Closed Sets of Higher-Order Functions

MSc Thesis *(Afstudeerscriptie)*

written by

Evan Marzion
(born August 20, 1992 in West Allis, Wisconsin, USA)

under the supervision of **Dr. Piet Rodenburg**, and submitted to the Board of Examiners in partial fulfillment of the requirements for the degree of

MSc in Logic

at the *Universiteit van Amsterdam.*

Date of the public defense:	**Members of the Thesis Committee:**
June 28th, 2016	Dr. Maria Aloni
	Dr. Benno van den Berg
	Dr. Piet Rodenburg
	Prof. Dr. Ronald de Wolf

Institute for Logic, Language and Computation

Abstract

In universal algebra, clones may be viewed as a way of studying definability between functions within the presence of certain natural operations, namely projection and composition. We show how the simply typed lambda calculus provides a suitable framework for extending this study to higher-order functions as well. We define what we call a combinatory clone, a higher-order analogue of regular clones, and establish some basic results about them. Inspired by Post's classification of the boolean clones, boolean combinatory clones are studied. Finally, we consider an extension of the simply typed lambda calculus with product types, and show how they do not affect anything from the point of view of combinatory clones.

Acknowledgements

First and foremost, I would like to thank my supervisor Piet Rodenburg for all of the work he put into advising me on this thesis. He encouraged me to devise and study a problem that had almost no previous work done on it, when it would have been perfectly understandable for him to instead suggest that I head down a more well-trodden path. I am grateful for all of the helpful comments, suggestions, assistance, and words of encouragement he offered me over the course of our weekly meetings. All in all, writing this thesis was a pleasant experience, and I believe that much of that is owed to having a supervisor like Piet.

I would like to also thank the three other members of my committee, Maria Aloni, Benno van den Berg, and Ronald de Wolf, for taking time out of their no-doubt busy schedules to read through this thesis and offer their criticisms.

Lastly, I would like to thank my parents Mark and Wendy for all the support they have given me over these last two years (and all the support they gave in the years before that, for that matter).

Contents

Introduction

Motivation

In mathematical logic and computer science, lambda calculi have been extensively studied for their expressive power as systems in which to do mathematics and computation. In computability theory, the untyped system is one of the most well-known Turing-complete models of computation. In type theory, typed lambda calculi have been developed as foundational systems for constructive mathematics. In programming language theory, both typed and untyped lambda calculi serve as the basis for the design of many programming languages, especially those falling under the functional paradigm.

From a less foundational perspective, one may view a system such as the simply-typed lambda calculus as an elegant and flexible notational system for defining new functions from old, especially in the case of higher-order[1] functions. As an example, consider the operation of pointwise addition on functions. Given some algebraic structure $(R, +)$ and X some set, we can define pointwise addition on R^X with some equation like $(f \dot{+} g)(x) := f(x) + g(x)$. Using the lambda calculus, we would write something like

$$\dot{+} := \lambda f^{X \to R} g^{X \to R} x^X . f(x) + g(x).$$

Conceptually, this has the advantage of clarifying precisely what the free variables f, g, x are doing in the above expression, while also isolating $\dot{+}$ as an entity itself in the domain $R^X \times R^X \to R^X$.

While this example may not be terribly impressive, suppose that we take the level of abstraction one step further:

$$\texttt{pointwise} := \lambda +^{R \times R \to R} f^{X \to R} g^{X \to R} x^X . f(x) + g(x).$$

Already at this point, mathematicians are arguably more likely to think of this more as a pattern of construction ("every binary operation on R can naturally be made into a binary operation on R^X") than as an object in its own right ("there is an operation in the domain $(R \times R \to R) \to (R^X \times R^X \to R^X)$ such

[1] Functions which accept other functions as arguments.

that..."). We can then see the benefit of lambda notation in defining higher-order operations which mathematicians would otherwise have difficulty writing out in full.

When we say that we are taking a less foundational perspective, we mean that we take for granted the existence of sets and functions. In the above example, for instance, we do not concern ourselves with the existence of the operation + (or the existence of R,X, or even function spaces, for that matter). We only concern ourselves with how functions may be combined using the simply-typed lambda calculus in order to define other functions. To put it another way, we are interested in *relative* definability between functions, elements, and higher-order operations, as opposed to *absolute* definability of these objects within some foundational framework.

This suggests a study of the simply-typed lambda calculus as a sort of "algebra of functions". One of the earliest works which studied the interdefinability of functions within a notational framework was Post's classification of the boolean clones[5]. Post was motivated by questions concerning the interdefinability of connectives in classical propositional logic. Since classical propositional logic is complete with respect to the two-element boolean algebra, this question could naturally be rephrased in terms of finitary operations on the two-element boolean domain. While the language of propositional logic naturally suggests a sort of notational system (sentence letters as boolean variables, connectives as functions being applied to expressions containing these variables), it is not immediately obvious how this notion of definability can be captured algebraically. For instance, if we have some ternary connective $C(x, y, z)$, it is intuitively clear that we can define from it the binary connective $C(x, y, x)$. However, it is not necessarily clear what steps were needed to derive a definition of the second from the first. A satisfactory answer to this question comes from the notion of a clone, a set of finitary operations which contain the projections and is closed under composition.

By virtue of currying, we may always view functions of the form $X^k \to X$ as also taking the form $X \to \ldots \to X \to X$. From this perspective, clones represent just the non-higher-order portion of the functions represented by the simply-typed lambda calculus. We then endeavor to extend the study of clones to include these higher-order functions as well.

Overview of this work

In Chapter 1, we review basic definitions and facts about clones and the simply-typed lambda calculus. In Chapter 2, we establish notions of definability between higher-order functions. As in the case of clones, there are two equivalent notions: one notational (given by the lambda calculus) and one algebraic (what

we call a combinatory clone, based on combinatory logic). Basic results on combinatory clones and how they relate to regular clones are established. In Chapter 3, combinatory clones over finite sets are studied. In Chapter 4, we tackle the specific problem of classifying boolean combinatory clones, with an eye toward a classification of the coatomic clones similar to Post's. In Chapter 5, we briefly consider the problem of extending combinatory clones with additional type constructors.

Chapter 1

Preliminaries

1.1 Clones

Fix a family of sets $\mathbf{X} = (X_\alpha)_{\alpha \in \mathcal{A}}$.

Definition 1.1.1. By a **clone** over $\mathbf{X}$, we shall mean a collection $\mathcal{C}$ of functions of the form $X_{\alpha_1} \times \ldots \times X_{\alpha_n} \to X_\beta$ such that

1. Every projection function $\pi_i^n : X_{\alpha_1} \times \ldots \times X_{\alpha_n} \to X_{\alpha_i}$ given by the rule

$$\langle x_1, \ldots, x_n \rangle \mapsto x_i$$

   is in $\mathcal{C}$

2. If $f : X_{\beta_1} \times \ldots \times X_{\beta_m} \to X_\gamma$ is in $\mathcal{C}$ and $g_1, \ldots, g_m$ are functions in $\mathcal{C}$ of type $g_i : X_{\alpha_1} \times \ldots \times X_{\alpha_n} \to X_{\beta_i}$ for each i, the composed function $f \circ (g_1, \ldots, g_m) : X_{\alpha_1} \times \ldots \times X_{\alpha_n} \to X_\gamma$ given by the rule

$$\langle x_1, \ldots, x_n \rangle \mapsto f\left(g_1(x_1, \ldots, x_n), \ldots, g_m(x_1, \ldots, x_n)\right)$$

   is in $\mathcal{C}$.

Note that our definition of a clone differs from the standard presentation in two ways: First, we allow for the possibility of multiple sorts, whereas clones are traditionally taken to be over a single set. Second, constants or nullary functions are included, whereas they are often omitted in standard presentations.

It is easy to see that the clones over $\mathbf{X}$ always form a bounded lattice, which we denote by $\mathbf{Cl}(\mathbf{X})$. We shall denote the top (the set of all finitary operations over $\mathbf{X}$) and bottom (the set of all projections) clones by $\mathcal{C}(\mathbf{X})$ and $\Pi(\mathbf{X})$, respectively.

1.2 Boolean Clones

Let $\mathbf{B} = \{\bot, \top\}$ be the two-element boolean domain. In [5], Post gives a complete characterization of the lattice of the clones[1] over $\mathbf{B}$. He shows that the lattice of boolean clones is countable and has a simple structure. He also shows that every clone is finitely generated. In particular, he describes the five coatoms of the lattice, which we list here:

1.2.1 False-Preserving Functions

A function $f : \mathbf{B}^k \to \mathbf{B}$ is said to be **false-preserving** if

$$f(\bot, \ldots, \bot) = \bot.$$

A basis for the false-preserving functions is given by $\{\vee, \oplus\}$ (where $\oplus$ denotes exclusive disjunction). If nullary functions are considered, the constant $\bot$ must be added.

1.2.2 True-Preserving Functions

The **true-preserving** functions are similarly defined. A basis for them is given by $\{\wedge, \to\}$, and of course we must include $\top$ if nullary functions are considered.

1.2.3 Monotone Functions

Give $\mathbf{B}$ the obvious order with $\bot \leq \top$. We can then endow $\mathbf{B}^k$ with the product ordering. A function $f : \mathbf{B}^k \to \mathbf{B}$ is then said to be **monotone** if

$$\mathbf{x} \leq \mathbf{y} \Rightarrow f(\mathbf{x}) \leq f(\mathbf{y}).$$

A basis for the monotone functions is given by $\{\vee, \wedge, \bot_1, \top_1\}$, the latter two being the unary constant functions. If nullary functions are considered, we may replace the unary constant functions with their actual constants.

1.2.4 Self-dual Functions

A function $f : \mathbf{B}^k \to \mathbf{B}$ is said to be **self-dual** if

$$f(\neg x_1, \ldots, \neg x_k) = \neg f(x_1, \ldots, x_k).$$

A basis for the self-dual functions is given by $\{\neg, \mathbf{Maj}\}$, where **Maj** denotes the ternary majority function:

[1] Although he chose to omit nullary functions from his definition of a clone.

$$\mathbf{Maj}(x, y, z) := (x \wedge y) \vee (x \wedge z) \vee (y \wedge z).$$

1.2.5 Affine Functions

If $\mathbf{v}, \mathbf{w}$ are k-length boolean vectors, their dot product $\mathbf{v} \cdot \mathbf{w}$ is given by $\bigoplus_{i=1}^{k} v_i \wedge w_i$.

A function $f : \mathbf{B}^k \to \mathbf{B}$ is then said to be **affine** if there is a $\mathbf{v} \in \mathbf{B}^k$ and $b \in \mathbf{B}$ such that

$$f(\mathbf{x}) = \mathbf{v} \cdot \mathbf{x} \oplus b.$$

A basis for the affine functions is given by $\{\oplus, \top_1\}$. If nullary functions are considered, we may replace $\top_1$ with just $\top$.

1.3 The Typed Lambda Calculus

We briefly provide a formulation of the simply-typed lambda calculus and recall some basic facts about it. Since we will largely be using the lambda calculus as a means of denoting functions and elements, our focus will lie more on the semantic side of things. For a more thorough overview of typed lambda calculus, especially with respect to syntactic matters, the reader may consult [1].

1.3.1 Syntax

Types

Given a set of atomic types $\mathcal{A}$, we let $\mathcal{T}$ denote the set of types freely generated by $\mathcal{A}$ and $\to$:

$$\mathcal{T} ::= \mathcal{T} \to \mathcal{T} \mid \alpha \ \ (\alpha \in \mathcal{A}).$$

Convention 1.3.1. We take $\to$ to be right-associative. That is, $\alpha \to \beta \to \gamma$ will stand for $\alpha \to (\beta \to \gamma)$, and more generally $\sigma_1 \to \sigma_2 \to \ldots \sigma_{n-1} \to \sigma_n$ will stand for $\sigma_1 \to (\sigma_2 \to (\ldots (\sigma_{n-1} \to \sigma_n)))$.

The follow fact about types is simple to show, but useful enough that it warrants statement:

Claim 1.3.2. *Let $\sigma \in \mathcal{T}$. Then there is a natural number N (possibly zero, in the case that σ is atomic), types $\sigma_1, \ldots, \sigma_N \in \mathcal{T}$, and $\alpha \in \mathcal{A}$ such that*

$$\sigma = \sigma_1 \to \ldots \to \sigma_N \to \alpha.$$

Terms

To every $\sigma \in \mathcal{T}$ we assume an inexhaustible supply of variables $x^\sigma, y^\sigma, z^\sigma \ldots$. We then build up terms inductively in the following manner:

1. Each variable x^σ is a term of type σ.
2. If f is a term of type $\sigma \to \tau$ and s is a term of type σ, then fs is a term of type τ.
3. If t is a term of type τ, then $\lambda x^\sigma.t$ is a term of type $\sigma \to \tau$.

If t is a term of type τ, we shall then write $t : \tau$. Free variables and bound variables are defined in the standard fashion, and terms without free variables will be called closed terms.

There are two ways to simplify a term which preserve the inherent meaning of the term:

- The **beta reduction:** $(\lambda x.M)N \rightsquigarrow_\beta [N/x]M$.
- The **eta reduction:** $\lambda x.Mx \rightsquigarrow_\eta M$ (when x is not free in M).

Both reductions may also occur within subterms. We will more generally use $T \rightsquigarrow_\beta T'$ and $T \rightsquigarrow_\eta T'$ to mean that T' can be obtained from T by a finite sequence of beta (eta) reductions. If T' can be obtained from T by a sequence of both types of reductions, we write $T \rightsquigarrow_{\beta\eta} T'$.

1.3.2 Set-based Semantics

Let $\mathbf{X} = (X_\alpha)_{\alpha \in \mathcal{A}}$ be an assignment of set domains to each atomic type. We may inductively define the set domain $\mathbf{X}_\sigma$ for arbitrary $\sigma \in \mathcal{T}$:

$$\begin{aligned} \mathbf{X}_\alpha &:= X_\alpha, \quad \alpha \in \mathcal{A} \\ \mathbf{X}_{\sigma\to\tau} &:= \mathbf{X}_\tau^{\mathbf{X}_\sigma}. \end{aligned}$$

If we have that $s \in \mathbf{X}_\sigma$, we will equivalently write $s : \sigma$.

Let g be a partial mapping of variables x^σ into $\mathbf{X}_\sigma$ (for arbitrary $\sigma \in \mathcal{T}$). By $g[y^\tau \mapsto T]$ we shall mean the mapping which results from modifying g in order to send the variable y^τ to the element $T \in \mathbf{X}_\tau$.

Given a partial variable mapping g and an arbitrary term $t : \tau$, we assign to it an element $[\![t]\!]_g \in \mathbf{X}_\tau$ in the following inductive manner:

1. $[\![x^\sigma]\!]_g := g(x)$
2. If $f : \sigma \to \tau$ and $s : \sigma$, then $[\![fs]\!]_g := [\![f]\!]_g([\![s]\!]_g)$

3. If $t : \tau$, then $[\![\lambda x^\sigma.t]\!]_g$ is the function given by the rule

$$a \mapsto [\![t]\!]_{g[x^\sigma \mapsto a]}.$$

Convention 1.3.3. We will employ the symbol $\boldsymbol{\lambda}$ as a function-defining operator at set level, whereby the third rule above can be given as

$$[\![\lambda x.t]\!]_g := \boldsymbol{\lambda} a.[\![t]\!]_{g[x \mapsto a]}.$$

As we said before, beta and eta reductions preserve the meaning of terms. We make this claim explicit:

Claim 1.3.4. *Suppose $M \rightsquigarrow_{\beta\eta} N$. Then for any $\mathbf{X}$ and g we have that $[\![M]\!]_g = [\![N]\!]_g$.*

1.3.3 Long Normal Forms

It will occasionally be useful to assume that our lambda terms are in certain syntactic normal forms.

Definition 1.3.5. Let $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \alpha$, α atomic. We say that $s : \sigma$ is in **long normal form** if $s = \lambda x_1^{\sigma_1} \ldots x_n^{\sigma_n}.vT_1 \ldots T_m$ for v some variable (either free or one of the x_i's) and $T_1, \ldots, T_m$ each in long normal form.

Claim 1.3.6. *For every $s : \sigma$ there is an $s' : \sigma$ such that s' is in long normal form and for all assignments g we have that $[\![s]\!]_g = [\![s']\!]_g$.*

Proof. See [1]. □

1.3.4 Combinatory Completeness

Given types σ, τ, ρ, the so-called K- and S-combinators are defined as follows:

$$\mathsf{K}_{\sigma,\tau} := \lambda x^\sigma y^\tau.x : \sigma \to \tau \to \sigma$$

and

$$\mathsf{S}_{\sigma,\tau,\rho} := \lambda x^{\sigma\to\tau\to\rho} y^{\sigma\to\tau} z^\sigma.xz(yz) : (\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho.$$

It is a classic result that the K- and S-combinators form a basis for the closed terms of the simply-typed lambda calculus. To be precise:

Claim 1.3.7. *Let $t : \tau$ be a closed term. Then there is a $t' : \tau$ built by repeated application of the K's and S combinators such that for any $\mathbf{X}$, $[\![t]\!] = [\![t']\!]$.*

Throughout the remainder of this work, we shall let $\mathbf{K}_{\sigma,\tau} := [\![\mathsf{K}_{\sigma,\tau}]\!]$ and $\mathbf{S}_{\sigma,\tau,\rho} := [\![\mathsf{S}_{\sigma,\tau,\rho}]\!]$.

Chapter 2

Combinatory Clones

We are interested in formulating a notion of definability between elements and functions within $\bigcup_{\sigma \in \mathcal{T}} \mathbf{X}_\sigma$. Intuitively, when we say that t is definable from a set of elements S, we mean that there is a lambda term T which uses $s_1, \ldots, s_n \in S$ as variables which represents t. To be more precise:

Definition 2.0.1. Let $s_1 \in \mathbf{X}_{\sigma_1}, \ldots, s_n \in \mathbf{X}_{\sigma_n}, t \in \mathbf{X}_\tau$. We say that t is **definable** from $s_1, \ldots, s_n$ if there is a lambda term $T : \tau$ with free variables $x_1^{\sigma_1}, \ldots, x_n^{\sigma_n}$ such that

$$[\![T]\!]_{[x_1 \mapsto s_1, \ldots, x_n \mapsto s_n]} = t.$$

Note that as a consequence that the free variables of T must be limited to $x_1, \ldots, x_n$.

When establishing definability, we will not be so pedantic:

Convention 2.0.2. When giving definitions between elements, we will employ notation which freely mixes the syntax of the lambda calculus with names for elements in $\mathbf{X}$. For example, if we wished to show that boolean conjunction is definable from boolean disjunction and boolean negation, we will write out something like

$$\vee = \lambda xy.\neg(\neg x \wedge \neg y)$$

when technically we should write something like

$$\vee = [\![\lambda xy.f(g(fx)(fy))]\!]_{[f \mapsto \neg, g \mapsto \wedge]}.$$

We immediately obtain the following result which allows us to control the occurrences of S in a definition of t:

Lemma 2.0.3. *Let $t \in \mathbf{X}_\tau$ be definable from $s_1 \in \mathbf{X}_{\sigma_1}, \ldots, s_n \in \mathbf{X}_{\sigma_n}$. Then there is a closed lambda term (in the strict sense that it contains no variables representing the s_i) $M : \sigma_1 \to \ldots \to \sigma_n \to \tau$ such that*

$$t = Ms_1 \dots s_n.$$

Proof. Let $T : \tau$ represent t with occurrences of $s_1, \dots, s_n$ represented by free variables $x_1, \dots, x_n$, respectively. We can then capture these variables in lambda abstractions to obtain M:

$$M := \lambda x_1^{\sigma_1} \dots x_n^{\sigma_n}.T.$$

□

As in the case of standard clones, we seek a second notion of definability which is more algebraic in nature. Our first notion based on lambda notation has the advantage that it allows us to easily write out definitions between elements. However, it is difficult to reason inductively about the structure of definitions. The fundamental difficulty comes from lambda abstraction: if we look at a closed term $\lambda x.T$, we pass from a (potentially) un-closed term (T) to closed one ($\lambda x.T$). So if we wish to prove something along the lines of "all functions definable from the class of functions C have property P" and $\lambda x.T$ denotes such a function, we will not be able to apply our inductive hypothesis on the term T, since it no longer denotes a function. At best, we could attempt to rephrase the result over arbitrary terms with free variables, but this is messy and complicates matters.

Instead, we shall develop a second notion of definability based on combinators, which is equivalent to our first notion by virtue of combinatory completeness.

2.1 Definitions and Basic Results

Given a set of functions and elements $\mathcal{G} \subseteq \bigcup_{\sigma \in \mathcal{T}} \mathbf{X}_\sigma$, let $\mathcal{G}_\sigma := \mathcal{G} \cap \mathbf{X}_\sigma$. That is, $\mathcal{G}_\sigma$ is the set of elements of $\mathcal{G}$ which have type σ.

Definition 2.1.1. A **combinatory clone** over $\mathbf{X}$ is a set of functions and elements $\mathcal{G} \subseteq \bigcup_{\sigma \in \mathcal{T}} \mathbf{X}_\sigma$ such that

1. for all $\sigma, \tau \in \mathcal{T}$ the function $\mathbf{K}_{\sigma,\tau} \in \mathcal{G}_{\sigma \to \tau \to \sigma}$
2. for all $\sigma, \tau, \rho \in \mathcal{T}$ the function $\mathbf{S}_{\sigma,\tau,\rho} \in \mathcal{G}_{(\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho}$
3. for all $\sigma, \tau \in \mathcal{T}$, if $f \in \mathcal{G}_{\sigma \to \tau}$ and $s \in \mathcal{G}_\sigma$, then $fs \in \mathcal{G}_\tau$.

Definition 2.1.2. Given $S \subseteq \bigcup_{\sigma \in \mathcal{T}} \mathbf{X}_\sigma$, the **closure** of S, denoted by $\overline{S}$, is the least combinatory clone containing S.

Definition 2.1.3. $\mathcal{B} \subseteq \bigcup_{\sigma \in \mathcal{T}} \mathbf{X}_\sigma$ is said to be a **basis** for a combinatory clone $\mathcal{G}$ if $\overline{\mathcal{B}} = \mathcal{G}$.

Lemma 2.1.4. *For any $t \in \mathbf{X}_\tau$ and $S \subseteq \bigcup_{\sigma \in \mathcal{T}} \mathbf{X}_\sigma$, t is definable from S if and only if $t \in \overline{S}$.*

Proof. $\Rightarrow$ Let t be definable from S. By lemma 2.0.3, there is a closed term M and elements $s_1, \dots s_n \in S$ such that

$$t = M s_1 \dots s_n.$$

Since M is closed, it follows that there is an M' built up from K- and S-combinators which is extensionally equivalent. Thus,

$$t = M' s_1 \dots s_n.$$

We see now that t can be derived solely by application of the K- and S-combinators to some elements of S. Thus, $t \in \overline{S}$.

$\Leftarrow$ Let $t \in \overline{S}$. We induct on the membership of t in $\overline{S}$: If $t \in S$, then t represents itself. If t is either $\mathbf{K}_{\sigma,\tau}$ or $\mathbf{S}_{\sigma,\tau,\rho}$, then it is represented by $\mathsf{K}_{\sigma,\tau}$, $\mathsf{S}_{\sigma,\tau,\rho}$, respectively. Finally if $t = t_1 t_2$, by inductive hypothesis, t_1, t_2 are represented by terms T_1, T_2, respectively. Thus, t is represented by $T_1 T_2$. □

2.1.1 The Lattice of Combinatory Clones

Let $\mathcal{G}(\mathbf{X}) := \bigcup_{\sigma \in \mathcal{T}} \mathbf{X}_\sigma$. In the case that $\mathcal{A}$ contains a single atomic type (which we shall always denoted by 0) and $\mathbf{X}$ consists of the single set X, we will make a slight abuse of notation and write $\mathcal{G}(X)$.

Claim 2.1.5. *$\mathcal{G}(\mathbf{X})$ is a combinatory clone.*

Claim 2.1.6. *Let $\mathcal{G}, \mathcal{H}$ be combinatory clones. Then $\mathcal{G} \cap \mathcal{H}$ is a combinatory clone.*

Definition 2.1.7. Given $\mathcal{G}, \mathcal{H}$ combinatory clones, let $\mathcal{G} \vee \mathcal{H} := \overline{\mathcal{G} \cup \mathcal{H}}$.

Definition 2.1.8. Let $\Lambda(\mathbf{X}) := \overline{\{\mathbf{K}_{\sigma,\tau}\}_{\sigma,\tau \in \mathcal{T}} \cup \{\mathbf{S}_{\sigma,\tau,\rho}\}_{\sigma,\tau,\rho \in \mathcal{T}}}$, the lambda-definable functions.

Let $\mathbf{CCl}(\mathbf{X})$ denote the set of all combinatory clones over $\mathbf{X}$.

Claim 2.1.9. *$\mathbf{CCl}(\mathbf{X})$ is a bounded lattice when ordered by inclusion, with top element given by $\mathcal{G}(\mathbf{X})$, bottom element given by $\Lambda(\mathbf{X})$, meets given by $\cap$, and joins given by $\vee$.*

2.2 Relating Clones to Combinatory Clones

Definition 2.2.1. The **flat** types $\mathcal{F}$ are those which can be generated by the following grammar:

$$\mathcal{F} ::= \alpha \mid \alpha \to \mathcal{F} \quad (\alpha \in \mathcal{A})$$

Any element of $\mathcal{G}(\mathbf{X})$ with flat type can naturally be thought of as an element of $\mathcal{C}(\mathbf{X})$ through the typical uncurrying operation. Specifically, if $f : X_{\alpha_1} \to \ldots \to X_{\alpha_n} \to X_\beta$, there is a related function $\hat{f} : X_{\alpha_1} \times \ldots \times X_{\alpha_n} \to X_\beta$. We will not always be so explicit in differentiating these two functions, and will typically choose to conflate a function with its curried or uncurried form, when applicable.

Lemma 2.2.2. *Let $\mathcal{G} \in \mathbf{CCl}(\mathbf{X})$. Then the flat elements of $\mathcal{G}$ form a clone over $\mathbf{X}$.*

Proof. First, the projection functions $\pi_i^n : X_{\alpha_1} \times \ldots X_{\alpha_n} \to X_{\alpha_i}$ are given by the lambda term

$$\lambda x_1^{\alpha_1} \ldots x_n^{\alpha_n}.x_i.$$

If $f \in \mathcal{G}_{\beta_1 \to \ldots \to \beta_n \to \gamma}$ and $g_1, \ldots, g_n$ are such that $g_i \in \mathcal{G}_{\alpha_1 \to \ldots \to \alpha_m \to \beta_i}$, then the term

$$\lambda x_1^{\alpha_1} \ldots x_m^{\alpha_m}.f(g_1 x_1 \ldots x_m) \ldots (g_n x_1 \ldots x_m)$$

corresponds to $f \circ (g_1, \ldots, g_n)$. □

In light of this lemma, we give the following definition:

Definition 2.2.3. Let $\Phi : \mathbf{CCl}(\mathbf{X}) \to \mathbf{Cl}(\mathbf{X})$ denote the map which takes $\mathcal{G} \in \mathbf{CCl}(\mathbf{X})$ to the clone of its flat elements.

Of course, we also have a natural way to produce a combinatory clone from a standard clone: Given $\mathcal{C} \in \mathbf{Cl}(\mathbf{X})$, we may curry the elements of $\mathcal{C}$ and take the closure of the resulting set.

Definition 2.2.4. Let $\Psi : \mathbf{Cl}(\mathbf{X}) \to \mathbf{CCl}(\mathbf{X})$ denote the map which takes $\mathcal{C} \in \mathbf{Cl}(\mathbf{X})$ to $\overline{\mathcal{C}} \in \mathbf{CCl}(\mathbf{X})$.

Obviously, $\mathcal{C} \subseteq \Phi(\Psi(\mathcal{C}))$. Does the reverse inclusion hold? Perhaps the presence of the higher-order lambda notation allows us to define additional flat functions. Fortunately, we may use long normal forms to see that this isn't the case.

Lemma 2.2.5. *Let $T : \alpha$ (α an atomic type) be a term with free variables among $t_1, \ldots, t_n$ of flat type and $x_1 : \beta_1, \ldots, x_n : \beta_n$ of atomic type. For any $f_1, \ldots, f_n$ we have that the function*

$$[\![\lambda x_1^{\beta_1} \ldots x_n^{\beta_n}.T]\!]_{[t_1 \mapsto f_1, \ldots, t_n \mapsto f_n]}$$

belongs to the clone generated by $f_1, \ldots, f_n$.

Proof. By claim 1.3.6 we may suppose that T is in long normal form. We then induct on the structure of the term. There are two cases to consider: First, T could just be a variable of atomic type, in which case T after lambda abstractions will be interpreted as a projection function. Secondly, T could begin with a flat free variable t_i. We then have that $T = t_i T_1 \ldots T_m$ for $T_1, \ldots, T_m$ of atomic

type. By inductive assumption, $T_1, \ldots, T_m$ correspond to functions $g_1, \ldots, g_m$ in the clone generated by $f_1, \ldots, f_n$. We then see that

$$\begin{aligned}[\![\lambda x_1^{\beta_1} \ldots x_n^{\beta_n}.T]\!]_{[t_1 \mapsto f_1, \ldots, t_n \mapsto f_n]} &= [\![\lambda x_1^{\beta_1} \ldots x_n^{\beta_n}.t_i T_1 \ldots T_m]\!]_{[t_1 \mapsto f_1, \ldots, t_n \mapsto f_n]} \\ &= f_i \circ (g_1, \ldots g_m).\end{aligned}$$

□

Lemma 2.2.6. *$\Phi(\Psi(\mathcal{C})) \subseteq \mathcal{C}$ for any $\mathcal{C} \in \mathbf{Cl}(\mathbf{X})$.*

Proof. Suppose $f : X_{\beta_1} \times \ldots \times X_{\beta_n} \to X_\alpha \in \Phi(\Psi(\mathcal{C}))$. There is then a term F, free variables $t_1, \ldots, t_n$ and flat functions $f_1, \ldots, f_n \in \mathcal{C}$ such that

$$[\![F]\!]_{[t_1 \mapsto f_1, \ldots t_n \mapsto f_n]} = f.$$

Again, by 1.3.6 we may assume that F is of the form $\lambda x_1^{\beta_1} \ldots x_n^{\beta_n}.T$ for T of atomic type containing as free variables $t_1, \ldots, t_n$ and $x_1, \ldots x_n$. The claim then follows from the previous lemma. □

Corollary 2.2.7. *$\Phi(\Psi(\mathcal{C})) = \mathcal{C}$ for any $\mathcal{C} \in \mathbf{Cl}(\mathbf{X})$.*

2.3 Connections to Logic

Through the well-known "propositions as types" paradigm, many type theories can be seen as corresponding to certain systems of logic. We briefly recall the nature of this correspondence in the case of the simply-typed lambda calculus, and mention some applications to the study of combinatory clones.

2.3.1 Intuitionistic Implicational Logic

Let $\mathcal{L}$ denote the set of formulas built up from atomic sentences $\mathcal{A}$ and the connective $\to$. Intuitionistic Implicational Logic may be given by a Hilbert-style proof system over the language $\mathcal{L}$ with the inference rule modus ponens and the following two axiom schemes:

1. $\varphi \to (\psi \to \varphi)$.
2. $(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))$.

Formulas in $\mathcal{L}$ are naturally recast as types in $\mathcal{T}$ and vice-versa, and in fact we will entirely conflate the two from here on out. Axioms 1 and 2 correspond to the types of the K- and S-combinators, and modus ponens corresponds to function application.

Claim 2.3.1. *For any $\sigma \in \mathcal{T}$, σ is a tautology of intuitionistic implicational logic if and only there is a closed term $S : \sigma$.*

Claim 2.3.2. *Let $\mathcal{G}$ be a combinatory clone. Define $\mathcal{T}_{\mathcal{G}} := \{\sigma \mid \mathcal{G}_\sigma \neq \varnothing\}$. Then $\mathcal{T}_{\mathcal{G}}$ is a deductively closed theory.*

Claim 2.3.3. *Let $T \subseteq \mathcal{T}$ be a deductively closed theory. Then $\bigcup_{\tau \in T} \mathbf{X}_\tau$ is a combinatory clone.*

Proof. The inclusion of the K- and S-combinators is given by the fact that their types correspond to tautologies, and closure under application is given by the fact that T is closed under modus ponens. □

In light of this fact, we give the following definition:

Definition 2.3.4. For any deductively closed T, $\mathcal{G}^T(\mathbf{X})$ will denote the combinatory clone of elements with types in T. In particular, $\mathcal{G}^{\text{taut}}(\mathbf{X})$ will denote the combinatory clones of elements with types which are tautologies. Combinatory clones of this form will be called **logical**.

Claim 2.3.5. *$T \mapsto \mathcal{G}^T(\mathbf{X})$ is a lattice embedding.*

Definition 2.3.6. Let $\mathcal{G}$ be a combinatory clone. We say that $\mathcal{G}$ is **extensional** if for any $\sigma, \tau \in \mathcal{T}$ and $f, g \in \mathbf{X}_{\sigma\to\tau}$, if $f \in \mathcal{G}_{\sigma\to\tau}$ and $f|_{\mathcal{G}_\sigma} = g|_{\mathcal{G}_\sigma}$, then $g \in \mathcal{G}_{\sigma\to\tau}$.

Lemma 2.3.7. *Let $\mathcal{G}$ be extensional. Then $\mathcal{G}$ is either a logical combinatory clone, or it is covered by a logical combinatory clone.*

Proof. Clearly, $\mathcal{G} \subseteq \mathcal{G}^{\mathcal{T}_\mathcal{G}}$. If $\mathcal{G} = \mathcal{G}^{\mathcal{T}_\mathcal{G}}$ we are done, so suppose instead that $\mathcal{G} \subsetneq \mathcal{G}^{\mathcal{T}_\mathcal{G}}$. Let $s \notin \mathcal{G}_\sigma$ for $\sigma \in \mathcal{T}_\mathcal{G}$. We must show that $\overline{\mathcal{G} \cup \{s\}} = \mathcal{G}^{\mathcal{T}_\mathcal{G}}$.

$\overline{\mathcal{G} \cup \{s\}} \subseteq \mathcal{G}^{\mathcal{T}_\mathcal{G}}$ is obvious. Suppose then that $t : \tau \in \mathcal{G}^{\mathcal{T}_\mathcal{G}}$. By definition of $\mathcal{T}_\mathcal{G}$, there must be some $f \in \mathcal{G}_{\sigma\to\tau}$. Define $f^* : \sigma \to \tau$ as follows:

$$f^*(x) := \begin{cases} t & x = s \\ f(x) & \text{otherwise.} \end{cases}$$

Since $s \notin \mathcal{G}_\sigma$, we have that $f|_{\mathcal{G}_\sigma} = f^*|_{\mathcal{G}_\sigma}$. By extensionality of $\mathcal{G}$, we then have that $f^* \in \mathcal{G}$. Thus, $t = f^* s \in \overline{\mathcal{G} \cup \{s\}}$. □

2.3.2 The Single-Typed Case

Of particular interest to us will be the case where there is only one atomic type which we will denoted by 0 ($\mathcal{A} = \{0\}$). In this case, the underlying logic becomes essentially boolean:

Claim 2.3.8. *Let $\sigma \in \mathcal{T}$. Then σ is logically equivalent to either 0 or $0 \to 0$.*

Corollary 2.3.9. *There are only two logical combinatory clones in the single-typed case, $\mathcal{G}(\mathbf{X})$ and $\mathcal{G}^{taut}(\mathbf{X})$.*

The following two claims are useful:

Claim 2.3.10. *Let $\sigma = \sigma_1 \to \ldots \to \sigma_n \to 0 \in \mathcal{T}$ be a tautology. Then there is some i such that σ_i is a non-tautology.*

Claim 2.3.11. *Let $\sigma = \sigma_1 \to \ldots \to \sigma_n \to 0 \in \mathcal{T}$ be a non-tautology. Then for all i, σ_i is a tautology.*

Proof. See Proposition 2.4.4 of [1]. □

Bases for tautological clones

Suppose that X is our sole base set. Given some $b \in X$, b has more defining power than $\boldsymbol{\lambda} x^X.b$ simply by virtue of its type: $0 \to 0$ is a tautology, while 0 isn't, and so b cannot be recovered from $\boldsymbol{\lambda} x.b$ without using some other element of non-tautological type. In a sense, however, that is the only difference between the two: b and $\boldsymbol{\lambda} x.b$ will define the same elements which are tautologies. We make this claim precise:

Lemma 2.3.12. *Let $\mathcal{B}$ be a basis for $\mathcal{G}$. Then $\{\boldsymbol{\lambda} x^X.b \mid b \in \mathcal{B}\}$ is a basis for $\mathcal{G} \cap \mathcal{G}^{taut}(X)$.*

Proof. Let $t : \tau$ be any element in $\mathcal{G} \cap \mathcal{G}^{\text{taut}}(X)$, with $\tau = \tau_1 \to \ldots \to \tau_n \to 0$. Since τ is a tautology, by 2.3.10 there is some τ_i which is not a tautology. Thus, there is some closed term $T : \tau_i \to 0$.

Since $t \in \mathcal{G}$, there are $b_1, \ldots, b_m \in \mathcal{B}$ and a closed term M such that $t = Mb_1 \ldots b_n$. Fully eta-expanding yields

$$t = \lambda x_1^{\tau_1} \ldots x_n^{\tau_n}.Mb_1 \ldots b_m x_1 \ldots x_n.$$

This gives us access to the bound variable x_i, allowing us to form the term $Tx_i : 0$. We can then modify the above equation as follows:

$$t = \lambda x_1^{\tau_1} \ldots x_n^{\tau_n}.M((\boldsymbol{\lambda} x.b_1)(Tx_i)) \ldots ((\boldsymbol{\lambda} x.b_m)(Tx_i))x_1 \ldots x_n.$$

Thus, t is defined in terms of $\boldsymbol{\lambda} x.b_1, \ldots, \boldsymbol{\lambda} x.b_m$.

□

Pairing

An old result due to Grzegorczyk[4] says that the simply typed lambda calculus with with a single atomic type has product-like types with terms which mimic the behavior of pairing and projection functions. A proof may be found in Proposition 1.4.21 of [1]. As Grzegorczyk was studying functionals over the natural numbers, he naturally allowed his terms to include a constant **0** of type 0. This is unsatisfactory for our purposes, and so we present a modest improvement which avoids using a free variable of type 0:

Claim 2.3.13. *Let $\sigma, \tau \in \mathcal{T}$. Then there is a type $\sigma \underline{\times} \tau \in T$ and closed terms $\mathsf{P} : \sigma \to \tau \to \sigma \underline{\times} \tau$, $\mathsf{P}_1 : \sigma \underline{\times} \tau \to \sigma$, and $\mathsf{P}_2 : \sigma \underline{\times} \tau \to \tau$ such that any two terms $s : \sigma$, $t : \tau$,*

$$\mathsf{P}_1(\mathsf{P}st) \rightsquigarrow_{\beta\eta} s$$
$$\mathsf{P}_2(\mathsf{P}st) \rightsquigarrow_{\beta\eta} t$$

Proof. Let $\sigma = \sigma_1 \to \ldots \to \sigma_n \to 0$ and $\tau = \tau_1 \to \ldots \to \tau_m \to 0$ be given. There are three cases to consider: i) σ, τ are both non-tautologies; ii) σ, τ are both tautologies; and iii) one is a tautology and the other is a non-tautology.

Case i: σ and τ are both non-tautologies.

Let $\sigma \underline{\times} \tau := (0 \to 0 \to 0) \to \sigma_1 \to \ldots \to \sigma_n \to \tau_1 \to \ldots \to \tau_m \to 0$.

By 2.3.11, $\sigma_1, \ldots, \sigma_n, \tau_1, \ldots, \tau_m$ are all tautologies. Therefore, there are closed terms $s_1, \ldots, s_n, t_1 \ldots t_m$ of each such type. We then let

$$\begin{aligned}
\mathsf{P} &:= \lambda f^{\sigma} g^{\tau} h^{0\to 0\to 0} x_1^{\sigma_1} \ldots x_n^{\sigma_n} y_1^{\tau_1} \ldots y_m^{\tau_m}.h(fx_1 \ldots x_n)(gy_1 \ldots y_m) \\
\mathsf{P}_1 &:= \lambda P^{\sigma \underline{\times} \tau} \lambda x_1^{\sigma_1} \ldots x_n^{\sigma_n}.P(\lambda x^0 y^0.x)x_1 \ldots x_n t_1 \ldots t_m \\
\mathsf{P}_2 &:= \lambda P^{\sigma \underline{\times} \tau} \lambda y_1^{\tau_1} \ldots y_m^{\tau_m}.P(\lambda x^0 y^0.y)s_1 \ldots s_n y_1 \ldots y_m.
\end{aligned}$$

We then have that for any $s : \sigma, t : \tau$,

$$\begin{aligned}
\mathsf{P}_1(\mathsf{P}st) &= \Big(\lambda P\bar{x}.P(\lambda xy.x)\bar{x}\bar{t}\Big)\Big((\lambda fgh\bar{x}\bar{y}.h(f\bar{x})(g\bar{y}))st\Big) \\
&\rightsquigarrow_{\beta} \Big(\lambda P\bar{x}.P(\lambda xy.x)\bar{x}\bar{t}\Big)\Big(\lambda h\bar{x}\bar{y}.h(s\bar{x})(t\bar{y})\Big) \\
&\rightsquigarrow_{\beta} \lambda\bar{x}.\Big(\lambda h\bar{x}\bar{y}.h(s\bar{x})(t\bar{y})\Big)(\lambda xy.x)\bar{x}\bar{t} \\
&\rightsquigarrow_{\beta} \lambda\bar{x}.(\lambda xy.x)(s\bar{x})(t\bar{y}) \\
&\rightsquigarrow_{\beta} s\bar{x} \\
&\rightsquigarrow_{\eta} s
\end{aligned}$$

and similarly in the case of P_2.

Case ii: σ and τ are both tautologies.

By 2.3.10, there are σ_i, τ_j which are non-tautologies. Thus, $\tau_j \to \sigma_1, \ldots, \tau_j \to \sigma_n, \sigma_i \to \tau_1, \ldots, \sigma_i \to \tau_m$ are all tautologies, and so there are corresponding closed terms $s_1, \ldots, s_n, t_1, \ldots, t_m$ of each of these types. We define P as before, but the definitions for $\mathsf{P}_1, \mathsf{P}_2$ need slight modifications to ensure that types match:

$$\mathsf{P}_1 := \lambda P^{\sigma \underline{\times} \tau} \lambda x_1^{\sigma_1} \dots x_n^{\sigma_n}.P(\lambda x^0 y^0.x)x_1 \dots x_n(t_1 x_i) \dots (t_m x_i)$$
$$\mathsf{P}_2 := \lambda P^{\sigma \underline{\times} \tau} \lambda y_1^{\tau_1} \dots y_m^{\tau_m}.P(\lambda x^0 y^0.y)(s_1 y_j) \dots (s_n y_j) y_1 \dots y_m.$$

Note that the x_i and y_j variables are bound in each case, so these terms are indeed closed. The proofs for correctness work much like before.

Case iii: σ is a tautology and τ is a non-tautology.

We see here that we must modify $\sigma \underline{\times} \tau$, since we will need it to be a non-tautology, and yet it would be a tautology if we were to use the previous definition. The issue is that at least one of the σ_i's is a non-tautology. The simplest way to turn all of these into tautologies is to simply precede them with a 0; thus, our product in this case will be

$$\sigma \underline{\times} \tau := (0 \to 0 \to 0) \to (0 \to \sigma_1) \to \dots \to (0 \to \sigma_n) \to \tau_1 \to \dots \to \tau_m \to 0.$$

Since $0 \to \sigma_1, \dots, 0 \to \sigma_n, \tau_1, \dots, \tau_m$ are all tautologies, fix closed terms $s_1, \dots, s_n$ and $t_1, \dots, t_m$ of corresponding type. We then let

$$\mathsf{P} := \lambda f^{\sigma} g^{\tau} h^{0 \to 0 \to 0} x_1^{0 \to \sigma_1} \dots x_n^{0 \to \sigma_n} y_1^{\tau_1} \dots y_m^{\tau_m}.h(f(x_1(g\bar{y}) \dots (x_n(g\bar{y}))(gy_1 \dots y_m)$$
$$\mathsf{P}_1 := \lambda P^{\sigma \underline{\times} \tau} \lambda x_1^{\sigma_1} \dots x_n^{\sigma_n}.P(\lambda x^0 y^0.x)(\lambda z^0.x_1) \dots (\lambda z^0.x_n)t_1 \dots t_m$$
$$\mathsf{P}_2 := \lambda P^{\sigma \underline{\times} \tau} \lambda y_1^{\tau_1} \dots y_m^{\tau_m}.P(\lambda x^0 y^0.y)s_1 \dots s_n y_1 \dots y_m.$$

It may seem strange that $g(\bar{y})$ is present in the arguments of f, but we only need it as something of type 0 that can fill in the holes left by the newly added abstractions before each σ_i. The proof of P_2's correctness works much as before, but let us show why P_1 works:

$$\begin{aligned}
&\mathsf{P}_1(\mathsf{P}st) \\
&= \Big(\lambda P\bar{x}.P(\lambda xy.x)(\lambda z.x_1) \dots (\lambda z.x_n)\bar{t}\Big)\Big((\lambda fgh\bar{x}\bar{y}.h(f(x_1(g\bar{y}) \dots (x_n(g\bar{y}))(g\bar{y}))st\Big) \\
&\rightsquigarrow_\beta \lambda\bar{x}.\Big((\lambda h\bar{x}\bar{y}.h(s(x_1(t\bar{y})) \dots (x_n(t\bar{y}))(t\bar{y}))\Big)(\lambda xy.x)(\lambda z.x_1) \dots (\lambda z.x_n)\bar{t} \\
&\rightsquigarrow_\beta \lambda\bar{x}.(\lambda xy.x)(s((\lambda z.x_1)(t\bar{t})) \dots ((\lambda z.x_n)(t\bar{t}))(t\bar{t}) \\
&\rightsquigarrow_\beta \lambda\bar{x}.sx_1 \dots x_n \\
&\rightsquigarrow_\eta s.
\end{aligned}$$

□

Corollary 2.3.14. *Let $\mathcal{B}$ be a finite basis for a combinatory clone $\mathcal{G}$. Then $\mathcal{G}$ is generated by a single element.*

Proof. All the elements of $\mathcal{B}$ can be paired together into one by the previous result. □

Nullary Functions in Clones

Propositional logic sheds a bit of light on the nature of the nullary functions in standard clones. Logically speaking, the type $X^k \to X$ is a tautology when $k > 0$ and is just X when $k = 0$. It is then easy to justify their exclusion, since they may in some sense be viewed as an exceptional case or an annoying technicality. In the higher-order case, however, we have an infinitude of types which are non-tautologies, and so it becomes far more difficult to justify their exclusion. For this reason, and for the fact that the general problem is more easily stated with non-tautological types included, we choose not to exclude them in either the standard or combinatory clone cases.

2.3.3 The Multi-Typed Case

In the single-typed case, we saw that there are only two distinct types modulo logical equivalence, and as a consequence, there are only two deductively closed theories and two logical combinatory clones. Of course, this does not hold if we assume more than one atomic type. However, there is something to say if the set of atomic types is finite.

It is a classic result due to Diego[2] that intuitionistic implicational logic with finitely many atoms generates only finitely many sentences:

Claim 2.3.15. *Let $\mathcal{A}$ be finite, and let $\mathcal{L}$ be the set of sentences generated by $\mathcal{A}$ and $\to$. Then there are only finitely many sentences of $\mathcal{L}$, modulo logical equivalence.*

Proof. See [6]. □

Corollary 2.3.16. *Let $\mathcal{A}$ be finite. Then for any $\mathbf{X}$, there are only finitely many logical combinatory clones over $\mathbf{X}$.*

Pairing

It is not difficult to see that we cannot obtain pairing in the case where there are two or more atomic types. Logically, $\sigma \underline{\times} \tau$ is the greatest lower bound on σ and τ. However, for $\alpha \neq \beta$ atomic, α and β don't even share a common lower bound, much less a greatest lower bound.

Chapter 3

The General Case with Finite Sets

We now restrict our attention to the case when $\mathbf{X}$ is a family of finite sets. Our main result will be that the elements of $\mathbf{Cl}(\mathbf{X})$ are enough to generate all elements of $\mathbf{CCl}(\mathbf{X})$:

Theorem 3.0.1. *Suppose* $\mathbf{X}$ *is a family of finite sets. Then* $\mathcal{G}(\mathbf{X})$ *is generated by* $\mathcal{C}(\mathbf{X})$*.*

For simplicity's sake, we consider separately the cases where $\mathbf{X}$ does not or does contain a set with two elements.

3.1 Zero or one elements

Lemma 3.1.1. *Suppose* $\mathbf{X}$ *is a family of sets which are either empty or singletons. Then the flat elements* $\mathcal{C}(\mathbf{X})$ *generate all of* $\mathcal{G}(\mathbf{X})$*.*

Proof. It is easy to see that for every $\sigma \in \mathcal{T}$, $\mathbf{X}_\sigma$ is either empty or a singleton. In case $\mathbf{X}_\sigma$ is a singleton, we shall denote its unique element by u_σ. If $\mathbf{X}_\sigma$ is empty, for arbitrary τ we denote the empty map from $\mathbf{X}_\sigma \to \mathbf{X}_\tau$ by $e_{\sigma,\tau}$.

If every set in $\mathbf{X}$ is a singleton, the proof is simple: For arbitrary $\sigma = \sigma_1 \to \ldots \to \sigma_n \to \alpha$ (α atomic), we have that

$$u_\sigma = \lambda x_1^{\sigma_1} \ldots x_n^{\sigma_n}.u_\alpha$$

and since u_α is a flat term, u_σ is then definable from $\mathcal{C}(\mathbf{X})$.

Thus, we may assume that at least one set in $\mathbf{X}$, which we will call X_α, is empty.

Our proof will go more smoothly if we instead prove the following, which of course entails our previous claim:

For any $\sigma \in \mathcal{T}$, if $\mathbf{X}_\sigma$ is non-empty, then u_σ is definable from $\mathcal{C}(\mathbf{X})$, and if σ is empty, then the empty maps $e_{\alpha,\sigma}, e_{\sigma,\alpha}$ are both definable from $\mathcal{C}(\mathbf{X})$.

We now proceed naturally by induction on σ. If σ is atomic, then u_σ, $e_{\alpha,\sigma}$, and $e_{\sigma,\alpha}$ are all of flat type, and thus belong to $\mathcal{C}(\mathbf{X})$.

Suppose that $\sigma = \tau_1 \to \tau_2$. If $\mathbf{X}_{\tau_2}$ is a singleton, then

$$u_{\tau_1 \to \tau_2} = \lambda x^{\tau_1}.u_{\tau_2}$$

which is definable from $\mathcal{C}(\mathbf{X})$ since u_{τ_2} is definable from $\mathcal{C}(\mathbf{X})$ by inductive hypothesis.

Suppose then that $\mathbf{X}_{\tau_2}$ is empty. If $\mathbf{X}_{\tau_1}$ is also empty, then

$$u_{\tau_1 \to \tau_2} = \lambda x^{\tau_1}.e_{\alpha,\tau_2}(e_{\tau_1,\alpha}x)$$

where $e_{\alpha,\tau_2}, e_{\tau_1,\alpha}$ are definable from $\mathcal{C}(\mathbf{X})$ by inductive hypothesis.

Finally, we must consider the case where $\mathbf{X}_{\tau_2}$ is empty and $\mathbf{X}_{\tau_1}$ is non-empty. Since $X_{\tau_1 \to \tau_2}$ is then empty, we must give definitions for $e_{\alpha,\tau_1\to\tau_2}$ and $e_{\tau_1\to\tau_2,\alpha}$. They are given by

$$\begin{aligned} e_{\alpha,\tau_1\to\tau_2} &= \lambda x^\alpha y^{\tau_1}.e_{\alpha,\tau_2}x \\ e_{\tau_1\to\tau_2,\alpha} &= \lambda f^{\tau_1\to\tau_2}.e_{\tau_2,\alpha}(fu_{\tau_1}) \end{aligned}$$

where $e_{\alpha,\tau_2}, e_{\tau_2,\alpha}, u_{\tau_1}$ are definable from $\mathcal{C}(\mathbf{X})$ by inductive hypothesis.

□

3.2 Two or more elements

We now tackle the case where at least one set in $\mathbf{X}$ has two distinct elements. The essential ideas and constructions behind our proof were already noted by Zaionc in [7], where he establishes the definability of all Church-encoded[1] boolean functionals in the simply typed lambda calculus.

[1] That is, booleans are given the type $0 \to 0 \to 0$, with $\bot := \lambda x^0 y^0.x$ and $\top := \lambda x^0 y^0.y$. Zaionc's proof uses only the definability of the two constants along with a functionally complete set of connectives (for instance, $\wedge$ and $\neg$ given by the terms $\lambda p^{0\to0\to0} q^{0\to0\to0} x^0 y^0.pxqxy$) and $\lambda p^{0\to0\to0} x^0 y^0.pyx$, respectively). Therefore, his result can be viewed as equivalent to ours within our framework.

Suppose X_α has cardinality at least two, with $0 \neq 1 \in X_\alpha$. Let $\sqcap : \alpha \to \alpha \to \alpha$ denote the function defined by

$$\sqcap a_1 a_2 := \begin{cases} 1 & a_1 = a_2 = 1 \\ 0 & \text{otherwise.} \end{cases}$$

Since $\sqcap$ is flat, we are free to use it. For any $\sigma \in \mathcal{T}$, let $\mathbf{Eq}_\sigma : \sigma \to \sigma \to \alpha$ denote the function defined by

$$\mathbf{Eq}_\sigma s_1 s_2 := \begin{cases} 1 & s_1 = s_2 \\ 0 & \text{otherwise.} \end{cases}$$

Claim 3.2.1. *For any $\beta \in \mathcal{A}$, $\mathbf{Eq}_\beta \in \mathcal{C}(\mathbf{X})$.*

Claim 3.2.2. *For any $\sigma, \tau \in \mathcal{T}$, $\mathbf{Eq}_{\sigma\to\tau}$ is definable from $\mathcal{C}(\mathbf{X})$, $\mathbf{Eq}_\tau$ and all of the elements of $\mathbf{X}_\sigma$.*

Proof. Since $\mathbf{X}_\sigma$ is finite, extensional equality between two functions can be written out in full. Suppose $\mathbf{X}_\sigma = \{s_1, \ldots, s_N\}$. Then

$$\mathbf{Eq}_{\sigma\to\tau} := \lambda f^{\sigma\to\tau} g^{\sigma\to\tau}. \bigsqcap_{i=1}^{N} (\mathbf{Eq}_\tau (f s_i)(g s_i)).$$

□

For any $\sigma \in \mathcal{T}$, let $\mathbf{If}_\sigma : \alpha \to \sigma \to \sigma \to \sigma$ denote the function defined by

$$\mathbf{If}_\sigma a s_1 s_2 := \begin{cases} s_1 & a = 1 \\ s_2 & \text{otherwise.} \end{cases}$$

Claim 3.2.3. *For any $\sigma \in \mathcal{T}$, $\mathbf{If}_\sigma$ is definable from $\mathcal{C}(\mathbf{X})$.*

Proof. Induction on σ. If σ is atomic, then $\mathbf{If}_\sigma$ is flat. Suppose then that $\sigma = \tau_1 \to \tau_2$. We then have that

$$\mathbf{If}_{\sigma\to\tau} = \lambda a^\alpha f^{\tau_1\to\tau_2} g^{\tau_1\to\tau_2} x^{\tau_1}. \mathbf{If}_{\tau_2} a (f x)(g x)$$

where the definability of $\mathbf{If}_{\tau_2}$ follows from inductive hypothesis. □

We are now ready to show how elements of $\mathcal{G}(\mathbf{X})$ can be defined in terms of these operations.

Claim 3.2.4. *For any $\beta \in \mathcal{A}$, $X_\beta \subseteq \mathcal{C}(\mathbf{X})$.*

Claim 3.2.5. *For any $\sigma, \tau \in \mathcal{T}$, the elements of $\mathbf{X}_{\sigma\to}$ are definable from $\mathcal{C}(\mathbf{X})$, $\mathbf{Eq}_\sigma$, and all of the elements of $\mathbf{X}_\sigma$ and $\mathbf{X}_\tau$.*

Proof. Let $f : \sigma \to \tau$. Since $\mathbf{X}_\sigma$ is finite, we may represent f by a table:

x	fx
s_1	t_1
$\vdots$	$\vdots$
s_N	t_N

We then give the following definition of f:

$$\begin{aligned} f := \lambda x^{\sigma}.&\mathbf{If}_{\tau}(\mathbf{Eq}_{\sigma} x s_1) t_1 \\ &\mathbf{If}_{\tau}(\mathbf{Eq}_{\sigma} x s_2) t_2 \\ &\vdots \\ &\mathbf{If}_{\tau}(\mathbf{Eq}_{\sigma} x s_{N-1}) t_{N-1} \ \ t_N. \end{aligned}$$

The idea behind this construction is on input x to run through the list $s_1, \ldots, s_N$, and after finding some s_i which is equal to x, output the corresponding t_i. □

Lemma 3.2.6. *For any $\sigma \in \mathcal{T}$, all elements of $\mathbf{X}_\sigma$ are definable from $\mathcal{C}(\mathbf{X})$.*

Proof. As before, it helps to instead prove a modified statement:

For any $\sigma \in \mathcal{T}$, $\mathbf{Eq}_\sigma$ and all elements of $\mathbf{X}_\sigma$ are definable from $\mathcal{C}(\mathbf{X})$.

This naturally follows by induction on σ using 3.2.1, 3.2.2, 3.2.4, and 3.2.5. □

3.3 Infinite Sets

It is not difficult to see that issues of cardinality prevent us from establishing a similar result in the case of infinite sets. Consider the single-typed case with set ω. A basic cardinality calculation shows that $|\mathcal{C}(\omega)| = 2^{\aleph_0}$. The number of lambda terms using metavariables among $\mathcal{C}(\omega)$ is then also size $2^{\aleph_0}$. However, ω^{ω^ω} already has cardinality $2^{2^{\aleph_0}}$[2].

[2] In fact, $|\mathcal{G}(\omega)| = \beth_\omega$, considerably larger than $|\mathcal{C}(\omega)| = \beth_1$.

Chapter 4

The Boolean Case

We now focus our attentions on $\mathbf{CCl}(\mathbf{B})$, the combinatory clones over the single set $\mathbf{B}$, which is essentially the simplest non-trivial case to consider. Throughout this chapter, we take $\mathcal{A} = \{0\}$ with 0 corresponding to $\mathbf{B}$. In particular, we work toward a classification of the coatomic combinatory clones in hopes of a classification of the bases of $\mathbf{CCl}(\mathbf{B})$ similar to Post's classification of the functionally complete sets of connectives.

4.1 Basic Results

4.1.1 Non-injectivity of Φ

In 2.2.7, we saw that the clones over a family of sets $\mathbf{X}$ inject into the combinatory clones over $\mathbf{X}$ via the map Ψ. We now provide a simple counterexample which shows that Φ is not injective.

Throughout this section, let $\mathbf{F} : (0 \to 0) \to 0 \to 0$ denote the following function:

f	$\mathbf{F}(f)$
$\boldsymbol{\lambda} b.\bot$	$\boldsymbol{\lambda} b.\bot$
$\boldsymbol{\lambda} b.b$	$\boldsymbol{\lambda} b.b$
$\neg$	$\boldsymbol{\lambda} b.\bot$
$\boldsymbol{\lambda} b.\top$	$\boldsymbol{\lambda} b.\top$

Lemma 4.1.1. $\mathbf{F}$ *is not lambda definable.*

Proof. Let $T : (0 \to 0) \to 0 \to 0$ be a closed lambda term. By 1.3.6, suppose it is in long normal form. It is well known that the long normal forms of type $(0 \to 0) \to 0 \to 0$ are $\lambda f^{0\to 0} x^0 . f^k x$ for $k \in \omega$, the so-called Church numerals.

It is not hard to see that the Church numerals cannot map $\neg$ to $\boldsymbol{\lambda} x.\bot$:

$$[\![\lambda f x.f^k x]\!](\neg) = \begin{cases} \boldsymbol{\lambda} x.x & k \text{ even} \\ \neg & k \text{ odd.} \end{cases}$$

Therefore, no such T can represent $\mathbf{F}$.

□

Lemma 4.1.2. *Let $T : 0$ be a term whose free variables are among $F : (0 \to 0) \to 0 \to 0$ and $x_1, \ldots, x_n : 0$. Then there is an x_i such that for all assignments g we have that*

$$[\![T]\!]_{g[F \mapsto \mathbf{F}]} = g(x_i).$$

Proof. Assume T is in long normal form. We now induct on the structure of T. If T is just a variable, then it must be some x_i, and so the claim follows immediately. The other option is that T begins with F, in which case we have that $T = F(\lambda y.U)V$, for $U, V : 0$ in long normal form. By inductive hypothesis, U and V each correspond to some free variables of type 0. Let x_j be the free variable for V. For U, there are two cases to consider: First, the variable could be the y from the neighboring lambda abstraction. In that case we have that

$$\begin{aligned} [\![T]\!]_{g[F \mapsto \mathbf{F}]} &= [\![F(\lambda y.U)V]\!]_{g[F \mapsto \mathbf{F}]} \\ &= \mathbf{F}([\![\lambda y.U]\!]_{g[F \mapsto \mathbf{F}]})([\![V]\!]_{g[F \mapsto \mathbf{F}]}) \\ &= \mathbf{F}(\boldsymbol{\lambda} b.[\![U]\!]_{g[F \mapsto \mathbf{F}, y \mapsto b]})(g(x_j)) \\ &= \mathbf{F}(\boldsymbol{\lambda} b.b)(g(x_j)) \\ &= (\boldsymbol{\lambda} b.b)(g(x_j) \\ &= g(x_j). \end{aligned}$$

U might also correspond to one of the x_i's. In that case,

$$\begin{aligned} [\![T]\!]_{g[F \mapsto \mathbf{F}]} &= [\![F(\lambda y.U)V]\!]_{g[F \mapsto \mathbf{F}]} \\ &= \mathbf{F}([\![\lambda y.U]\!]_{g[F \mapsto \mathbf{F}]})([\![V]\!]_{g[F \mapsto \mathbf{F}]}) \\ &= \mathbf{F}(\boldsymbol{\lambda} b.[\![U]\!]_{g[F \mapsto \mathbf{F}, y \mapsto b]})(g(x_j)) \\ &= \mathbf{F}(\boldsymbol{\lambda} b.g(x_i))(g(x_j)) \\ &= (\boldsymbol{\lambda} b.g(x_i))(g(x_j)) \\ &= g(x_i). \end{aligned}$$

□

Corollary 4.1.3. *The only flat functions definable from $\mathbf{F}$ are the projections.*

Proof. Suppose G is a flat term with at most one free variable $F : (0 \to 0) \to 0 \to 0$. As always, we may assume G is in long normal form:

$$G = \lambda x_1^0 \dots x_n^0.T$$

where T is of type 0 with free variables among F and $x_1, \dots, x_n$. Applying the previous lemma to T, we obtain some variable x_i which corresponds to it. Thus,

$$\begin{aligned}[\![G]\!]_{[F\mapsto\mathbf{F}]} &= [\![\lambda x_1 \dots x_n.T]\!]_{[F\mapsto\mathbf{F}]} \\ &= \boldsymbol{\lambda} b_1 \dots b_n.[\![T]\!]_{[F\mapsto\mathbf{F},x_1\mapsto b_1,\dots,x_n\mapsto b_n]} \\ &= \boldsymbol{\lambda} b_1 \dots b_n.b_i\end{aligned}$$

which is of course a projection.

□

We can summarize these results as follows:

Corollary 4.1.4. $\Lambda(\mathbf{B}) \neq \overline{\{\mathbf{F}\}}$, *but* $\Phi(\Lambda(\mathbf{B})) = \Phi(\overline{\{\mathbf{F}\}}) = \Pi(\mathbf{B})$.

4.1.2 Bases in CCl(B)

We briefly restate some of our previous results in terms of combinatory clones over $\mathbf{B}$.

Claim 4.1.5. *Let $\mathcal{B}$ be a basis for $\mathcal{C}(\mathbf{B})$. Then $\mathcal{B}$ is a basis for $\mathcal{G}(\mathbf{B})$.*

Proof. This follows from 3.0.1. □

Claim 4.1.6. $\mathcal{G}(\mathbf{B})$ *is generated by a single element.*

Proof. Take a finite basis for $\mathcal{C}(\mathbf{B})$, e.g. $\{\bot, \to\}$. By the previous lemma, it is also a basis for $\mathcal{G}(\mathbf{B})$. The claim then follows from 2.3.14. □

Claim 4.1.7. *Let $\mathcal{B}$ be a basis for $\mathcal{C}(\mathbf{B})$ save for the two boolean constants (that is, $\mathcal{B}$ generates all functions of arity 1 or higher; for example, $\{\neg, \wedge\}$). Then $\mathcal{B}$ is a basis for $\mathcal{G}^{taut}(\mathbf{B})$.*

Proof. By 4.1.5 and 2.3.12, $\{\boldsymbol{\lambda} x.\bot, \boldsymbol{\lambda} x.\ \to\}$ are a basis for $\mathcal{G}^{\mathrm{taut}}(\mathbf{B})$. Since $\mathcal{B}$ generates all flat non-constants, both are definable from $\mathcal{B}$. □

Let us conclude by showing that $\mathcal{G}^{\mathrm{taut}}(\mathbf{B})$ is a coatom:

Claim 4.1.8. $\mathcal{G}^{taut}(\mathbf{B})$ *is a coatom.*

Proof. Let σ be a non-tautology and let $s \in \mathcal{G}(\mathbf{B})_\sigma$ be arbitrary. Let $t \in \mathcal{G}(\mathbf{B})_\tau$ be any element. Clearly, $\sigma \to \tau$ must be a tautology, so $\boldsymbol{\lambda} x^\sigma.t \in \mathcal{G}^{\mathrm{taut}}(\mathbf{B})$. Thus, $t = (\boldsymbol{\lambda} x.t)s \in \overline{\mathcal{G}^{\mathrm{taut}}(\mathbf{B}) \cup \{s\}}$. This hold for arbitrary t, so $\overline{\mathcal{G}^{\mathrm{taut}}(\mathbf{B}) \cup \{s\}} = \mathcal{G}(\mathbf{B})$. □

4.1.3 A Lindenbaum lemma for CCl(B)

Lemma 4.1.9. *Let $\mathcal{G} \subsetneq \mathcal{H} \in \mathbf{CCl}(\mathbf{B})$ with $\mathcal{H}$ generated by a single element. Then there is a combinatory clone $\mathcal{G}'$ covered by $\mathcal{H}$ such that $\mathcal{G} \subseteq \mathcal{G}'$.*

Proof. We employ a standard Lindenbaum-like argument: Let $\mathcal{H}$ be generated by t and let $\{t_i\}_{i\in\omega}$ be an ordering of the elements of $\mathcal{H}$. Define $\mathcal{G}_n$ inductively:

$$\mathcal{G}_0 := \mathcal{G}$$
$$\mathcal{G}_{n+1} := \begin{cases} \mathcal{G}_n & t \in \overline{\mathcal{G}_n \cup \{t_n\}} \\ \mathcal{G}_n \cup \{t_n\} & \text{otherwise.} \end{cases}$$

Let $\mathcal{G}' := \bigcup_{i\in\omega} \mathcal{G}_i$. Obviously, $\mathcal{G} \subseteq \mathcal{G}' \subseteq \mathcal{H}$. Furthermore, $\mathcal{G}' \subsetneq \mathcal{H}$, since $t \notin \mathcal{G}'$: Suppose $t = t_N$. Since $t \notin \mathcal{G}$, if t were in $\mathcal{G}'$ then it would've been added at stage $N+1$ of this process, which clearly isn't possible. The real work then is showing that $\mathcal{G}'$ is a combinatory clone. Of course, $\mathcal{G}$ contains all **K**'s and **S**'s, since those were already present in $\mathcal{G}$ at stage 0, so it remains to be seen that $\mathcal{G}'$ is closed under application. Let $t_i, t_j \in \mathcal{G}'$ and suppose that $t_k = t_i t_j \notin \mathcal{G}_i$. The only way this can be is that t_k wasn't added at stage $k+1$, meaning that t is definable from t_k, the elements of $\mathcal{G}$, and some $t_{k_1}, \ldots, t_{k_n} \in \mathcal{G}_k$. Consider $M := \max\{i, j, k_1, \ldots, k_n\}$. At stage $M+1$, adding t_M would yield all the necessary ingredients to define t, since $\mathcal{G}_{M+1}$ would then have all elements of $\mathcal{G}$, each such t_{k_l}, and t_i, t_j which are sufficient to define t_k. Therefore, $t_M \notin \mathcal{G}_M$, which contradicts our assumption that t_M was added.

Finally, $\mathcal{G}'$ is covered by $\mathcal{H}$: Let $t_k \in \mathcal{H} \setminus \mathcal{G}'$. We have that t_k was not added at stage $k+1$, so t is definable from t_k and $\mathcal{G}_k \subseteq \mathcal{G}'$. Since t generated all of $\mathcal{H}$, we then have that $\overline{\mathcal{G}' \cup \{t_k\}} = \mathcal{H}$.

□

Corollary 4.1.10. *Let $\mathcal{C}$ be coatomic in $\mathbf{Cl}(\mathbf{B})$. Then there is some $\mathcal{G}$ coatomic in $\mathbf{CCl}(\mathbf{B})$ such that $\Phi(\mathcal{G}) = \mathcal{C}$.*

Proof. By 2.2.7, $\Phi(\Psi(\mathcal{C}) = \mathcal{C}$. By 4.1.9, we may find some $\mathcal{G}$ such that $\Psi(\mathcal{C}) \subseteq \mathcal{G}$ and $\mathcal{G}$ is covered by $\mathcal{G}(\mathbf{B})$. Clearly, we have that

$$\mathcal{C} = \Phi(\Psi(\mathcal{C})) \subseteq \Phi(\mathcal{G})$$

and since $\mathcal{C}$ is a coatom, $\Phi(\mathcal{G})$ must either be $\mathcal{C}$ or $\mathcal{C}(\mathbf{B})$. Since $\mathcal{C}(\mathbf{B})$ generates all of $\mathcal{G}(\mathbf{B})$, we can conclude that $\Phi(\mathcal{G}) = \mathcal{C}$. □

4.2 Categorical characterizations of the coatomic boolean clones

While 4.1.9 establishes the existence of a coatomic combinatory clone for each coatomic clone in $\mathbf{Cl}(\mathbf{B})$, the proof provides us with little extra information. In particular, the following questions remain unanswered:

- Do the coatoms in $\mathbf{CCl}(\mathbf{B})$ have nice characterizations? If so, what are they?
- Can there be multiple coatoms in $\mathbf{CCl}(\mathbf{B})$ for a given coatom in $\mathbf{Cl}(\mathbf{B})$ (in the same way that a maximal consistent extension of a theory need not be unique)?

In 1.2, we saw that the coatoms of $\mathbf{CCl}(\mathbf{B})$ have nice, fairly natural characterizations. The obvious thing to do is then to see if these characterizations can be extended to higher-order type. However, it is not always clear how to "correctly" generalize a property from flat to higher-order. Take, for instance, the false-preserving functions. Every type has a bottom element of sorts: For $\sigma = \sigma_1 \to \ldots \to \sigma_n \to 0$, take $\bot_\sigma := \lambda x_1^{\sigma_1} \ldots x_n^{\sigma_n}.\bot$. We might then expect that false-preserving functions from $\mathbf{B}_\sigma \to \mathbf{B}_\tau$ be given as those f for which $f(\bot_\sigma) = \bot_\tau$. As it turns out, this is *not* the correct definition, but it is still helpful to understand why it does not work. To that end, we begin by exploring how clones can be understood in terms of structures over $\mathbf{B}$ (specifically, those that lie in a category with finite products).

Claim 4.2.1. *Let $\mathcal{C}$ be a set-based*[1] *category with finite products and X an object in $\mathcal{C}$ whose underlying set is $\mathbf{B}$. Then the collection*

$$\bigcup_{i \in \omega} \mathrm{Hom}(X^k, X)$$

reinterpreted as elements of $\mathcal{C}(\mathbf{B})$ is a clone.

Proof. Standard. □

In fact, every clone over a family of sets can be seen as arising in this manner, albeit from a possibly quite artificial category. Nonetheless, we will now observe that the five coatomic clones as given by Post have very natural categorical interpretations:

4.2.1 False- and True-Preserving Functions

Recall that a **pointed set** is just a set X paired with an element of $x_0 \in X$. A homomorphism between pointed sets $\langle X, x_0 \rangle$ and $\langle Y, y_0 \rangle$ is a function $f : X \to Y$ such that $f(x_0) = y_0$. Pointed sets have products given by $\langle X, x_0 \rangle \times \langle Y, y_0 \rangle := \langle X \times Y, \langle x_0, y_0 \rangle \rangle$.

Let $\mathbf{B}_\bot := \langle \mathbf{B}, \bot \rangle$.

Claim 4.2.2. *The false-preserving functions of arity k are precisely* $\mathrm{Hom}(\mathbf{B}_\bot^k, \mathbf{B}_\bot)$.

Proof. Notice that the distinguished point in $\mathbf{B}_\bot^k$ is just the vector $\langle \bot, \ldots, \bot \rangle$. □

Of course, the true-preserving functions have a similar characterization.

[1] To be precise, there is a forgetful functor from $\mathcal{C}$ into **Set**.

4.2.2 Monotone Functions

Given posets $\langle X, \leq \rangle$, $\langle Y, \preceq \rangle$, a homomorphism between them is a function $f : X \to Y$ which preserves order: $x \leq x' \Rightarrow f(x) \preceq f(x')$. Posets have products given by the product orderings. Let $\mathbf{B}_{\leq}$ denote $\mathbf{B}$ with the standard ordering: $\bot \leq \top$.

Claim 4.2.3. *The monotone functions of arity k are precisely* $\mathrm{Hom}(\mathbf{B}_{\leq}^k, \mathbf{B}_{\leq})$.

4.2.3 Self-dual Functions

By a **unary system** we mean a set X together with a unary operation $f_X : X \to X$. Given unary systems $\langle X, f_X \rangle$, $\langle Y, f_Y \rangle$, a homomorphism between them is a function $g : X \to Y$ such that for all $x \in X$, $g(f_X(x)) = f_Y(g(x))$. Unary systems have products given by $\langle X, f_X \rangle \times \langle Y, f_Y \rangle := \langle X \times Y, f_X \times f_Y \rangle$, with

$$(f_X \times f_Y)\langle x, y \rangle = \langle f_X(x), f_Y(y) \rangle.$$

Let $\mathbf{B}_{\neg} := \langle \mathbf{B}, \neg \rangle$.

Claim 4.2.4. *The self-dual functions of arity k are precisely* $\mathrm{Hom}(\mathbf{B}_{\neg}^k, \mathbf{B}_{\neg})$.

Proof. The unary operation corresponding to $\mathbf{B}_{\neg}^k$ is the function given by the rule

$$\langle x_1, \ldots, x_k \rangle \mapsto \langle \neg x_1, \ldots, \neg x_k \rangle$$

and so a function $f : \mathbf{B}^k \to \mathbf{B}$ is a unary system homomorphism from $\mathbf{B}_{\neg}^k$ to $\mathbf{B}_{\neg}$ if and only if for all $x_1, \ldots, x_k$

$$f(\neg x_1, \ldots, \neg x_k) = \neg f(x_1, \ldots, x_k).$$

□

4.2.4 Affine functions

Let $\langle M, *, 1_M \rangle$, $\langle N, \circ, 1_N \rangle$ be monoids. A function $f : M \to N$ is said to be **affine** if there is a monoid homomorphism $\varphi : M \to N$ and a $b \in N$ such that for all $x \in M$, $f(x) = \varphi(x) \circ b$.

Monoids with affine functions form a category with products given by the usual product on monoids.

Let $\mathbf{B}_{\oplus} := \langle \mathbf{B}, \oplus, \bot \rangle$, the monoid whose operation is given by exclusive disjunction.

Claim 4.2.5. *The affine functions of arity k are precisely* $\mathrm{Hom}(\mathbf{B}_{\oplus}^k, \mathbf{B}_{\oplus})$ *in the category of monoids with affine maps.*

Proof.
$\Rightarrow$ Suppose $f(\mathbf{x}) = \mathbf{v} \cdot \mathbf{x} \oplus b$. Observe that $\mathbf{x} \mapsto \mathbf{v} \cdot \mathbf{x}$ is a monoid homomorphism by virtue of the linearity of the dot product.

$\Leftarrow$ Suppose $f(\mathbf{x}) = \varphi(\mathbf{x}) \oplus b$ is affine and arity k. For $1 \leq i \leq k$, let $\mathbf{e}_i$ denote the vector whose entries are all $\bot$ except in the i position where it is $\top$.

Let $\mathbf{v} := \langle \varphi(\mathbf{e}_1), \ldots, \varphi(\mathbf{e}_k) \rangle$. We now claim that for all $\mathbf{x}$, $\mathbf{v} \cdot \mathbf{x} = \varphi(\mathbf{x})$. Since both are homomorphisms and since $\{\mathbf{e}_1, \ldots, \mathbf{e}_k\}$ generate all of $\mathbf{B}^k_\oplus$, it suffices to show that for each i that $\mathbf{v} \cdot \mathbf{e}_i = \varphi(\mathbf{e}_i)$. This is immediate. □

With these characterizations in hand, we look to extend them to the higher-order types. In addition to finite products, we will expect that our categories of interest have exponential objects, i.e. are Cartesian closed.

4.3 False- and True-Preserving Functions

In the previous section, we saw that false- and true-preserving (flat) functions correspond to the category of pointed sets. Unfortunately, this category is not Cartesian closed. Therefore, our idea from the start of 4.2 won't work. However, if we instead think of $\bot$ (in the false-preserving case) not as a distinguished point but as a distinguished *subset*, we are able to define exponentials:

Definition 4.3.1. A **set with subset** is a set X paired with some distinguished $S_X \subseteq X$.

A homomorphism from $\langle X, S_X \rangle$ to $\langle Y, S_Y \rangle$ is a function $f : X \to Y$ for which $f(S_X) \subseteq S_Y$.

Sets with subsets form a Cartesian closed category with exponentials given by

$$\mathbf{Y}^{\mathbf{X}} := \langle Y^X, \mathrm{Hom}(\mathbf{X}, \mathbf{Y}) \rangle.$$

With this in mind, we generate the generalized false-preserving functions from the object $\langle \mathbf{B}, \{\bot\} \rangle$. To be precise:

Definition 4.3.2. FP, the combinatory clone of false-preserving functions is defined inductively:

$$\begin{aligned} \mathrm{FP}_0 &:= \{\bot\} \\ \mathrm{FP}_{\sigma \to \tau} &:= \{f : \mathbf{B}_\sigma \to \mathbf{B}_\tau \mid f(\mathrm{FP}_\sigma) \subseteq \mathrm{FP}_\tau\}. \end{aligned}$$

The true-preservers are similarly defined:

Definition 4.3.3.

$$\begin{aligned} \mathrm{TP}_0 &:= \{\top\} \\ \mathrm{TP}_{\sigma \to \tau} &:= \{f : \mathbf{B}_\sigma \to \mathbf{B}_\tau \mid f(\mathrm{TP}_\sigma) \subseteq \mathrm{TP}_\tau\}. \end{aligned}$$

Since they are nearly identically defined, we will only prove a given result for one of either FP or TP.

Lemma 4.3.4. FP *is a combinatory clone.*

Proof. FP is closed under application by design, so it suffices to check that the **K** and **S** combinators belong to FP.

Suppose $s \in \mathrm{FP}_\sigma$. We must check that $\mathbf{K}_{\sigma,\tau} s = \boldsymbol{\lambda} y.s \in \mathrm{FP}_{\tau\to\sigma}$. Suppose then that $t \in \mathrm{FP}_\tau$. We then have that $\mathbf{K}st = s \in \mathrm{FP}_\sigma$ by assumption. Thus, $\mathbf{K}s \in \mathrm{FP}_{\sigma\to\tau}$, and so $\mathbf{K} \in \mathrm{FP}_{\sigma\to\tau\to\sigma}$.

A no less simple but much more tedious argument shows that an arbitrary **S** combinator belongs to FP as well. □

Let us be sure that our generalized version of false-preserving encompasses the old version:

Lemma 4.3.5. *Let* $\mathrm{FP}_{\mathrm{flat}}$ *denote the* FP *functions of flat type. For any* f*,* $f \in \mathrm{FP}_{\mathrm{flat}} \Leftrightarrow f\bot \ldots \bot = \bot$.

Proof. Induction on the arity of f. If f is nullary, the statement is immediate. Suppose then that f is $n+1$-ary.

$\Leftarrow$ Suppose $f \in \mathrm{FP}_{\mathrm{flat}}$. By definition, this means that $f\bot$ is an n-ary function in $\mathrm{FP}_{\mathrm{flat}}$. By inductive assumption, this means that $f\bot\bot \ldots \bot = (f\bot)\bot \ldots \bot = \bot$.

$\Rightarrow$ Suppose that $f\bot\bot \ldots \bot = (f\bot)\bot \ldots \bot = \bot$. By inductive assumption, this means that $f\bot \in \mathrm{FP}_{\mathrm{flat}}$. Thus, f sends every element in FP_0 ($\bot$) to one in FP ($f\bot$). Thus, $f \in \mathrm{FP}_{\mathrm{flat}}$. □

It is not hard to see that FP is extensional: We only require a given FP function to send FP elements to FP elements, but that function can do whatever it wants with the non-FP elements. Thus, we obtain the following result:

Lemma 4.3.6. FP *is a coatom.*

Proof. By 2.3.7, FP is either a logical combinatory clone or covered by one. Clearly, it isn't logical (since $\bot \in \mathrm{FP}$ but $\top \notin \mathrm{FP}$) and it cannot be covered by $\mathcal{G}^{\mathrm{taut}}(\mathbf{B})$ (since $\bot \in \mathrm{FP}$). Thus, FP must be covered by $\mathcal{G}(\mathbf{B})$. □

4.3.1 Completeness of $\mathrm{TP}_{\mathrm{flat}}$

Close examination of the proof of 3.0.1 shows that in the boolean case, the only connectives needed are conjunction, equivalence, the if-then-else operator, and the two boolean constants. Since each of these except for the false constant are

truth-preserving, one might wonder if the proof can be modified to show that the flat truth-preservers are enough to generate all truth-preservers. Indeed, this is exactly what we endeavor to do.

Immediately, one finds some obvious difficulties with this approach. First, to define a function of type $\sigma \to \tau$, we needed definitions for all elements of σ and τ. Since our inductive assumption will not give us the non-TP elements of these types, we will need to work around this. However, we know that the flat truth-preservers with the single addition of $\bot$ is functionally complete, and so for any element $s : \sigma$ there is a $\mathrm{TP}_{\mathrm{flat}}$-definable function $T^s : 0 \to \sigma$ such that $T^s \bot = s$.

The idea now is this: if our function f maps the non-TP element s to a (possibly non-TP) element t, we can use the fact that s is non-TP in order to give a definition of $\bot$ from it. We can then use $\bot$ and T^t in order to define t.

We shall do this using the indicator functions for TP. Given a type σ, let $\mathbf{IsTP}_\sigma : \sigma \to 0$ be the function given by

$$\mathbf{IsTP}_\sigma\, x = \begin{cases} \top & x \in \mathrm{TP}_\sigma \\ \bot & \text{otherwise.} \end{cases}$$

Claim 4.3.7. $\mathbf{IsTP}_0$ *is* $\mathrm{TP}_{\mathrm{flat}}$*-definable.*

Proof. Obviously, $\mathbf{IsTP}_0 = \lambda x^0.x$. □

Claim 4.3.8. $\mathbf{IsTP}_{\sigma\to\tau}$ *is* $\mathrm{TP}_{\mathrm{flat}}$*-definable from* $\mathbf{IsTP}_\tau$ *and the elements of* TP_σ*.*

Proof. Suppose $s_1, \ldots, s_N$ are the elements of TP_σ. We then have that $\mathbf{IsTP}_{\sigma\to\tau} = \lambda f^{\sigma\to\tau} \bigwedge_{i=1}^{N} \mathbf{IsTP}_\tau(f s_i)$. □

The second issue we encounter is that the old proof requires that we have an operation $\mathbf{Eq}_\sigma : \sigma \to \sigma \to 0$ which mimics equality. However, we quickly see that this won't be definable from truth-preserving functions alone: We might have two terms $s \neq s' \in \mathrm{TP}_\sigma$; however, if $\mathbf{Eq}_\sigma$ is truth-preserving, then $\mathbf{Eq}_\sigma\, ss' = \top$.

To get around this, we need to augment our domain of truth values with a new false constant which itself is truth-preserving. The simplest way to do this is to take our new truth value domain to be $0 \to 0$, with $\lambda x^0.\bot$,$\lambda x^0.\top$ interpreted as false and true as expected, but with $\lambda x^0.x$ additionally interpreted as false (negation remains uninterpreted).

We now need new logical operations to match this new set of truth values. Let $\mathbf{If} : 0 \to 0 \to 0 \to 0$ denote the standard if-then-else operator (that is, $\mathbf{If}\top xy = x$ and $\mathbf{If}\bot xy = y$). We modify $\mathbf{If}$ in order to accommodate our new set of truth values in the first argument:

$$\tilde{\mathbf{If}} : (0 \to 0) \to 0 \to 0 \to 0 := \lambda b^{0\to 0} x^0 y^0 . \mathbf{If}(b(x \wedge y))xy.$$

Notice that $\tilde{\mathbf{If}}$ is definable from $\mathrm{TP}_{\mathrm{flat}}$, since $\mathbf{If}, \wedge \in \mathrm{TP}_{\mathrm{flat}}$.

Claim 4.3.9. *We have the following equations:*

$$\tilde{\mathbf{If}}(\lambda x.\top)ab = a$$
$$\tilde{\mathbf{If}}(\lambda x.x)ab = b$$
$$\tilde{\mathbf{If}}(\lambda x.\bot)ab = b.$$

Proof. Routine. □

Therefore, $\tilde{\mathbf{If}}$ correctly interprets $\lambda x.\top$ as true and $\lambda x.x$,$\lambda x.\bot$ as false.

We extend this definition to arbitrary type, letting $\tilde{\mathbf{If}}_0 := \tilde{\mathbf{If}}$ and letting

$$\tilde{\mathbf{If}}_{\sigma\to\tau} := \lambda b^{0\to 0} f^{\sigma\to\tau} g^{\sigma\to\tau} x^\sigma . \tilde{\mathbf{If}}_\tau b(fx)(gx).$$

Equations similar to the above three hold for arbitrary $\tilde{\mathbf{If}}_\sigma$.

We define conjunction pointwise: $\tilde{\wedge} : (0 \to 0) \to (0 \to 0) \to 0 \to 0 := \lambda b^{0\to 0} c^{0\to 0} x^0 . (bx) \wedge (cx)$. One can verify that this correct with respect to our interpreted truth values.

By an equality operation on a type σ we shall mean any function $f : \sigma \to \sigma \to 0 \to 0$ such that for all $s : \sigma$, $fss = \lambda x.\top$, and for $s \neq s' : \sigma$, fss' equals either $\lambda x.\bot$ or $\lambda x.x$.

Claim 4.3.10. *There is a* $\mathrm{TP}_{\mathrm{flat}}$*-definable equality operator* $\mathbf{Eq}_0$ *on 0.*

Proof. Let $\mathbf{Eq}_0 := \lambda xyz.x \leftrightarrow y$. □

Claim 4.3.11. *There is a* $\mathrm{TP}_{\mathrm{flat}}$*-definable equality operator* $\mathbf{Eq}_{\sigma\to\tau}$ *on* $\sigma \to \tau$ *definable from the elements of* TP_σ *and an equality operator on* τ*.*

Proof. Let $\mathbf{Eq}_\tau$ be an equality operator on τ. Let $s_1, \ldots, s_N$ be the TP_σ elements and let $u_1, \ldots, u_M$ be the non-TP_σ elements. We define $\mathbf{Eq}_{\sigma\to\tau}$ to be the following function:

$$\lambda f^{\sigma\to\tau} g^{\sigma\to\tau} x^0 . \Big(\tilde{\bigwedge}_{i=1}^{N} \mathbf{Eq}_\tau(fs_i)(gs_i) \tilde{\wedge} \tilde{\bigwedge}_{j=1}^{M} \mathbf{Eq}_\tau(f(T^{u_j}x))(g(T^{u_j}x)) \Big) x.$$

□

We are now ready to demonstrate how we can use these operations to define elements of TP.

Claim 4.3.12. *All elements of* TP_0 *are* $\mathrm{TP}_{\mathrm{flat}}$*-definable.*

Claim 4.3.13. *All elements of* $\mathrm{TP}_{\sigma\to\tau}$ *are* $\mathrm{TP}_{\mathrm{flat}}$*-definable from* $\mathbf{IsTP}_\sigma$*, an equality operator* $\mathbf{Eq}_\sigma$*, and the elements of* TP_σ *and* TP_τ*.*

Proof. Let $s_1, \ldots, s_N$ be the elements of TP_σ and let $u_1, \ldots, u_M$ be the non-TP_σ elements. Suppose $f : \sigma \to \tau$ is given by the following table:

x	fx
s_1	t_1
$\vdots$	$\vdots$
s_N	t_N
u_1	v_1
$\vdots$	$\vdots$
u_M	v_M.

Since $f \in \mathrm{TP}$, we know that $t_1, \ldots, t_N \in \mathrm{TP}_\tau$, so we are free to use them in our definition. We now define f as follows:

$$
\begin{aligned}
f := \lambda x^\sigma.&\tilde{\mathbf{If}}_\tau(\mathbf{Eq}_\sigma x s_1)t_1\\
&\tilde{\mathbf{If}}_\tau(\mathbf{Eq}_\sigma x s_2)t_2\\
&\vdots\\
&\tilde{\mathbf{If}}_\tau(\mathbf{Eq}_\sigma x s_N)t_N\\
&\tilde{\mathbf{If}}_\tau(\mathbf{Eq}_\sigma x(T^{u_1}(\mathbf{IsTP}_\sigma\, x)))v_1\\
&\tilde{\mathbf{If}}_\tau(\mathbf{Eq}_\sigma x(T^{u_2}(\mathbf{IsTP}_\sigma\, x)))v_2\\
&\vdots\\
&\tilde{\mathbf{If}}_\tau(\mathbf{Eq}_\sigma x(T^{u_{M-1}}(\mathbf{IsTP}_\sigma\, x)))v_{M-1}v_M.
\end{aligned}
$$

The construction works much like before, with the added wrinkle that if we "reach" the lines dealing with non-TP elements, we know that the argument x of the function f is non-TP. Therefore, $\mathbf{IsTP}_\sigma\, x$ will equal $\bot$, and so the T^t functions will yield the necessary elements.

□

Lemma 4.3.14. *For every* $\sigma \in \mathcal{T}$ *we have that all elements of* TP_σ*,* $\mathbf{IsTP}_\sigma$*, and an equality operator* $\mathbf{Eq}_\sigma$ *are* $\mathrm{TP}_{\mathrm{flat}}$*-definable.*

Proof. This follows from an induction on the type σ using claims 1.4, 1.5, 1.6, 1.7, 1.8, and 1.9. □

Corollary 4.3.15. $\mathrm{TP} = \overline{\mathrm{TP}_{\mathrm{flat}}}$ *and* TP *is the unique generalized clone* $\mathcal{G}$ *such that* $\Phi(\mathcal{G}) = \mathrm{TP}_{\mathrm{flat}}$.

Proof. Suppose $\mathcal{G}$ is such that $\Phi(\mathcal{G}) = \mathrm{TP}_{\mathrm{flat}}$. By the previous lemma, $\mathrm{TP} \subseteq \mathcal{G}$. Since TP is a coatom, it follows that $\mathcal{G}$ is either TP or $\mathcal{G}(\mathbf{B})$. The latter is easily ruled out, and so $\mathcal{G} = \mathrm{TP}$. □

4.4 Monotone Functions

At first, we are encouraged by the fact that posets are Cartesian-closed: Given posets $\mathbf{X} = \langle X, \leq_X \rangle$ and $\mathbf{Y} = \langle Y, \leq_Y \rangle$, the exponential is given by

$$\mathbf{Y}^{\mathbf{X}} := \langle \mathrm{Hom}(\mathbf{X}, \mathbf{Y}), \preceq \rangle$$

with $\preceq$ defined pointwise:

$$f \preceq g :\equiv \forall x \in X, f(x) \leq_Y g(x).$$

There is one problem in basing our definition of higher-order monotone on this: The exponentials "forget" the non-monotone elements. For instance, suppose we wish to see what the monotone elements of type $(0 \to 0) \to 0$ are. Naturally, we expect them to be $\mathrm{Hom}(\mathbf{B}_{\leq}^{\mathbf{B}_{\leq}}, \mathbf{B}_{\leq})$. However, the functions in this set have as domain the three element set $\mathrm{Hom}(\mathbf{B}_{\leq}, \mathbf{B}_{\leq})$ and not the four element set $\mathbf{B}^{\mathbf{B}}$ as desired. In other words, these functions don't tell us what to do with the sole non-monotone element $(\neg)$.

We then must be very careful in defining what we mean by monotone in the higher-order case.

Definition 4.4.1. For each $\sigma \in \mathcal{T}$, define Mon_σ and $\leq_\sigma$ by mutual recursion:

$$\begin{aligned} \mathrm{Mon}_0 &:= \{\bot, \top\} \\ \leq_0 &:= \leq . \end{aligned}$$

$$\begin{aligned} \mathrm{Mon}_{\sigma\to\tau} := \{f : \mathbf{B}_\sigma \to \mathbf{B}_\tau \mid\ & f(\mathrm{Mon}_\sigma) \subseteq \mathrm{Mon}_\tau \\ & \& \quad \forall s, s' \in \mathrm{Mon}_\sigma, s \leq_\sigma s' \to fs \leq_\tau fs'\} \\ f \leq_{\sigma\to\tau} g :\equiv \forall s \in \mathrm{Mon}_\sigma, & fs \leq_\tau gs. \end{aligned}$$

Note that $\leq_\sigma$ is generally now a preorder and not a partial order.

Lemma 4.4.2. Mon *is a combinatory clone.*

Proof. Once again, Mon is closed under application by design. We are then left to check the inclusion of the **K** and **S** combinators. We give a proof for **K**:

Let $s \in \mathrm{Mon}_\sigma$.

Sub-claim: $\boldsymbol{\lambda} y.s \in \mathrm{Mon}_{\tau\to\sigma}$.

Let $t \in \mathrm{Mon}_\tau$.

Sub-sub-claim: $(\boldsymbol{\lambda} y.s)t = s \in \mathrm{Mon}_\sigma$. This of course follows from assumption.

Let $t \leq_\tau t' \in \mathrm{Mon}_\tau$.

Sub-sub-claim: $s = (\boldsymbol{\lambda} y.s)t \leq_\sigma (\boldsymbol{\lambda} y.s)t' = s$. It is not hard to see that $\leq$ is always reflexive, so this holds.

Let $s \leq_\sigma s' \in \mathrm{Mon}_\sigma$.

Sub-claim: $\boldsymbol{\lambda} y.s \leq_{\sigma\to\tau} \boldsymbol{\lambda} y.s'$. Let $t \in \mathrm{Mon}_\tau$ be arbitrary. We must show that $s = (\boldsymbol{\lambda} y.s)t \leq_\sigma (\boldsymbol{\lambda} y.s')t = s'$. This follows from assumption.

As one might expect, a proof for **S** is long and tedious, and so it is omitted. □

It is not hard to see that Mon is extensional, so by virtue of 2.3.7 we have the following:

Claim 4.4.3. Mon *is a coatom.*

Proof. The argument is similar to that of 4.3.6. □

We now verify that Mon encompasses our old notion of monotone:

Lemma 4.4.4. *Let f, g be flat function of arity k. If for all $x_1, \ldots, x_k$ we have that*

$$f x_1, \ldots x_k \leq g x_1, \ldots x_k$$

then $f \leq_{0\to\ldots\to 0} g$.

Proof. Induction on k. If $k = 0$ the statement is trivial. Suppose then that $k > 0$. For $b \in \mathbf{B}$ arbitrary, we have that $fbx_2 \ldots x_k \leq gbx_2 \ldots x_k$ for arbitrary x's. Therefore, $fb \leq gb$ by inductive assumption. This held for arbitrary b, so $f \leq g$. □

Lemma 4.4.5. *Let* $\mathrm{Mon}_{\mathtt{flat}}$ *denote the* Mon *functions of flat type. For any f, $f \in \mathrm{Mon}_{\mathtt{flat}}$ if and only if for all $\langle x_1, \ldots, x_k\rangle \leq \langle y_1, \ldots, y_k\rangle$ we have that $fx_1 \ldots x_k \leq fy_1 \ldots y_k$.*

Proof. Induction on k. The statement is trivial for $k = 0$ so assume $k > 0$.

$\Rightarrow$ Suppose $f \in Mf$ and let $x, \bar{x} \leq y, \bar{y}$ be arbitrary vectors of boolean values. Since $x \leq y$ we then have that $fx \leq fy$. Since $fx \in \mathrm{Mon}_{\mathrm{flat}}$, by inductive assumption we have that $fx\bar{x} \leq fx\bar{y}$. Since $fx \leq fy$, we have that $fx\bar{y} \leq fy\bar{y}$. Thus, $fx\bar{x} \leq fx\bar{y} \leq fy\bar{y}$.

$\Leftarrow$ Suppose f has the right-hand property. We see that $f\bot$, $f\top$ each have the same property, so by inductive assumption, both are in $\mathrm{Mon}_{\mathrm{flat}}$. From the previous lemma, we see that $f\bot \leq f\top$, so we are done. □

We conclude by showing that $\mathrm{Mon}_{\mathrm{flat}}$ does not generate all of Mon.

Let $\mathbf{F} : (0 \to 0) \to 0$ denote the following function:

f	$\mathbf{F}(f)$
$\boldsymbol{\lambda} b.\bot$	$\top$
$\boldsymbol{\lambda} b.b$	$\top$
$\neg$	$\bot$
$\boldsymbol{\lambda} b.\top$	$\top$

This can be viewed as a sort of indicator function for the monotone elements of type $0 \to 0$.

Claim 4.4.6. $\mathbf{F} \in \mathrm{Mon}$.

Proof. Routine verification. □

Lemma 4.4.7. *There is no term $T : 0$ definable from $\mathrm{Mon}_{\mathrm{flat}}$ with a single free variable $f : 0 \to 0$ such that for all $g \in \mathbf{B}^{\mathbf{B}}$, $[\![T]\!]_{[f \mapsto g]} = \mathbf{F}(g)$.*

Proof. Suppose such a T exists. We then induct on the structure of T. Assume T is in long normal form. Since $\mathrm{Mon}_{\mathrm{flat}}$ has $\{\wedge, \vee, \top, \bot\}$ as a basis, there are four cases to consider:

Case i: $T = fT'$. We then have that

$$\begin{aligned}
[\![T]\!]_{[f \mapsto \boldsymbol{\lambda} x.\bot]} &= [\![fT']\!]_{[f \mapsto \boldsymbol{\lambda} x.\bot]} \\
&= (\boldsymbol{\lambda} x.\bot)[\![T']\!]_{[f \mapsto \boldsymbol{\lambda} x.\bot]} \\
&= \bot \\
&\neq \mathbf{F}(\boldsymbol{\lambda} x.\bot).
\end{aligned}$$

Case ii: T is one of the two constants. This is immediately ruled out, because $\mathbf{F}$ is not constant.

Case iii: $T = \wedge T_1 T_2$. We have that $[\![\wedge T_1 T_2]\!]_{[f \mapsto \neg]} = \mathbf{F}(\neg) = \bot$, so one of either T_1, T_2 must then evaluate to $\bot$. Without loss of generality suppose it is T_1. Since $\wedge T_1 T_2$ evaluates to true under substitution of the other three functions, we have that T_1 must evaluate to true under all of these substitutions as well. Thus, T_1 has the behavior of $\mathbf{F}$, a contradiction by inductive assumption.

Case iv: $T = \vee T_1 T_2$. We have that $[\![\vee T_1 T_2]\!]_{[f \mapsto \boldsymbol{\lambda} x.\bot]} = \mathbf{F}(\boldsymbol{\lambda} x.\bot) = \top$. Thus, one of either T_1 or T_2 must evaluate to $\top$ under this substitution. Without loss of generality let it be T_1. Since $\lambda f.T_1$ is defined from monotone functions, it must itself be monotone. Thus, T_1 evaluates to $\top$ as well when $\boldsymbol{\lambda} x.x$ or $\boldsymbol{\lambda} x.\top$ are substituted in for the variable f. However, we have that $[\![\vee T_1 T_2]\!]_{[f \mapsto \neg]} = \mathbf{F}(\neg) = \bot$, so T_1 must evaluate to $\bot$ when $\neg$ is substituted in for f. Thus, T_1 has the behavior of $\mathbf{F}$, a contradiction by inductive assumption.

□

Corollary 4.4.8. $\mathbf{F}$ *is not definable from* $\mathrm{Mon}_{\mathrm{flat}}$.

Proof. Suppose T defines F in normal form. We then have that $T = \lambda f^{0 \to 0}.T'$ for some T'. We then apply the previous lemma to T'. □

4.5 Self-dual Functions

Intuitively, the dual of an arity-k boolean function f is the function that results from negating the inputs of f and then negating the output:

$$\langle x_1, \ldots, x_k \rangle \mapsto \neg f(\neg x_1, \ldots, \neg x_k).$$

In this sense, the self-dual functions according to Post really are the functions which are equal to their own dual. It seems natural then to go about generalizing the notion of self-dual by instead generalizing the notion of a dual.

Definition 4.5.1. For each $\sigma \in \mathcal{T}$, define the **dualizer** $d_\sigma : \mathbf{B}_\sigma \to \mathbf{B}_\sigma$ inductively:

$$\begin{aligned} d_0 &:= \neg \\ d_{\sigma \to \tau} &:= (d_\tau \circ - \circ d_\sigma). \end{aligned}$$

For any $s : \sigma$, $d_\sigma s$ is called the **dual** of s.

Lemma 4.5.2. *For each* $\sigma \in \mathcal{T}$, d_σ *is an involution.*

Proof. Straightforward induction on σ. □

Claim 4.5.3. *Duals are distributive in the following sense: For each $\sigma, \tau \in T$, $f : \sigma \to \tau$ and $s : \sigma$, we have that*

$$d_\tau(fs) = (d_{\sigma\to\tau} f)(d_\sigma s).$$

Proof.

$$\begin{aligned}(d_{\sigma\to\tau} f)(d_\sigma s) &= (d_\tau \circ f \circ d_\sigma)(d_\sigma s)\\ &= d_\tau(f(d_\sigma(d_\sigma s)))\\ &= d_\tau(fs).\end{aligned}$$

□

Definition 4.5.4. We define SD, the combinatory clone of self-dual elements, as the set of elements which are equal to their own dual:

$$\mathrm{SD}_\sigma := \{s \in \mathbf{B}_\sigma \mid d_\sigma s = s\}.$$

Lemma 4.5.5. SD *is a combinatory clone.*

Proof. First, to show that it is closed under application, let $f \in \mathrm{SD}_{\sigma\to\tau}$, $s \in \mathrm{SD}_\sigma$. By 4.5.3 we then have that

$$d_\tau(fs) = (d_{\sigma\to\tau} f)(d_\sigma s) = fs.$$

The fact that **K** combinators are self-dual follows from a straightforward calculation:

$$\begin{aligned}&d_{\sigma\to\tau\to\sigma}(\lambda xy.x)\\ =\ &\lambda x.d_{\tau\to\sigma}(\lambda y.d_\sigma x).\\ =\ &\lambda xy.d_\sigma(d_\sigma x)\\ =\ &\lambda xy.x.\end{aligned}$$

And likewise for the **S** combinators:

$$
\begin{aligned}
& d_{(\sigma\to\tau\to\rho)\to(\sigma\to\tau)\to\sigma\to\rho}(\lambda xyz.xz(yz)) \\
&= \lambda x.d_{(\sigma\to\tau)\to\sigma\to\rho}(\lambda yz.(d_{\sigma\to\tau\to\rho}x)z(yz)) \\
&= \lambda x.d_{(\sigma\to\tau)\to\sigma\to\rho}(\lambda yz.d_{\tau\to\rho}(xd_\sigma z(yz))) \\
&= \lambda x.d_{(\sigma\to\tau)\to\sigma\to\rho}(\lambda yz.d_\rho((x(d_\sigma z))d_\tau(yz))) \\
&= \lambda xy.d_{\sigma\to\rho}(\lambda z.d_\rho((x(d_\sigma z))d_\tau(d_{\sigma\to\tau}yz))) \\
&= \lambda xy.d_{\sigma\to\rho}(\lambda z.d_\rho((x(d_\sigma z))d_\tau(d_\tau(y(d_\sigma z)))) \\
&= \lambda xyz.d_\rho(d_\rho((x(d_\sigma(d_\sigma z)))d_\tau(d_\tau(y(d_\sigma(d_\sigma z)))))) \\
&= \lambda xyz.xz(yz).
\end{aligned}
$$

□

Common sense tells us that $\bot$ and $\top$ must be treated symmetrically. After all, they are just two elements in the structureless set $\mathbf{B}$, so if we have a combinatory clone, we can switch the roles of $\bot$ and $\top$ in it and still have a combinatory clone. We make this precise:

Corollary 4.5.6. *Let $\mathcal{G}$ be a boolean combinatory clone. Then*

$$\mathcal{G}^d := \{d_\sigma s \mid \sigma \in \mathcal{T}, s \in \mathcal{G}_\sigma\}$$

is a combinatory clone.

Proof. By 4.5.5, the $\mathbf{K}$ and $\mathbf{S}$ combinators are self-dual. Since $\mathbf{K}_{\sigma,\tau} \in \mathcal{G}$, we then have that $\mathbf{K}_{\sigma,\tau} = d_{\sigma\to\tau\to\sigma}\mathbf{K}_{\sigma,\tau} \in \mathcal{G}^d$, and likewise for $\mathbf{S}$.

Suppose $f' = d_{\sigma\to\tau}f \in \mathcal{G}^d$ and $s' = d_\sigma s \in \mathcal{G}^d$. Since $\mathcal{G}$ is closed under application, $fs \in \mathcal{G}$. By 4.5.3, we then have that

$$
\begin{aligned}
f's' &= (d_{\sigma\to\tau}f)(d_\sigma s) \\
&= d_\tau(fs) \in \mathcal{G}^d.
\end{aligned}
$$

□

For instance, $\mathrm{FP}^d = \mathrm{TP}$.

Once more, let us verify that our generalized notion encompasses the old one:

Lemma 4.5.7. *Let* $\mathrm{SD}_{\mathrm{flat}}$ *denote the* SD *functions of flat type. For any flat f of arity k, the dual of f is given by $\boldsymbol{\lambda} x_1^0 \dots x_k^0.\neg(f(\neg x_1)\dots(\neg x_k))$.*

Proof. Induction on k. The statement is immediate for $k = 0$. Suppose then that f is of type $0 \to 0 \to \dots \to 0$. We then have that

$$\begin{aligned} d_{0\to 0\to \ldots \to 0}(f) &= \boldsymbol{\lambda} x_1^0 . d_{0\to \ldots \to 0}(f(d_0 x_1)) \\ &= \boldsymbol{\lambda} x_1 . d_{0\to \ldots \to 0}(f(\neg x_1)) \\ &= \boldsymbol{\lambda} x_2 \ldots x_k . \neg(f(\neg x_1)(\neg x_2)\ldots(\neg x_k)). \end{aligned}$$

□

The following alternative characterization of self-dual functions is useful:

Lemma 4.5.8. *For any σ, τ, $f : \sigma \to \tau$ is self-dual if and only if f preserves duals. That is, for all $s : \sigma$, $f(d_\sigma s) = d_\tau(fs)$.*

Proof. $\Rightarrow$ Suppose f is self-dual. Then for any s,

$$\begin{aligned} f(d_\sigma s) &= (d_{\sigma\to\tau} f)(d_\sigma s) \\ &= d_\tau(f(d_\sigma(d_\sigma(s)))) \\ &= d_\tau(fs). \end{aligned}$$

$\Leftarrow$ Suppose f preserves duals. We then have that

$$\begin{aligned} d_{\sigma\to\tau} f &= d_\tau \circ f \circ d_\sigma \\ &= d_\tau \circ d_\tau \circ f \\ &= f. \end{aligned}$$

□

Claim 4.5.9. *For any σ, SD_σ is non-empty if and only if σ is a tautology.*

Proof. We use induction on σ. The atomic case is obvious. Suppose then that our type is $\sigma \to \tau$. If τ is a tautology, then by the inductive hypothesis, there is some $t \in \mathrm{SD}_\tau$. We then see that $\sigma \to \tau$ is a tautology and $\lambda x.t \in \mathrm{SD}_{\sigma\to\tau}$.

Suppose then that τ is not a tautology, and thus equivalent to 0. If σ is a tautology, then by inductive hypothesis, there is some $s \in \mathrm{SD}_\sigma$. Since $\sigma \to \tau$ is then not a tautology, suppose for contradiction that $f : \sigma \to \tau$ is self-dual. We then have that $fs \in \mathrm{SD}_\tau$, a contradiction by inductive hypothesis.

We are then left to consider the case where σ and τ are both non-tautologies. Since $\sigma \to \tau$ is then a tautology, we must provide some self-dual $f : \sigma \to \tau$. Fix some $t : \tau$ and let t' be the dual of t. By inductive hypothesis, $t \neq t'$. Since σ has no self-dual elements, d_σ partitions σ into two-element subsets. Within each two-element subset, send one to t and the other (its dual) to t'. In this fashion we construct a function $f : \sigma \to \tau$ which one can easily verify is self-dual.

□

Since SD doesn't contain any non-tautological elements, we cannot say that SD is maximal, since it is below the full combinatory clone of tautological elements. Nonetheless, we have some results:

Claim 4.5.10. *Let $b \in \mathbf{B}$. $\overline{\mathrm{SD} \cup \{b\}} = \mathcal{G}(\mathbf{B})$.*

Proof. Since $\neg \in \mathrm{SD}_{0\to 0}$, assume without loss of generality that $b = \bot$. Consider the function $f : 0 \to 0 \to 0 \to 0$ which sends $\bot$ to $\to$ and $\top$ to $d_{0\to 0\to 0}(\to)$. Clearly, $f \in \mathrm{SD}_{0\to 0\to 0\to 0}$, so $f\bot$ provides a definition of $\to$. Since $\bot, \to$ is functionally complete, we are done. □

Claim 4.5.11. *Let $s : \sigma$ where σ is a non-tautology. Then $\overline{\mathrm{SD} \cup \{s\}} = \mathcal{G}(\mathbf{B})$.*

Proof. From the previous claim, it suffices to define a boolean. Since $\sigma \to 0$ is a tautology, there is some self-dual $f : \sigma \to 0$, and so fs gives us the definition of a boolean. □

Claim 4.5.12. SD *is covered by $\mathcal{G}^{taut}(\mathbf{B})$.*

Proof. Let $f : \sigma \to \tau$ be a non-self-dual element of tautological type. We shall show that f can be used to define a function of flat type which is not self-dual. By claim 4.1.7, this will be enough to generate all of $\mathcal{G}^{\mathrm{taut}}(\mathbf{B})$.

Since f is non-self-dual, there is some $s : \sigma$ such that $f(d_\sigma s) \neq d_\tau(fs)$. Let $g : 0 \to \sigma$ send $\bot$ to s and $\top$ to $d_\sigma s$. Clearly, $g \in \mathrm{SD}_{0\to\sigma}$. Now we consider two case:

(i) fs is self-dual. In this case define $h : \tau \to 0 \to 0 \to 0$ by sending t to π_1 and everything else to π_2. We can see that h is self-dual, so $h \circ f \circ g \in \overline{\mathrm{SD} \cup \{f\}}$. However, $h(f(g\bot)) = \pi_1$ and $h(f(g\top)) = \pi_2$, so $h \circ f \circ g$ is not self-dual.

(ii) fs is not self-dual. In this case, let $h : \tau \to 0 \to 0$ send fs to $\lambda x.\bot$, $d_\tau(fs)$ to $\lambda x.\top$, and everything else to $\lambda x.x$. Clearly, $h \in \mathrm{SD}_{\tau\to 0\to 0}$. Observe that $h(f(g(\bot))) = \lambda x.\bot$, and $h(f(g(\top)))$ is either $\lambda x.x$ or $\lambda x.\bot$. In either case, $h \circ f \circ g$ is not self-dual. □

Remark 4.5.13. Since SD is not extensional, it provides a counterexample to the converse of 2.3.7.

4.5.1 G-sets

We will now give an alternative categorical characterization of the self-dual functions.

Fix a group $G := \langle G, e, * \rangle$. Recall that a **group action** from G to a set X is a map $\cdot : G \times X \to X$ such that

1. $\forall x \in X$, $e \cdot x = x$

2. $\forall a, b \in G,\ x \in X,\ a \cdot (b \cdot x) = (a * b) \cdot x$.

A G**-set** is a pair $\langle X, \cdot \rangle$ such that $\cdot$ is a group action from G to X. Given G-sets $\langle X, \cdot \rangle$, $\langle Y, \circ \rangle$, a G**-set homomorphism** is a function $f : X \to Y$ such that for all $a \in G$ and $x \in X$,

$$f(a \cdot x) = a \circ f(x).$$

Given G-sets $\langle X, \cdot \rangle$, $\langle Y, \circ \rangle$, we may define a group action on Y^X as follows:

$$\langle a, f \rangle \mapsto \boldsymbol{\lambda} x. a^{-1} \circ f(a \cdot x).$$

In fact, this defines an exponential object, and so G-sets form a cartesian-closed category.

Let $\mathbf{B}_\oplus$ denote the group $\langle \mathbf{B}, \bot, \oplus \rangle$. Any group G can be thought of as acting on its own underlying set, with the action given by the group operation. With that in mind let $\mathfrak{B}$ denote the $\mathbf{B}_\oplus$-Set $\langle \mathbf{B}, \oplus \rangle$.

Given $\sigma \in \mathcal{T}$, define $\mathfrak{B}_\sigma$ inductively:

$$\begin{aligned} \mathfrak{B}_0 &:= \mathfrak{B} \\ \mathfrak{B}_{\sigma \to \tau} &:= \mathfrak{B}_\tau^{\mathfrak{B}_\sigma}. \end{aligned}$$

Lemma 4.5.14. *For each $\sigma \in \mathcal{T}$, let $\cdot$ denote the group action associated with $\mathfrak{B}_\sigma$. Then $\top \cdot s = d_\sigma s$.*

Proof. Induction on σ. In the atomic case, $\top \cdot s = \top \oplus s = \neg s = d_0 s$.

Suppose then that $f : \sigma \to \tau$. Then

$$\begin{aligned} \top \cdot f &= \boldsymbol{\lambda} x. \top^{-1} \cdot (f(\top \cdot x)) \\ &= \boldsymbol{\lambda} x. \top \cdot (f(d_\sigma x)) \\ &= \boldsymbol{\lambda} x. d_\tau (f(d_\sigma x)) \\ &= d_{\sigma \to \tau} f. \end{aligned}$$

□

Lemma 4.5.15. *$f : \mathbf{B}_\sigma \to \mathbf{B}_\tau$ is self-dual if and only if $f \in \mathrm{Hom}(\mathfrak{B}_\sigma, \mathfrak{B}_\tau)$.*

Proof. We use the previous lemma and 4.5.3.

⇒ Suppose that f is self-dual. For any $s \in \mathbf{B}_\sigma$ we must verify that $f(\bot \cdot s) = \bot \cdot fs$ and $f(\top \cdot s) = \top \cdot fs$. Since $\bot$ is the identity, the first equation follows from the definition of a group action for arbitrary f. We are left to verify the second:

$$\begin{aligned} f(\top \cdot s) &= f(d_\sigma s) \\ &= d_\tau(fs) \\ &= \top \cdot fs. \end{aligned}$$

$\Leftarrow$ Suppose that $f \in \mathrm{Hom}(\mathfrak{B}_\sigma, \mathfrak{B}_\tau)$. For arbitrary s we have that

$$\begin{aligned} f(d_\sigma s) &= f(\top \cdot s) \\ &= \top \cdot fs \\ &= d_\tau(fs). \end{aligned}$$

Thus, f is self-dual.

□

Chapter 5

The Addition of Products

We now consider a natural extension of the simply-typed lambda calculus with explicit product types, and the resulting algebra of functions. A nice discussion of this system may be found in Chapters 3 and 4 of [3].

5.1 Simply Typed Lambda Calculus with Products

5.1.1 Syntax

Fix a set of atomic types $\mathcal{A}$. Define the types over $\mathcal{A}$ to be

$$\mathcal{T} ::= \mathcal{T} \times \mathcal{T} \mid \mathcal{T} \to \mathcal{T} \mid \alpha \ \ (\alpha \in \mathcal{A}).$$

Convention 5.1.1. $\times$ is given precedence over $\to$. That is, by $\sigma \times \tau \to \rho$ we will mean $(\sigma \times \tau) \to \rho$.

Convention 5.1.2. We take $\times$ to be right-associative. That is, by $\sigma_1 \times \ldots \times \sigma_N$ we will mean $\sigma_1 \times (\sigma_2 \times (\ldots \times (\sigma_{N-1} \times \sigma_N)))$.

Lambda terms are defined as before, but with three additional constructions:

- If $S : \sigma$ and $T : \tau$, then $\langle S, T \rangle : \sigma \times \tau$.
- If $P : \sigma \times \tau$, then $\pi_1 P : \sigma$ and $\pi_2 P : \tau$.

In addition to the usual beta and eta reductions, we introduce three new reductions:

- The **projection reductions:** $\pi_1 \langle S, T \rangle \rightsquigarrow_{\mathrm{P}} S$ and $\pi_2 \langle S, T \rangle \rightsquigarrow_{\mathrm{P}} T$.
- The **surjective pairing reduction:** $\langle \pi_1 P, \pi_2 P \rangle \rightsquigarrow_{\mathrm{SP}} P$.

Definition 5.1.3. A term T is said to be **normal** if it cannot be reduced further using beta, eta, or projection reductions.

Claim 5.1.4. *Every term has a normal form which is unique.*

Proof. See sections 4.1-3 of [3]. □

We will take $T \rightsquigarrow T'$ to mean that there is a finite sequence of beta, eta, projection, and surjective pairing reductions from T to T'.

The pairing and projection operations can naturally be extended to higher arities:

Claim 5.1.5. *Let $n \geq 2$ and $\sigma_1, \ldots, \sigma_n \in \mathcal{T}$. There is an arity n pairing term $\langle -, \ldots, - \rangle : \sigma_1 \to \ldots \to \sigma_n \to \sigma_1 \times \ldots \times \sigma_n$ and for each $1 \leq i \leq n$ a projection term $\pi_i^n : \sigma_1 \times \ldots \times \sigma_n \to \sigma_i$ which satisfy the following generalized pairing and surjective pairing reductions:*

$$\pi_i^n \langle s_1, \ldots, s_n \rangle \rightsquigarrow_P s_i$$
$$\langle \pi_1^n v, \ldots, \pi_n^n v \rangle \rightsquigarrow_{SP} v.$$

We will wish to examine the simply typed lambda calculus as a fragment of the lambda calculus with products. To this end, we introduce the following definitions:

Definition 5.1.6. A type $\sigma \in \mathcal{T}$ is said to be **simple** if it is built up from $\mathcal{A}$ and $\to$.

Definition 5.1.7. A term S of type σ is said to be **simple** if all of its variables are of simple type, and it does not contain any instances of pairing or projections.

Notice that simple terms are necessarily of simple type.

In some sense, the lambda calculus with products is conservative over the simply-typed lambda calculus. In order to show this, we prove a couple of lemmas:

Lemma 5.1.8. *Let $P : \sigma \times \tau$ be in normal form whose free variables are all simple. Then P is a pair.*

Proof. Induction on P. P cannot be a variable, since the type of P is not simple. P cannot be a lambda abstraction, since P is a product. If $P = \pi_i Q$, then Q is a pair by inductive hypothesis. However, this contradicts the fact that P is normal. Lastly, suppose $P = MN$ for $M : \rho \to \sigma \times \tau$ and $N : \rho$. M cannot be a variable, since M is not of simple type. M cannot be a lambda abstraction, since P would then not be normal. M cannot be a pair, since its type is not a product. Thus, M must itself be a result of function application. Continuing this argument, we arrive at the conclusion that $MN = vP_1 \ldots P_n N$ for some variable v and $P_1, \ldots, P_n$. Since $MN : \sigma \times \tau$, we have that $v : \sigma_1 \to \ldots \sigma_n \to \sigma_{n+1} \to \sigma \times \tau$. This contradicts the fact that as a free variable, v must be of simple type. □

Lemma 5.1.9. *Let $T : \tau$ be a term of simple type whose free variables are all of simple type. Then the normal form of T is a simple term.*

Proof. Induction on T. If T is a variable, then of course it is simple. If $T = \lambda x.U$, then x is a variable of simple type, and so U is a term of simple type with free variables all simple, and so by inductive hypothesis, it is a simple term. T cannot be a pairing, since its type is simple. If $T = \pi_i U$, then by the previous lemma, U is a pairing, which violates normality. Lastly, suppose $T = MN$ with $M : \sigma \to \tau$ and $N : \sigma$. If M is a variable, then σ must also be a simple type, and so N is a simple term by inductive hypothesis. M cannot be a lambda abstraction, since that would violate normality. M cannot be a pair, since it is not a product. The case of M being the result of a projection is once again ruled out by the previous lemma. Thus, M is also a result of function application. Once again, we continue this argument and arrive at the conclusion that $MN = vP_1 \dots P_n N$ for some variable v and $P_1, \dots, P_n$. Since v must be of simple type, $P_1, \dots, P_n N$ must all be of simple type. By inductive hypothesis, they are simple terms, and so we are done. □

5.1.2 Semantics

Fix $\mathbf{X} = (X_\alpha)_{\alpha \in \mathcal{A}}$. We define $\mathbf{X}_\sigma$ as before, additionally stipulating that

$$\mathbf{X}_{\sigma \times \tau} := \mathbf{X}_\sigma \times \mathbf{X}_\tau.$$

Pairs are interpreted as ordered pairs, with π_1, π_2 interpretated as the first and second coordinates:

- $[\![\langle M, N \rangle]\!]_g := \langle [\![M]\!]_g, [\![N]\!]_g \rangle$
- $[\![\pi_1 P]\!]_g :=$ the first coordinate of $[\![P]\!]_g$, and $[\![\pi_2 P]\!]_g :=$ the second coordinate of $[\![P]\!]_g$.

As before, we have that reduction preserves denotation:

Claim 5.1.10. *Suppose $M \rightsquigarrow N$. Then for any $\mathbf{X}$ and g we have that $[\![M]\!]_g = [\![N]\!]_g$.*

5.1.3 Congruent Types

We wish to show that every type can in a sense be "rewritten" as a product of simple types. This will allow us to "factor" arbitrary elements as finite tuples of simple elements. To this end, we introduce the notion of congruence:

Definition 5.1.11. Let $\sigma, \tau \in \mathcal{T}$. We say that σ and τ are **congruent** ($\sigma \cong \tau$) when there are closed terms $F : \sigma \to \tau$ and $G : \tau \to \sigma$ such that for all $s : \sigma$, $t : \tau$, $G(Fs) \rightsquigarrow s$ and $F(Gt) \rightsquigarrow t$.

Claim 5.1.12. *(i) $\cong$ is an equivalence relation.*

(ii) $\cong$ *is a congruence with respect to the type operations* $\to$ *and* $\times$*: If* $\sigma \cong \sigma'$ *and* $\tau \cong \tau'$ *then* $\sigma \to \tau \cong \sigma' \to \tau'$ *and* $\sigma \times \tau \cong \sigma' \times \tau'$*.*

Claim 5.1.13. $\times$ *is associative:* $(\sigma \times \tau) \times \rho \cong \sigma \times \tau \times \rho$*, and more generally,*

$$(\sigma_1 \times \ldots \times \sigma_n) \times (\tau_1 \times \ldots \times \tau_m) \cong \sigma_1 \times \ldots \times \sigma_n \times \tau_1 \times \ldots \times \tau_m.$$

Lemma 5.1.14. *For any* $\sigma_1, \ldots, \sigma_n, \tau$ *we have that*

$$\sigma_1 \times \ldots \times \sigma_n \to \tau \cong \sigma_1 \to \ldots \to \sigma_n \to \tau.$$

Proof. Define

$$F := \lambda f^{\sigma_1 \times \ldots \times \sigma_n \to \tau} x_1^{\sigma_1} \ldots x_n^{\sigma_n}.f\langle x_1, \ldots, x_n\rangle$$

and

$$G := \lambda g^{\sigma_1 \to \ldots \to \sigma_n \to \tau} v^{\sigma_1 \times \ldots \times \sigma_n}.g(\pi_1^n v) \ldots (\pi_n^n v).$$

We then have that

$$\begin{aligned} G(Ff) &\rightsquigarrow_\beta (\lambda g v.g(\pi_1^n v) \ldots (\pi_n^n v))(\lambda x_1 \ldots x_n.f\langle x_1, \ldots, x_n\rangle) \\ &\rightsquigarrow_\beta \lambda v.(\lambda x_1 \ldots x_n.f\langle x_1, \ldots, x_n\rangle)(\pi_1^n v) \ldots (\pi_n^n v) \\ &\rightsquigarrow_\beta \lambda v.f\langle \pi_1^n v, \ldots, \pi_n^n v\rangle \\ &\rightsquigarrow_{\mathrm{SP}} \lambda v.fv \\ &\rightsquigarrow_\eta f \end{aligned}$$

and

$$\begin{aligned} F(Gg) &\rightsquigarrow_\beta (\lambda f x_1 \ldots x_n.f\langle x_1, \ldots, x_n\rangle)(\lambda v.g(\pi_1^n v) \ldots (\pi_n^n v)) \\ &\rightsquigarrow_\beta \lambda x_1 \ldots x_n.(\lambda v.g(\pi_1^n v) \ldots (\pi_n^n v))\langle x_1, \ldots, x_n\rangle \\ &\rightsquigarrow_\beta \lambda x_1 \ldots x_n.g(\pi_1^n\langle x_1, \ldots, x_n\rangle) \ldots (\pi_n^n\langle x_1, \ldots, x_n\rangle) \\ &\rightsquigarrow_{\mathrm{P}} \lambda x_1 \ldots x_n.gx_1, \ldots x_n \\ &\rightsquigarrow_\eta g. \end{aligned}$$

□

Lemma 5.1.15. *For any* $\sigma, \tau_1, \ldots, \tau_m \in \mathcal{T}$ *we have that*

$$\sigma \to \tau_1 \times \ldots \times \tau_m \cong (\sigma \to \tau_1) \times \ldots \times (\sigma \to \tau_m).$$

Proof. Define

$$F := \lambda f^{\sigma \to \tau_1 \times \ldots \times \tau_m}.\langle \lambda x^\sigma.\pi_1^n(fx), \ldots, \lambda x^\sigma.\pi_n^n(fx)\rangle$$

and

$$G := \lambda V^{(\sigma\to\tau_1)\times\ldots\times(\sigma\to\tau_m)} y^\sigma.\langle(\pi_1^n V)y,\ldots,(\pi_n^n V)y\rangle.$$

We then have that

$$\begin{aligned}
G(Ff) &\rightsquigarrow_\beta (\lambda V y.\langle(\pi_1^n V)y,\ldots,(\pi_n^n V)y\rangle)\langle\lambda x^\sigma.\pi_1^n(fx),\ldots,\lambda x^\sigma.\pi_n^n(fx)\rangle\\
&\rightsquigarrow_\beta \lambda y.\langle\pi_1^n\langle\lambda x.\pi_1^n(fx),\ldots,\lambda x.\pi_n^n(fx)\rangle y,\ldots,\\
&\qquad \pi_n^n\langle\lambda x.\pi_1^n(fx),\ldots,\lambda x.\pi_n^n(fx)\rangle y\rangle\\
&\rightsquigarrow_{\mathrm{P}} \lambda y.\langle(\lambda x.\pi_1^n(fx))y,\ldots,(\lambda x.\pi_n^n(fx))y\rangle\\
&\rightsquigarrow_\beta \lambda y.\langle\pi_1^n(fy),\ldots,\pi_n^n(fy)\rangle\\
&\rightsquigarrow_{\mathrm{SP}} \lambda y.fy\\
&\rightsquigarrow_\eta f
\end{aligned}$$

and

$$\begin{aligned}
F(GV) &\rightsquigarrow_\beta (\lambda f.\langle\lambda x.\pi_1^n(fx),\ldots,\lambda x.\pi_n^n(fx)\rangle)(\lambda y.\langle(\pi_1^n V)y,\ldots,(\pi_n^n V)y\rangle)\\
&\rightsquigarrow_\beta \langle\lambda x.\pi_1^n((\lambda y.\langle(\pi_1^n V)y,\ldots,(\pi_n^n V)y\rangle)x,\ldots,\\
&\qquad \lambda x.\pi_n^n((\lambda y.\langle(\pi_1^n V)y,\ldots,(\pi_n^n V)y\rangle)x)\rangle\\
&\rightsquigarrow_\beta \langle\lambda x.\pi_1^n(\langle(\pi_1^n V)x,\ldots,(\pi_n^n V)x\rangle),\ldots,\\
&\qquad \lambda x.\pi_n^n(\langle(\pi_1^n V)x,\ldots,(\pi_n^n V)x\rangle)\rangle\\
&\rightsquigarrow_{\mathrm{P}} \langle\lambda x.(\pi_1^n V)x,\ldots,\lambda x.(\pi_n^n V)x\rangle\\
&\rightsquigarrow_\eta \langle\pi_1^n V,\ldots,\pi_n^n V\rangle\\
&\rightsquigarrow_{\mathrm{SP}} V.
\end{aligned}$$

□

Lemma 5.1.16. *For every $\sigma \in \mathcal{T}$ there are simple types $\sigma_1,\ldots,\sigma_N$ such that $\sigma \cong \sigma_1\times\ldots\times\sigma_N$.*

Proof. Induction on σ. If $\sigma = \alpha$ atomic, then α is already simple.

Suppose $\sigma = \tau\times\rho$. By inductive assumption, $\tau \cong \tau_1\times\ldots\times\tau_N$ and $\rho \cong \rho_1\times\ldots\times\rho_M$, where τ_i,ρ_j are all simple. We then have that

$$\begin{aligned}
\sigma &= \tau\times\rho\\
&\cong (\tau_1\times\ldots\times\tau_N)\times(\rho_1\times\ldots\times\rho_M)\\
&\cong \tau_1\times\ldots\times\tau_N\times\rho_1\times\ldots\times\rho_M.
\end{aligned}$$

Lastly, suppose $\sigma = \tau\to\rho$ with $\tau \cong \tau_1\times\ldots\times\tau_N$ and $\rho\cong\rho_1\times\times\ldots\times\rho_M$. We then have that

$$\begin{aligned}\sigma &= \tau \to \rho \\ &\cong (\tau_1 \times \ldots \times \tau_N) \to (\rho_1 \times \ldots \times \rho_M) \\ &\cong (\tau_1 \times \ldots \times \tau_N \to \rho_1) \times \ldots \times (\tau_1 \times \ldots \times \tau_N \to \rho_M) \\ &\cong (\tau_1 \to \ldots \to \tau_N \to \rho_1) \times \ldots \times (\tau_1 \to \ldots \to \tau_N \to \rho_M).\end{aligned}$$

□

5.2 Combinatory Clones with Products

Given a family of base sets $\mathbf{X} = (X_\alpha)_{\alpha \in \mathcal{A}}$, we extend the notion of a generalized clone over $\mathbf{X}$ to be a collection of functions and elements containing the $\mathbf{K}$ and $\mathbf{S}$ combinators over all types in $\mathcal{T}$, closed under application, and furthermore containing for each $\sigma, \tau \in \mathcal{T}$

1. the projection functions $\pi_1^{\sigma,\tau} \in \mathbf{X}_{\sigma \times \tau \to \sigma}$, $\pi_2^{\sigma,\tau} \in \mathbf{X}_{\sigma \times \tau \to \tau}$.
2. the pairing function $\langle -, - \rangle_{\sigma,\tau} : \mathbf{X}_{\sigma \to \tau \to \sigma \times \tau}$.

For notational convenience we will leave out type annotations.

As before, the combinatory clones with products form a lattice which we will denote by $\mathbf{CCl}^\times(\mathbf{X})$.

5.2.1 Relating Combinatory Clones with and without products

As a consequence of 5.1.16, we obtain the following result:

Corollary 5.2.1. *Let $s \in \mathbf{X}_\sigma$. Then s is interdefinable with $s_1, \ldots, s_n$ of simple type.*

Proof. By 5.1.16, $\sigma \cong \sigma_1 \times \ldots \times \sigma_n$ where $\sigma_1, \ldots, \sigma_n$ are simple. Thus, there are closed terms $F : \sigma \to \sigma_1 \times \ldots \sigma_n$ and $G : \sigma_1 \times \ldots \times \sigma_n \to \sigma$ witnessing this congruence. Let $\langle s_1, \ldots, s_n \rangle := [\![F]\!](s)$. We then have that

$$s_i = \pi_i^n([\![F]\!](s))$$

and

$$\begin{aligned}s &= [\![G]\!]([\![F]\!](s)) \\ &= [\![G]\!](\langle s_1, \ldots, s_n \rangle).\end{aligned}$$

Thus, s and $s_1, \ldots, s_n$ are interdefinable.

□

As in 2.2, we have maps

$$\Phi : \mathbf{CCl}^{\times}(\mathbf{X}) \to \mathbf{CCl}(\mathbf{X})$$
$$\Psi : \mathbf{CCl}(\mathbf{X}) \to \mathbf{CCl}^{\times}(\mathbf{X})$$

where Φ denotes the operation of forgetting all elements whose types involve products, and Ψ denote the closure operation.

Lemma 5.2.2. *For all* $\mathcal{G} \in \mathbf{CCl}(\mathbf{X})$, $\Phi(\Psi(\mathcal{G})) = \mathcal{G}$.

Proof. $\mathcal{G} \subseteq \Phi(\Psi(\mathcal{G}))$ is obvious. Suppose then that $t \in \Phi(\Psi(\mathcal{G}))$. There is then a term T with free variable occurrences $t_1, \ldots, t_n$ corresponding to functions in $\mathcal{G}$ which represents t. By 5.1.9, we may take this T to be a simple term. Thus, t can be defined from $t_1, \ldots, t_n$ in the (non-product) combinatory clone setting. Thus, $t \in \mathcal{G}$. □

Lemma 5.2.3. *For all* $\mathcal{G} \in \mathbf{CCl}^{\times}(\mathbf{X})$, $\Psi(\Phi(\mathcal{G})) = \mathcal{G}$.

Proof. $\Psi(\Phi(\mathcal{G})) \subseteq \mathcal{G}$ is obvious. Suppose then that $t \in \mathcal{G}$. By 5.2.1, t is interdefinable with simple elements $t_1, \ldots, t_n$. Since each such t_i is simple and definable from t, we have that $t_1, \ldots, t_n \in \Phi(\mathcal{G}))$ and consequently $\Psi(\Phi(\mathcal{G}))$. Since t can be defined by $t_1, \ldots, t_n$, we conclude that $t \in \Psi(\Phi(\mathcal{G}))$. □

Corollary 5.2.4. $\mathbf{CCl}(\mathbf{X})$ *and* $\mathbf{CCl}^{\times}(\mathbf{X})$ *are isomorphic.*

Conclusion

We have seen that the simply typed lambda calculus provides a natural framework for extending the study of clones into the higher-order realm. After initiating the study of combinatory clones, we have established results showing that clones form the non-higher order fragment of combinatory clones. In the case of combinatory clones over finite sets, we have seen that the elements of flat type are sufficient to generate all of the higher-order elements as well. Finally, we have seen that the addition of product types does not essentially alter the study of combinatory clones. Precisely, all the information about a combinatory clone with products is already present in its elements which are of simple type.

In the study of boolean combinatory clones, some early work has been done, but there is still much work to be done toward a complete classification of the lattice of boolean combinatory clones or even just the coatoms. We have shown that multiple combinatory boolean clones may project onto the same boolean clone. In four of the five coatomic boolean clones, we have given corresponding coatomic combinatory clones with categorical interpretations. In two of those, we have shown that they are generated by their flat elements, and in one we have shown this to not be the case.

We close with some open questions:

- What is an example of a coatomic boolean clone corresponding to the affine functions? Does it have a nice categorical interpretation?
- Are there multiple coatomic boolean clones corresponding to the clones of monotone or affine functions?
- Is there a $\mathcal{G}$ coatomic such that $\Phi(\mathcal{G})$ is not a coatom?
- By virtue of his complete classification, Post showed that the lattice of boolean clones is countable. However, the lattice of clones over a three element set is already continuum-sized. What is the size of $\mathbf{CCl}(\mathbf{B})$? If there is a strong affinity between clones and combinatory clones, we might suspect it to be countable as well. On the other hand, one could imagine that the presence of higher-order types allows for enough pathology to make it uncountable.

- Post also showed through complete classification that all boolean clones are finitely based. Does this hold for combinatory boolean clones as well? The pairing result 2.3.14 would then imply that each combinatory clone is generated by a single element, radically taming the general problem of their classificiation.

Bibliography

[1] Barendregt, H., Statman, R., Dekkers, R. W. (2013). *Lambda Calculus with Types*. Cambridge: Cambridge University Press.

[2] Diego, A. *Sur les algébras de Hilbert*. Gauthier-Villars, Paris, 1966.

[3] Girard, J. (1989). *Proofs and types*. Cambridge: Cambridge University Press.

[4] Grzegorczyk, A. *Recursive Functionals in All Finite Types*. Fund. Math., 54 (1964), 73-93.

[5] Post, E.L. *The Two-Valued Iterative Systems of Mathematical Logic*, Annals of Mathematics Studies 5, 1941.

[6] Urquhart, A. *Implicational Formulas in Intuitionistic Logic*. The Journal of Symbolic Logic, Vol. 39, Number 4, Dec. 1974.

[7] Zaionc, M. *On the λ Definable Higher Order Boolean Operations*. Fundamenta Informaticae, Vol. 12, Number 1, March 1989.