (Springer, Berlin, 1973).

[52] Volger, H., Logical Categories, Semantical Categories and Topoi, *Lecture Notes in Mathematics*, Vol. 445 (Springer, Berlin, 1975), pp. 511-100.