Summer 8-2015

Some Results on Classical Semantics and Polymorphic Types

William J. Gunther
Carnegie Mellon University

Carnegie Mellon University

MELLON COLLEGE OF SCIENCE

THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF Doctor of Philosophy
in Mathematical Sciences

TITLE Some Results on Classical Semantics and Polymorphic Types

PRESENTED BY William Gunther

ACCEPTED BY THE DEPARTMENT OF Mathematical Sciences

Richard Statman	August 2015
MAJOR PROFESSOR	DATE
Thomas Bohman	August 2015
DEPARTMENT HEAD	DATE

APPROVED BY THE COLLEGE COUNCIL

Frederick J. Gilman	August 2015
DEAN	DATE

Carnegie Mellon University
Department of Mathematical Sciences

Doctoral Dissertation

Some Results on Classical Semantics and Polymorphic Types

William J. Gunther

August 2015

Submitted to the Department of Mathematical Sciences
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Mathematical Sciences

Dissertation Committee

Richard Statman, chair
Jeremy Avigad
Karl Crary
James Cummings

Abstract

In the first chapter we consider the simply typed λ-calculus over one ground type with a discriminator δ which distinguishes terms, augmented additionally with an existential quantifier and a description operator, all of lowest type. First we provide a proof of a folklore result which states that a function in the full type structure of $[n]$ is $\lambda\delta$-definable from the description operator and existential quantifier if and only if it is symmetric. This proof uses only elementary facts from algebra and a way to reduce arbitrary functions to functions of lowest type via a theorem of Henkin. Then we prove a necessary and sufficient condition for a function on $[n]$ to be $\lambda\delta$-definable without the description operator or existential quantifier, which requires a stronger notion of symmetry.

In the second chapter, we consider the system Q_0, extensively studied by Andrews [And72, And02]. This system is an axiomatic system of higher-order logic in the language of the simply typed lambda calculus, augmented with equality operators over every type and a description operator. We prove that the axiom of extensionality is independent from the other axioms by constructing an explicit model.

In the third chapter, we consider the question of the typability of a particular class of terms in system F, the polymorphic typed lambda calculus of Girard [Gir72] and Reynolds [Rey74]. $\omega \equiv \lambda x.xx$ is a classic example of a term which is not typeable in the simply typed lambda calculus, but is typable in (most) polymorphic systems, like system F. The question is if MM has a type in system F is it true that ωM has a type in system F? All terms which can be typed as inputs for ω require being typable with a free leftmost path. We will prove that if M is a normal term whose type requires a bound leftmost path, then the term MM is not strongly normalizing (thus not typable in system F). We will then establish that the undecidability of the question: given a normal term M typable in system F, is MM (or ωM) typable in system F?

Contents

Acknowledgements

This could not be completed without a lot of help from many people. I would like to begin by thanking my advisor, Richard Statman, who helped immeasurably, and whose advice and guidance over the last five years has been invaluable. There are many other faculty and staff in the mathematics department at Carnegie Mellon who taught and helped me over the years, and who work to create the amazing environment here, such as Stella Andreoletti, Jeremy Avigad, James Cummings, and William Hrusa.

During my undergraduate experience at Virginia Tech and Rutgers University, Griff Elder and Samuel Coskey stand out as two teachers without whom I would never found my fascination with mathematics and logic.

I am also very grateful for all of the graduate students who I have spent time thinking about mathematics with, and whom I have learned so much from, especially Emily Allen, Deepak Bal, William Boney, Lisa Espig, Samantha Gottlieb, Brian Kell, Chistopher Lambie-Hanson, Paul McKenney, Marla Slusky, and Brendan Sullivan.

Finally, and most importantly, I would like to thank my friends and family; my parents, Elwood and Patricia Gunther, are owed a special thanks. Without their support over these many years I would undoubtedly not have been able to complete this work.

Chapter 1

Survey of the lambda calculus

We begin with a rather terse introduction to the lambda calculus. For a complete treatment of the lambda calculus see [Bar84] and [BDS13] for the untyped and typed lambda calculus respective. Throughout the thesis, the concepts from this section will used. We begin with an exploration of the untyped lambda calculus, followed by the simply typed lambda calculus. We will define the polymorphic lambda calculus, system F, in Section 4.1.

The **lambda calculus** is a formal system which seeks to capture functions by way of abstraction and application. The lambda calculus was first proposed by Church in in [Chu32] as a foundational system for mathematics. This foundational system was found to be inconsistent by Kleene and Rosser in [KR35], proving a variation of Russell's Paradox now called the Kleene–Rosser Paradox.

After the inconsistency, the formal system of the **untyped lambda calculus** was used to by Church to describe effective computations. This system was used resolved Hilbert's *Entscheidungsproblem*, the decision problem [Chu36b].

1.0.1 The untyped lambda calculus

To begin, we have a countable collection of variables,

$$V = \{x, y, z, x_1, x_2, \ldots\}.$$

We define the set of terms of the lambda calculus, which we write Λ, inductively as follows:

$$V \subseteq \Lambda, \qquad \text{(variable)}$$
$$\text{if } M, N \in \Lambda \text{ then } (MN) \in \Lambda, \text{ and} \qquad \text{(application)}$$
$$\text{if } M \in \Lambda \text{ and } x \in V \text{ then } (\lambda x.M) \in \Lambda. \qquad \text{(abstraction)}$$

We write $M \equiv N$ if M and N are syntactically identical.

Definition 1. If M is a term, we define the set of **free variables** of M, denoted $\mathrm{FV}(M)$, inductively as follows:

- if $M \equiv y$, then $\mathrm{FV}(M) = \{y\}$;
- if $M \equiv PQ$, then $\mathrm{FV}(M) = \mathrm{FV}(P) \cup \mathrm{FV}(Q)$; and
- if $M \equiv \lambda x.P$, then $\mathrm{FV}(M) = \mathrm{FV}(P) \setminus \{x\}$.

Every variable in M either appears free or is bound to some lambda.

Remark 1 (Variable conventions). We follow the convention set in [Bar84] to identify terms as equal implicitly up to a change of bound variable. Formally, this notion is called **α-equivalence**. Specifically, if $\lambda x.P$ is a part of a term M, then, if you obtain a term N by changing this part to $\lambda y.P'$, where P' comes from replacing all the free instances of x in P with a fresh (that is, completely unused in P) variable y, we say that M and N are α-equivalent. Taking the transitive reflexive closure of terms under this relation, we get the full notion of α-equivalence relation.

We consider α-equivalent terms to be equal on a syntactic level, so when we are considering a term we are actually working with an equivalence class or terms. For example, we would say $\lambda x.x \equiv \lambda y.y$. Because of this association, we can allow ourselves to assume that all bound variables have different names than free variables, each bound variable in appearing in a term has a unique binding site with the same name.

Remark 2 (Notational conventions). To improve legibility, we will follow typical conventions with regard to associativity; namely we will always associate applications to the left. Therefore, MNP will be written to mean $(MN)P$.

We use the dot . after a binder in the fashion of Church's dot notation to remind the reader that the scope the variable is bound in is to be as large as possible. Therefore, $\lambda x.MN$ will mean $\lambda x.(MN)$.

We also will write successive abstractions into one abstraction; so we will write $\lambda xy.M$ instead of $\lambda x.\lambda y.M$.

For term variables, we will use lowercase Latin letters toward the end of the alphabet. For arbitrary terms we will choose capital Latin letters toward the middle of the alphabet.

Definition 2. We define the set of (one-hole) contexts, where a **context** can be written as $C[\ ]$, inductively in the same way as a term except we also allow a hole $[\ ]$ in the same place we allowed a variable in the definition of the set Λ. If $C[\ ]$ is a context, then $C\,[M]$ is the term that is obtained by changing the hole to M.

To carefully define the semantics of function application, we first need a formal notion of what a variable substitution is. This notion will be reused for polymorphic types in Chapter 4.

Definition 3. If M and N are terms then we write $M[x := N]$ to stand for the **capture-avoiding substitution** of free instances of x for N in M. Formally, this is an inductive definition:

- if $M \equiv x$, then $M[x := N] \equiv N$;
- if $M \equiv y$ (where $y \not\equiv x$), then $M[x := N] \equiv y$;
- if $M \equiv PQ$, $M[x := N] \equiv (P[x := N])(Q[x := N])$; and
- if $M \equiv \lambda y.P$, $M[x := N] \equiv \lambda y.P[x := N]$.

Definition 4. We call a term with no free variables **closed**, or a **combinator**. We denote the set of such terms $\Lambda^{\emptyset}$.

The operational semantics of the lambda calculus will be modeled around the following notion: $\lambda x.P$ is a function awaiting input which can be substituted for x in P . On a syntactic level, this semantics is realized by the notion of β**-reduction**, which we will denote $\to_\beta$. This relation on terms is defined as follows: if $C[\ ]$ is any one-hole context then

$$C\,[(\lambda x.M)N] \to_\beta C\,[M[x := N]]$$

We write $\twoheadrightarrow_\beta$ to stand for the transitive, reflexive closure of $\to_\beta$, and we write $=_\beta$ for the symmetric closure of this. Indeed, $=_\beta$ is an equivalence relation called β**-equality**, or β**-convertibility**.

We will make use of another notion of reduction called η**-reduction**, which we write $\to_\eta$ and whose transitive, reflexive closure we write $\twoheadrightarrow_\eta$. The definition of this form of reduction is

$$C\,[\lambda x.Mx] \to_\eta C\,[M] \qquad x \notin \mathrm{FV}(M)$$

where $C[\ ]$ is any one-hole context. η-reduction gives us a weak notion of extensionality. When we consider a notion of reduction where at any step we can perform either an η reduction or a β reduction we will write $\to_{\beta\eta}$, $\twoheadrightarrow_{\beta\eta}$ for the transitive, reflexive closure of $\to_{\beta\eta}$, and $=_{\beta\eta}$ for the symmetric closure of $\twoheadrightarrow_{\beta\eta}$.

Definition 5. We call the subterms which can be reduced **redexes**, short for reducible expressions. In the event a term contains no subterms which are redexes we call it a β-**normal form**, or $\beta\eta$-normal form, depending on the notion of reduction we are considering. If the M is a term and $M \twoheadrightarrow_\beta N$ for some term N in normal form, we call the term M **normalizing**. In the event that there is no infinite reduction sequence such as

$$M \to_\beta M_1 \to_\beta M_2 \to_\beta \cdots$$

then we call the term M **strongly normalizing**.

It is an undecidable problem to determine for an arbitrary term M if it is normalizing (or strongly normalizing) and also undecidable to determine if two terms are convertible [Chu36b].

A classical result of the lambda calculus is the **Church–Rosser theorem**, which is a result about $\beta\eta$-convertibility, and was proved by Church and Rosser in [CR36]. It states if that $M =_{\beta\eta} N$ then there is a term P such that $M \twoheadrightarrow_{\beta\eta} P$ and $N \twoheadrightarrow_{\beta\eta} P$.

A similar property is the **confluence** of $\beta\eta$-reduction (also known as the **diamond property**) which states that if $M \twoheadrightarrow_{\beta\eta} N$ and $M \twoheadrightarrow_{\beta\eta} P$ then there is a term Q such that $N \twoheadrightarrow_{\beta\eta} Q$ and $P \twoheadrightarrow_{\beta\eta} Q$. This is show in Figure 1.1.

Because of the above properties, if a term normalizes, it as has a unique normal form. Therefore, we can say *the* normal form of a term rather than *a* normal form. Another consequence is that lambda calculus is consistent, in that it is not true that for terms M and N we have $M =_{\beta\eta} N$. In particular, all distinct normal forms are not convertible to each other.

Some combinators

Some combinators are particularly important for encoding some data structures and make some appearances in this paper.

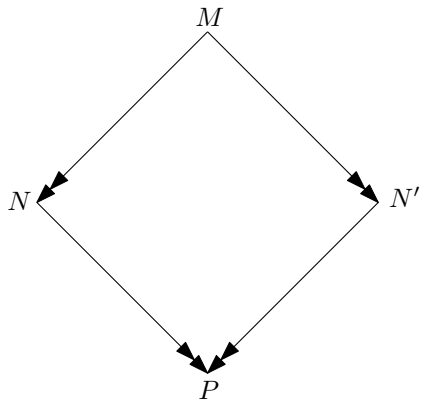

Figure 1.1: The diamond property

Definition 6.

$$\begin{aligned}\mathsf{I} &:= \lambda x.x\\ \mathsf{S} &:= \lambda xyz.xz(yz)\\ \mathsf{K} &:= \lambda xy.x\\ \omega &:= \lambda x.xx\end{aligned}$$

Definition 7. If $X \subseteq \Lambda$, then the **applicative closure** of X is the smallest set containing X which is closed under applications; that is, if M and N are in the applicative closure of X then MN is in the applicative closure of X.

We say that a set of combinators X is a **combinatorially complete** for the lambda calculus for every term M with free variables from $x_1, \ldots, x_n$ there is a combinator F from the applicative closure of X where we have that $Fx_1 \ldots x_n =_\beta M$

It is known that $\{\mathsf{S}, \mathsf{K}\}$ are combinatorially complete [Cur41].

Definition 8 (Church booleans)**.** Consider the combinators:

$$\mathsf{True} := \lambda xy.x \qquad \mathsf{False} := \lambda xy.y$$

We use these to encode boolean values. We can then define particular combinators to do all boolean logic.

$$\begin{aligned}\mathsf{And} &:= \lambda xy.xy\mathsf{False}\\ \mathsf{Or} &:= \lambda xy.x\mathsf{True}y\\ \mathsf{Not} &:= \lambda x.x\mathsf{False}\mathsf{True}\end{aligned}$$

Definition 9 (Church numerals [Chu32])**.** The Church numerals are an encoding of the natural numbers. The Church numeral for n, which we will write n is a function which iterates its first argument on the second n times; that is

$$\mathsf{n} = \lambda f x \mathbf{.} \underbrace{f(f(\ldots f(fx)\ldots))}_{n\text{-many}}$$

So $\mathsf{0} \equiv \lambda f x \mathbf{.} x$, $\mathsf{1} \equiv \lambda f x \mathbf{.} fx$, $\mathsf{2} = \lambda f x \mathbf{.} f(fx)$, etc.

In [Kle35], Kleene showed the definability of arithmetic operators in the lambda calculus using the Church numeral encoding, and proved their correctness. It is a classical theorem of Turing that the arithmetic functions computed in the lambda calculus in this sense are the same as those computed using Turing machines [Tur37].

Remark 3. The original system studied primarily by Church was called $\lambda\mathsf{I}$-calculus. This contrasts with the system of λ-calculus most encountered today, called the $\lambda\mathsf{K}$-calculus. In the $\lambda\mathsf{I}$-calculus, all bound variables must appear at least once in the term they abstract.

For example, the term K would not be a term in the $\lambda\mathsf{I}$-calculus since the y in $\lambda xy\mathbf{.}x$ does not appear; neither would the numeral $\mathsf{0}$. Historically, Church did sometimes consider $\lambda\mathsf{K}$-terms however, as he did in [Chu40].

The $\lambda\mathsf{I}$-calculus has some properties which the $\lambda\mathsf{K}$-calculus does not. One such property is that a term is strongly normalizing if and only if it is normalizing [Chu41]. This is not a property of the $\lambda\mathsf{K}$-calculus because of terms like $\mathsf{KI}(\omega\omega)$, where one could infinitely reduce $\omega\omega$, but a reduction could be done to arrive at a normal term:

$$\mathsf{KI}(\omega\omega) \to_\beta (\lambda y\mathbf{.}\mathsf{I})(\omega\omega) \to_\beta \mathsf{I}.$$

1.0.2 The simply typed lambda calculus

We wish to restrict the notion of function application to reflect the idea of a domain and codomain of a function. For this we will associate each term with a type which describes its behavior.

Define the set of **simple types** inductively as follows: begin with some nonempty set of type variables $a, b, c, a_1, a_2, a_3, \ldots$ which we call **atomic types** or **ground types**. Often one has fixed, finite set of theses atomic types which will correspond to different kinds of objects (for example, propositions, numbers). If α and β are types then $\alpha \to \beta$ is a type. We call such types **arrow types**, and they're meant to represent functions from things of type α to things of type β.

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ (\text{ax}) \qquad \frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}\ (\text{app})$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x . M : \alpha \to \beta}\ (\text{abs})$$

Figure 1.2: Deduction rules for the simply typed lambda calculus (Curry-style typing)

Remark 4 (Notational conventions). In contrast to application of terms, we associate $\to$ to the right. That is, $\alpha \to \beta \to \gamma$ is parsed as $\alpha \to (\beta \to \gamma)$.

For atomic types, we will use lowercase Latin letters toward the beginning of the alphabet. For arbitrary types we will use lowercase Greek letters toward the beginning of the alphabet.

There are two main ways to approach the assignment of types to terms, called **Curry-style typing** and **Church-style typing**. In Church-style typing, term variables (both free and bound) are all decorated with types which control the types of the subterm that compose. The other approach, which we use mostly in this thesis, is Curry-style typing, also called **implicit typing**. Here, the terms of the simply typed lambda calculus are all terms of the untyped lambda calculus, and the typing information is completely external from the term.

A proper Curry-style typing of a term is a derivation in the style of natural deductions. A **context** is a partial function Γ with finite support from the set of term variables to the set of types. We think of (and write) a context as a finite list

$$x_1 : \alpha_1, \ldots, x_n : \alpha_n$$

where each x_i is distinct. The derivation rules for deducing $M : \alpha$, read as M has type α, are shown in Figure 1.2.

Definition 10. We denote the collection of untyped lambda terms for which have a type $\Lambda^{\to}$. If $M \in \Lambda^{\to}$ then we say that M is **typable** (in the simply typed lambda calculus). Otherwise, if $M \in \Lambda \setminus \Lambda^{\to}$ we say that M is untypable.

These rules are **syntax directed**, meaning given typable term M there is one and only one rule that could have been applied last in every derivation

of a type for M. So, for instance, if M is the term $\lambda x.x$ then the last rule must have been an instance of the the abs rule.

Remark 5. There is a correspondence between the above type system and intuitionist implicational logic. If all the terms are erased so only the types remain then the derivations are exactly natural deduction proofs in this system. Moreover, the terms act as a 'certificate' that describes the proof. The relationship is an instance of the **Curry–Howard isomorphism**, which relates systems of typed lambda calculus and logical systems [How80].

Definition 11. We say that a Curry-style system of typed lambda calculus has the **subject reduction** property, also called **preservation**, if for any $M : \alpha$, if $M \twoheadrightarrow_\beta N$ then $N : \alpha$.

We say it has **subject expansion** if for any $M : \alpha$, if $N \twoheadrightarrow_\beta M$ then $N : \alpha$.

This simply typed lambda calculus has the subject reduction property, due to Curry [Cur34, CFC74]. It does not have the subject expansion property; for example, $\mathsf{KI}\omega \to_\beta I$, but the $\mathsf{KI}\omega$ is itself untypeable.

To see why ω does not have a simple type, suppose it did and consider the type of x in xx. This type must have some type $\alpha \to \beta$ since x appears in an application in the functional position. But, then x must have type α, but α and $\alpha \to \beta$ are distinct.

It is also the case that any term M which is typable has a normal form, as proved first by Turing in a note published in [Gan80]. In fact, every term is strongly normalizing, with the classical proof due to Tait in [Tai67].

In the light of the Curry–Howard isomorphism, subject reduction and normalization give us operational semantics on proofs (at least in intuitionist implicational logic), and a notion of a normal proof. These notions have very deep connections to proof theory, where normal proofs yield cut-free proofs. Therefore, the normalization result is actually a different form of the classical cut-elimination theorem of Gentzen [Gen35].

Definition 12. The **typability** problem is: given a term M in the untyped lambda calculus to determine if it is typable or not.

The **inhabitation** problem is: given a type α, is there a term M such that $M : \alpha$.

A **type substitution** is a map $\star$ which maps type variables to types which has finite support (that is, for all but finitely many types the map is the identity).

α is the **most general type** or **principal type** for a term M if $M : \alpha$ and for every β such that $M : \beta$ there is a type substitution which sends α to β.

Under the Curry–Howard isomorphism, this problem of inhabitation is the same as provability in intuitionist implicational logic, which is decidable.

Typability is also decidable; the algorithm that gives a term a type relies on an algorithm by Robinson [Rob65] solving the first-order unification problem. Specifically, the Hindley–Milner algorithm can decide if a term is typable, and if it is produce the most general type [Mil78, Hin69].

Remark 6. In Church-style typing, since it contains type information, every term M has a unique type. In Curry-style typing, this is of course not true. For example, $\mathsf{I} \equiv \lambda x\mathbf{.}x$ can be assigned both the type $a \to a$ and $(a \to a) \to (a \to a)$. We recover some form of this uniquicity in Curry-style typing for simple types in the form of principal types.

There is a clear connection between Church-style and Curry-style typings. One can map a Church-style term M having type α into a term of the untyped lambda calculus N by erasing all type decorations, as described in [BDS13]. Then, it is provable that $N : \alpha$ in the Curry-style system.

Definition 13. We consider another normal form of a term called the **long normal form** in the context of the typed lambda calculus with η-reduction. We define what it means for $M : \alpha$ to be in long normal form by induction on α.

If α is an atomic type, then M is is long normal form if and only if it is of the form $xM_1 \dots M_m$ where each M_i is in long normal form.

If $\alpha = \beta \to \gamma$, then M is in long normal form if and only if it is of the form $\lambda f\mathbf{.}N$ where N is in long normal form.

Each simply typed term has a unique long normal form which one can obtain by β-reductions and η-expansions.

Semantics

To perform a set theoretic interpretation of typed lambda calculus terms, we first need the idea of a type structure. A **type structure** is a family of nonempty sets $\mathscr{M}(\alpha)$ where α is a type, such that $\mathscr{M}(\alpha \to \beta) \subseteq \mathscr{M}(\beta)^{\mathscr{M}(\alpha)}$, where $\mathscr{M}(\beta)^{\mathscr{M}(\alpha)}$ is the set of all functions from $\mathscr{M}(\alpha)$ to $\mathscr{M}(\beta)$.. If the above relation is taken as equality instead of subset we call it the **full type structure**.

More abstractly, we can have a family of sets $\mathscr{M} = \{\mathscr{M}(\alpha) \mid \alpha \text{ a type}\}$ for each type α and a operation $\cdot_{\alpha\to\beta}$ as

$$\cdot_{\alpha\to\beta} : \mathscr{M}(\alpha \to \beta) \times \mathscr{M}(\alpha) \to \mathscr{M}(\beta),$$

which replaces the normal semantics of function application above. That is, in the case of a type structure, we have that members of $\mathscr{M}(\alpha \to \beta)$ act on members of $\mathscr{M}(\alpha)$ by function application. In this scenario, $\cdot_{\alpha\to\beta}$ describes the action of members of $\mathscr{M}(\alpha \to \beta)$ on $\mathscr{M}(\alpha)$.

We say that $\cdot_{\alpha\to\beta}$ is extensional if for any $f, g \in \mathscr{M}(\alpha \to \beta)$ then if for every $m \in \mathscr{M}(\alpha)$, $f \cdot m = g \cdot m$. In the event that this operation is extensional, we call this a **typed applicative structure**. A type structure is an example of an applicative structure where $\cdot$ is taken to be function application.

Often we conflate the family of sets $\mathscr{M}$ with $\bigcup_\alpha \mathscr{M}(\alpha)$ by saying $f \in \mathscr{M}$ to mean $f \in \mathscr{M}(\alpha)$ for some α.

Definition 14 (Friedman [Fri75])**.** A **partial homomorphism** is a partial function ξ between type structures $\mathscr{M}$ and $\mathscr{N}$ over the same ground types which has the following properties:

1. If $f \in \mathscr{M}$ and $\xi(f) \in \mathscr{N}(\alpha)$ then $f \in \mathscr{M}(\alpha)$.
2. If $f \in \mathscr{M}(\alpha \to \beta)$ then $\xi(f) = g$ if and only if g is the unique element of $\mathscr{N}(\alpha \to \beta)$ such that for all $x \in \mathrm{dom}(\xi) \cap \mathscr{M}(\alpha)$, we have $g(\xi(x)) = \xi(f(x))$

Note, the uniqueness may not hold, making this map partial. In the situations we will encounter, uniqueness will hold, and the map is total, and we call the map a **homomorphism**. The essential property of a homomorphisms the above captures in the more general situation is that

$$\xi(f(x)) = \xi(f)(\xi(x)).$$

Also note that a homomorphism is completely determined by the map restricted to the ground type.

An **environment** is a function ϕ which maps variables of type α to members of $\mathscr{M}(\alpha)$. An **interpretation** of terms subject to environment ϕ is in a type structure is a function $[\![\cdot]\!]_\phi$ which maps typed terms to members of $\mathscr{M}$ of appropriate type, such that

- $[\![x]\!]_\phi = \phi(x)$ for any variables,

- $[\![MN]\!]_\phi = [\![M]\!]_\phi\,([\![N]\!]_\phi)$,
- $[\![\lambda x\,.\,M]\!]_\phi = \lambda\!\!\lambda z\,.\,[\![M]\!]_{\phi[x:=z]}$

where $\phi[x := z]$ is the environment ϕ except x maps to z, and $\lambda\!\!\lambda z.M$ is function in our meta-language (commonly written as the function $z \mapsto M$). Note that this function is unique (given ϕ), but if the type structure is not full it is possible that $[\![\lambda x\,.\,M]\!]_\phi \notin \mathscr{M}(\alpha \to \beta)$.

This interpretation respects the semantics of $\beta\eta$-reduction. Namely, if M and N are terms of the untyped lambda calculus where M, $N : \alpha$ and $M =_{\beta\eta} N$ then $[\![M]\!]_\phi = [\![N]\!]_\phi$.

Chapter 2

Classical Definability

2.1 $\lambda\delta$-calculus and type theory

We extend the simply typed lambda calculus over one ground type 0 with a constant δ. This symbol will represent an equality operator. Adding such an equality operator over all types would elicit the study of higher order logic, which we discuss in Chapter 3. For our purposes, we are dealing with first-order classical logic, and our equality symbol will be just for the ground type 0.

To encode booleans in our logic, we will use the usual Church encoding discussed in Definition 8. Notice that the types of booleans over a type α is $\alpha \to \alpha \to \alpha$. We will call this type bool_α. For the remainder of this chapter, we will use the notation $\alpha^n \to \beta$ to stand for the type

$$\underbrace{\alpha \to \cdots \to \alpha}_{n-\text{many}} \to \beta.$$

That is, the type which takes is n-many terms of type α and returns a term of type β.

Definition 15. We add our constant $\delta : 0 \to 0 \to \mathsf{bool}_0$ to our language, as studied by Church in [Chu41]. We define δ-equivalence using the axiom

$$\delta xyuv = \begin{cases} u & \text{if } x = y \\ v & \text{if } x \neq y. \end{cases}$$

In [Sta00] it was proven that under $\beta\eta$-conversion, the equational consequences of this axiom are exactly the same as from these rules:

$$\begin{aligned}
\delta MMUV &= U && \text{(Reflexivity)}\\
\delta MNUU &= U && \text{(Identity)}\\
\delta XYUV &= \delta YXUV && \text{(Symmetry)}\\
\delta XYXY &= Y && \text{(Hypothesis)}\\
P(\delta MN) &= \delta MN(P\mathsf{True})(P\mathsf{False}) && \text{(Monotonicity)}\\
\delta MN(\delta MNUV)W &= \delta MNUW && \text{(Stutter)}\\
\delta MNU(\delta MNWV) &= \delta.MNUV && \text{(Stammer)}
\end{aligned}$$

Definition 16. In addition to δ, in order to do first-order logic, we add two other constants: a quantifier $\exists : (0 \to \mathsf{bool}_0) \to \mathsf{bool}_0$ and a description operator $\iota : (0 \to \mathsf{bool}_0) \to 0 \to 0$ defined by the rules,

$$\exists M = \begin{cases} \mathsf{True} & \text{if } Mn = \mathsf{True} \text{ for some } n : 0 \\ \mathsf{False} & \text{otherwise,} \end{cases}$$

and

$$\iota Mm = \begin{cases} n & \text{if } Mn = \mathsf{True} \text{ and } n \text{ is unique such} \\ m & \text{otherwise.} \end{cases}$$

We will define a family of sets $\mathscr{M}^n(\alpha)$ to be the full type structure over $[n]$; that is, where we have

$$\mathscr{M}^n(0) = \{1, \ldots, n\}.$$

For the interpretation of a term in this model with environment ϕ we write $[\![M]\!]^n_\phi$. We will write set-theoretic functions as lowercase Latin like f, g, h.

We have the following:

Theorem 1. *Let M and N be terms of type α.*

1. *(Soundness) If $M =_{\beta\eta\delta} N$ then for every $n \in \mathbb{N}$ and every ϕ we have $[\![M]\!]^n_\phi = [\![N]\!]^n_\phi$.*
2. *(Completeness) If $M \neq_{\beta\eta\delta} N$ then there is an $n \in \mathbb{N}$ and a ϕ such that $[\![M]\!]^n_\phi \neq [\![N]\!]^n_\phi$.*

Proof. Proof in Statman [Sta00] □

Definition 17. We say that a function $f \in \mathscr{M}$ is $\lambda\delta$-definable if there is a closed term M in the $\lambda\delta$-calculus where $[\![M]\!]^n_\phi$ is f.

We will also analyze define $\lambda\delta$-definablity in the system discussed about with $\exists$ and ι.

2.2 Henkin's Theorem

We can say that, in some way, every function in the above semantics can be represented in the $\lambda\delta$-calculus.

Theorem 2 ([Hen63])**.** *Fix an environment ϕ, a natural number n, and a type α. Then*

1. *There is a term $\delta_\alpha : 0^n \to \alpha \to \alpha \to \mathsf{bool}_\alpha$ such that for every $f, g, h, j \in \mathscr{M}^n(\alpha)$ we have*

$$([\![\delta_\alpha]\!]^n_\phi 1 \cdots n) f g h j = \begin{cases} h & \text{if } f = g \\ j & \text{otherwise.} \end{cases}$$

2. *If $f \in \mathscr{M}^n(\alpha)$ then there is a term $F : 0^n \to \alpha$ such that:*

$$[\![F]\!]^n_\phi 1 \cdots n = f$$

Proof. We do induction on the type α.

If $\alpha = 0$, then define

$$\delta_0 = \lambda x_1 \ldots x_n \bullet \delta.$$

If $f \in \mathscr{M}^n(0)$ then $f \in [n]$, so $f = i$ where $1 \leq i \leq n$. Then we can just make F be the ith projection:

$$\lambda x_1 \ldots x_n \bullet x_i.$$

Suppose that $\alpha = \beta \to \gamma$. By induction, We have closed terms δ_β and δ_γ with the desired properties. Enumerate all elements of $\mathscr{M}^n(\beta)$: $m_1, \ldots, m_k$. By the induction hypothesis, we have representations $M_1, \ldots, M_k$, closed terms of type $0^n \to \beta$ such that $[\![M_i]\!]^n_\phi 1 \ldots n = m_i$ for every i. So we define

$$\begin{aligned} \delta_{\beta\to\gamma} = \lambda \bar{x} F G \bullet \\ \delta_\gamma \bar{x}(F(M_1\bar{x}))(G(M_1\bar{x})) \wedge \delta_\gamma \bar{x}(F(M_2\bar{x}))(G(M_2\bar{x})) \\ \wedge \cdots \wedge \delta_\gamma \bar{x}(F(M_k\bar{x}))(G(M_k\bar{x})), \end{aligned}$$

where $\bar{x}$ is shorthand for $x_1 \ldots x_n$ and $M \wedge N$ is $\mathsf{And}MN$ (from Definition 6). This is as desired.

Let f be a function in $\mathscr{M}^n(\beta \to \gamma)$. Note that $f(m_i) \in \mathscr{M}^n(\gamma)$ by definition. Therefore, set $p_i = f(m_i)$. By induction hypothesis, there are

$P_1, \ldots, P_k$ such that $[\![P_i]\!]^n_\phi 1 \ldots n = p_i$ for every $1 \leq i \leq k$. We define F by doing cases on the input. That is

$$\begin{aligned} F = \lambda \bar{x} m \bullet \\ \text{If } \delta_\beta \bar{x} m (M_1 \bar{x}) \text{ then } P_1 \bar{x} \text{ else} \\ \text{If } \delta_\beta \bar{x} m (M_2 \bar{x}) \text{ then } P_2 \bar{x} \text{ else} \\ \ldots \\ \text{If } \delta_\beta \bar{x} m (M_{k-1} \bar{x}) \text{ then } P_{k-1} \bar{x} \text{ else } P_k \bar{x}, \end{aligned}$$

where "If M then N else P" is shorthand for MNP. This is as claimed. □

The following is an easy corollary to Theorem 2 and to the completeness result in Theorem 1.

Corollary 1. *Take $M, N : \alpha_1 \to \alpha_2 \to \cdots \to \alpha_k \to 0$ with all free variables having type 0. If $M \neq_{\beta\eta\delta} N$ then there some n and closed terms $F_i : 0^n \to \alpha_i$ such that*

$$M(F_1 \bar{x}) \cdots (F_k \bar{x}) \neq_{\beta\eta\delta} N(F_1 \bar{x}) \cdots (F_k \bar{x})$$

where $\bar{x}$ includes all variables free in both M and N.

Proof. By soundness, for some ϕ and n we have $[\![M]\!]^n_\phi \neq [\![N]\!]^n_\phi$. Take $\bar{x}$ to be a sequence of length n of free variables of type 0, containing all the free variables of M and N. Clearly $[\![\lambda \bar{x} \bullet M]\!]^n_\phi \neq [\![\lambda \bar{x} . N]\!]^n_\phi \bullet$

These are set-theoretic functions, therefore there are $f_1 \in \mathscr{M}^n(\alpha_1)$, ..., $f_k \in \mathscr{M}^n(\alpha_k)$ such that

$$([\![\lambda \bar{x} \bullet M]\!]^n_\phi 1 \ldots n) f_1 \ldots f_k \neq ([\![\lambda \bar{x} \bullet N]\!]^n_\phi 1 \ldots n) f_1 \ldots f_k.$$

By Henkin, all these f_i have closed terms of type $0^n \to \alpha_i$ representing them; denote those closed terms F_i. By soundness, the result follows. □

Consider the set of terms of type α in the language of the $\lambda\delta$-calculus, which we denote $\Lambda^\delta(\alpha)$. Define the set $\mathscr{T}(\alpha)$ by

$$\mathscr{T}(\alpha) := \Lambda^\delta(\alpha) / =_{\beta\eta\delta}.$$

That is, $\mathscr{T}(\alpha)$ is the set of terms of type α modulo $\beta\eta\delta$-equivalence. As a consequence to Corollary 1 above we have that $\mathscr{T}$ is a typed applicative structure. For each natural number n we consider a set of n free variables, $X = \{x_1, \ldots, x_n\}$ of type 0. We can take the set of all terms M in $\mathscr{T}$ which are $\lambda\delta$-definable from this set.

This is not necessarily a typed applicative structure. For we may have two terms M_1 and M_2 which are not extensionally equal, but are with respect to all terms λ-definable from $X \cup \{\delta\}$. That is, none of the witnesses that M_1 and M_2 are different are λ-definable from $X \cup \{\delta\}$. Therefore, we consider only the equivalence classes formed by equality under δ restricted only to the ground set X. So, if we have a M_1 and M_2 as above, we collapse them. We call the resulting model the **Gandy Hull** of $X \cup \{\delta\}$ in $\mathscr{T}$. This is a typed applicative structure, which we will denote by $\mathscr{T}^n$. For more information on the Gandy Hull construction, see [BDS13].

Remark 7. There is a natural homomorphism (see Definition 14) between $\mathscr{T}^n$ and $\mathscr{M}^n$ which is completely determined by a mapping of X to $[n]$. Further, we can consider some infinite models. For instance, we can define $\mathscr{M}^\omega$ to be the full type structure in this language over the natural numbers; so $\mathscr{M}^\omega(0) = \mathbb{N}$. We can then take the Gandy Hull of $\{1, 2, 3, \ldots\} \cup \delta$ in this model and get a model $\mathscr{M}$.

This model could be obtained another way. Fix a bijection from free variables of type 0 and ω. Then one can build a corresponding homomorphism from $\mathscr{T}$ to $\mathscr{M}^\omega$. The image is exactly $\mathscr{M}$. These models are discussed further in Statman [Sta82].

2.3 Symmetric iff λ-definable with δ, ∃, and ι

In this section, we will provide a proof of a folklore theorem giving necessarily and sufficient conditions for a function being $\lambda\delta$-definable with ι and $\exists$. The origin of this folklore theorem is murky; it was known to Robin Gandy in at least the 1940s, and is not unlikely that it was known to Church before that. There are few proofs in print, one being in [vBD01], but it is incomplete. Here we present a novel proof only using some basic algebra and Theorem 2.

Definition 18. The **symmetric group on n elements**, which we denote as S_n, is the subset of $\mathscr{M}^n(0 \to 0)$ which are bijections. These form a group with the operation of composition. We call members of the group permutations. We shall use lower case Greek letters in the middle of the alphabet to denote permutations, like π, ρ, σ, τ.

Members of S_n act naturally on objects of type 0 by application. But, we can lift this action to higher types. Consider $\pi \in S_n$. We define $\pi_\alpha \in \mathscr{M}^n(\alpha \to \alpha)$ by induction on α. If $\alpha = 0$, then we just take $\pi_0(n) := \pi(n)$. If $\alpha = \beta \to \gamma$ then we define

$$\pi_\alpha(f) := \pi_\gamma \circ f \circ \pi_\beta^{-1}$$

Therefore, we have an action of S_n on our entire model $\mathscr{M}^n$, where π acts on $f : \alpha$ by $\pi_\alpha(f)$. For this action, we will write $\pi \cdot f$.

Definition 19. If $f \in \mathscr{M}^n$, then we denote the stabilizer of f under this action $\mathrm{St}(f)$; that is, $\mathrm{St}(f)$ is the set of all permutation which fix f, as in

$$\mathrm{St}(f) := \{\pi \in S_n \mid \pi \cdot f = f\}.$$

We call an $f \in \mathscr{M}^n$ **symmetric** if $\mathrm{St}(f) = S_n$, that is, if f is fixed under the action by all permutations.

Remark 8. We can say that S_n acts on $\mathscr{T}^n$ as well. Any permutation of the free variables elicits an automorphism on the entire set $\mathscr{T}^n$ (that is, a bijective homomorphism from $\mathscr{T}^n$ to $\mathscr{T}^n$). The converse, however, is false; there are automorphisms of $\mathscr{T}^n$ that do not come from permutations of the variables.

For example, consider the automorphism f on $\mathscr{T}^4$ elicited by the following map on variables:

$$\begin{aligned} f(x_1) &= x_1 \\ f(x_2) &= x_2 \\ f(x_3) &= \delta x_1 x_2 x_3 x_4 \\ f(x_4) &= \delta x_1 x_2 x_4 x_3. \end{aligned}$$

This is a well-defined automorphism on $\mathscr{T}^4$.

Later, we will consider particular members of T^n to be symmetric. When we call $F \in \mathscr{T}^n$ symmetric, we mean preserved under all automorphisms, not just the "inner" automorphisms arising from permutations of variables.

Interestingly, the set of automorphisms is not the set of all permutations on all distinct objects of type 0. In particular, there is no automorphism of $\mathscr{T}^4$ which sends

$$\begin{aligned} x_1 &\mapsto x_1, \\ x_2 &\mapsto x_2, \\ x_3 &\mapsto x_3, \\ x_4 &\mapsto \delta x_1 x_2 x_3 x_4. \end{aligned}$$

This is because such an automorphism h would have

$$\begin{aligned} h(\delta x_1 x_2 x_3 x_4) &= \delta(h(x_1))(h(x_2))(h(x_3))(h(x_4)) \\ &= \delta x_1 x_2 x_3 (\delta x_1 x_2 x_3 x_4) \\ &= \delta x_1 x_2 x_3 x_4, \end{aligned}$$

where the last equality is by the Stammer property; so h would not be a bijection.

Theorem 3 (Folklore Thoerem)**.** *$f \in \mathscr{M}^n$ is symmetric if and only if it is λ-definable from $\delta, \iota, \exists$*

Proof. The right-to-left direction is straightforward. For δ, ι, and $\exists$ are all symmetric, as are combinators S and K. As S and K form a basis for all λ-terms, and λ-definable objects are closed under application, we have that all λ-definable objects are indeed symmetric. The left-to-right direction will constitute the majority of the remainder of this section.

At a high level, what we will do in this direction of the proof is begin by study a particular class of functions at low type $(0^n \to 0)$ which we call regular functions. We'll show that regular functions are easily definable (in fact, just in the $\lambda\delta$-calculus) just using some algebraic properties of the action of the symmetric group on these functions. Then, we'll show that an arbitrary symmetric function can be represented in terms of a set of functions we called coordinate functions, which are all regular functions of low type, $\exists$, and ι; this will complete the proof. To begin, however, we come up with some notation so we can more easily analyze some of the algebraic properties of these functions.

For each function $f : 0^n \to \alpha$ we associate a functional $f^+ : (0 \to 0) \to \alpha$ such that, for all $\pi : 0 \to 0$,

$$f^+\pi = f(\pi 1)(\pi 2)\dots(\pi n).$$

A function f is said to be **regular** if for every $g \in \mathscr{M}^n(0 \to 0)$ where $g \notin S_n$ we have $f^+g = g(1)$.

For the present we will restrict our attention only on functions $f : 0^n \to 0$. Note that the action $\pi \cdot f$ in this case is

$$\pi \cdot f = \lambda\!\!\lambda \bar{x}_\bullet \pi\big(f(\pi^{-1}x_1)\dots(\pi^{-1}x_n)\big).$$

From this and the fact that $\mathrm{St}(f)$ is a subgroup, we have that $\pi \in \mathrm{St}(f)$ if and only if

$$\lambda\!\!\lambda \bar{x}_\bullet \pi^{-1}\big(f(\pi x_1)\dots(\pi x_n)\big) = f. \tag{$\dagger$}$$

Fix $f : 0^n \to 0$ regular. We define a relation $\sim_f$ on S_n by

$$\pi \sim_f \sigma \iff \pi^{-1}\big(f(\pi 1)\dots(\pi n)\big) = \sigma^{-1}\big(f(\sigma 1)\dots(\sigma n)\big).$$

We restrict this this relation to be a right congruence by taking its **right congruence hull**, which we denote $\sim_f^*$, defined by

$$\pi \sim_f^* \sigma \iff \forall \rho \in S_n{}_\bullet \pi\rho^{-1} \sim_f \sigma\rho^{-1}.$$

Lemma 1. *For $f : 0^n \to 0$ regular, and $\pi \in S_n$, the following are equivalent:*

1. *$\pi \in St(f)$,*
2. *For all $\rho \in S_n$ we have $\pi\rho \sim_f \rho$, and*
3. *$\pi \sim_f^* id$.*

Proof. ((1) $\implies$ (2)). Take $\pi \in \mathrm{St}(f)$. By (†), we have

$$\lambda\!\!\lambda \bar{x}.\pi^{-1}\big(f(\pi x_1)\dots(\pi x_n)\big) = f.$$

Fix $\rho \in S_n$ and apply $\rho 1, \rho 2, \dots, \rho n$ to the right, giving us

$$\pi^{-1}\Big(f\big(\pi(\rho 1)\big)\dots\big(\pi(\rho n)\big)\Big) = f(\rho 1)\dots(\rho n).$$

Then, applying ρ^{-1} to the left, gives us

$$\rho^{-1}\Big(\pi^{-1}\Big(f\big(\pi(\rho 1)\big)\dots\big(\pi(\rho n)\big)\Big)\Big) = \rho^{-1}\big(f(\rho 1)\dots(\rho n)\big),$$

which implies that $\pi\rho \sim_f \rho$.

((2) $\implies$ (3)). Take $\rho \in S_n$. We want to show that $\pi\rho^{-1} \sim_f \mathrm{id}\rho^{-1}$. The right hand side is of course just ρ^{-1}, therefore this follows immediately from (2).

((3) $\implies$ (1)). By (†), it suffices to show that

$$\lambda\!\!\lambda \bar{x}.\pi^{-1}\Big(f\big(\pi x_1\big)\dots\big(\pi x_n\big)\Big) = f.$$

By extensionality, it suffices to show the above holds after an arbitrary application. Moreover, let us fix an arbitrary $g : 0 \to 0$ (not necessarily in S_n). The application of $g(1)$ to the right of both sides, followed by $g(2)$, etc, up to $g(n)$ is an arbitrary application as g is arbitrary, thus it suffices to show that

$$\pi^{-1}\Big(f\big(\pi(g1)\big)\dots\big(\pi(gn)\big)\Big) = f(g1)\dots(gn).$$

If $g \notin S_n$, then by regularly of f, both sides are exactly $g(1)$. Otherwise, set $\rho := g$, which is a member of S_n. By (3) (using the right congruence property on ρ^{-1}), we have that $\pi\rho \sim_f \rho$. This means that

$$\rho^{-1}\Big(\pi^{-1}\Big(f\big(\pi(\rho 1)\big)\dots\big(\pi(\rho n)\big)\Big)\Big) = \rho^{-1}\big(f(\rho 1)\dots(\rho n)\big)$$

which is exactly what we wanted. □

Let $\mathscr{B}$ be the set of equivalence classes of $\sim_f^*$. For each $B \in \mathscr{B}$, let $\chi_B : 0^n \to \mathsf{bool}_0$ denote its characteristic function; that is,

$$\chi_B^+(\pi) := \begin{cases} \mathsf{True} & \text{if } \pi \in B \\ \mathsf{False} & \text{otherwise.} \end{cases}$$

By definition of the equivalence relation, if π and σ are in a block B then $\pi^{-1}(f^+\pi) = \sigma^{-1}(f^+\sigma) =: i$, for some i. When f is given inputs corresponding to a permutation in B, f is just the ith projection function. Thus, to define f, we need only know which block the given input it in. So, f itself is $\lambda\delta$-definable from the set $\{\chi_B \mid B \in \mathscr{B}\}$ via the function

$$\begin{aligned} F = \lambda x_1 \dots x_n \bullet & \text{If } \chi_{B_1} x_1 \dots x_n \text{ then } x_{i_1} \text{ else} \\ & \text{If } \chi_{B_2} x_1 \dots x_n \text{ then } x_{i_2} \text{ else} \\ & \dots \\ & \text{If } \chi_{B_j} x_1 \dots x_n \text{ then } x_{i_j} \text{ else } x_1, \end{aligned}$$

where $\{B_1, \dots B_j\} = \mathscr{B}$ and i_k is the coordinate that f projects on block B_k.

Lemma 2. *If f is regular, symmetric of type $0^n \to 0$ then f is $\lambda\delta$-definable.*

Proof. As f is symmetric, by Lemma 1, there is only one block of the equivalence class formed by $\sim_f^*$ since every π is in the stabilizer of f. As f is $\lambda\delta$-definable from the set of blocks, we have that f is $\lambda\delta$-definable outright. □

Now, consider arbitrary symmetric $f : \alpha_1 \to \alpha_2 \to \dots \to \alpha_k \to 0$. It suffices, given the above, to show that f is definable from regular, symmetric functions of type $0^n \to 0$. Consider the set of lists

$$\mathscr{L} = \Lambda^\delta(\alpha_1) \times \dots \times \Lambda^\delta(\alpha_k) = \{\langle f_1, f_2, \dots, f_k\rangle \mid f_i : \alpha_i\}.$$

For each list $L = \langle f_1, \dots, f_k\rangle$ in $\mathscr{L}$ we define a function $c_L : 0^n \to 0$, which we call the **Lth coordinate function** defined by:

$$c_L = \lambda\!\!\lambda x_1 \dots x_n \bullet \begin{cases} f(F_1 x_1 \dots x_n) \dots (F_k x_1 \dots x_n) & \text{if } x_1, \dots x_k \text{ distinct} \\ x_1 & \text{otherwise,} \end{cases}$$

where the $F_i : 0^n \to \alpha_i$ are the terms from Theorem 2 corresponding to f_i. Each coordinate function is regular (by the cases defining it) and also symmetric (as f is). So, each c_L is $\lambda\delta$-definable. Thus we need only show

that f is definable from its coordinate functions; however f is not definable outright, but is with the use of ι and $\exists$. We begin by the remark that the function $\mathsf{alldiff} : 0^n \to \mathsf{bool}_0$ which returns True if all the first n inputs are different, and False otherwise is $\lambda\delta$-definable by

$$\mathsf{alldiff} := \lambda x_1 \cdots x_n \bullet \bigwedge_{1 \le i < j \le n} \delta x_i x_j$$

where $\bigwedge$ is a contraction of 'and's parameterized by i and j which are defined in terms of the And combinator from Definition 6. Now, we can define f as

$$\begin{aligned} f = \lambda x_1 \dots x_k \bullet \iota \Bigg(& \lambda z \bullet \exists y_1 \dots y_n \bullet (\mathsf{alldiff} y_1 \dots y_n) \wedge \\ & \bigvee_{\substack{L \in \mathscr{L} \\ L = \langle F_1, \dots F_k \rangle}} \Big(\delta x_1 (F_1 y_1 \dots y_n) \wedge \dots \wedge \delta x_k (F_k y_1 \dots y_n) \\ & \wedge (\delta z (c_L x_1 \dots x_k)) \Big) \Bigg) x_1 \end{aligned}$$

where the $\bigvee$ is a contraction of 'or's parameterized by L, which is defined in terms of the Or combinator from Definition 6.

Therefore we have that f is definable as each of the c_L are definable and we can substitute the definition for c_L into the above term. □

Theorem 4. *Fix a function $f \in \mathscr{M}^n$ and $A \subseteq \mathscr{M}^n$. Then if*

$$\left(\bigcap_{g \in A} St(g) \right) \subseteq St(f) \qquad (*)$$

then f is $\lambda\delta$-definable from functions in A along with $\iota, \exists$.

Proof. It's easy to see that $\mathrm{St}(f) = \bigcap \mathrm{St}(c_L)$, where the c_L are the coordinate functions of f; for, as f and its coordinate functions are definable from each other, any permutation which fixes f must fix its coordinate functions, and any which fixes all its coordinate functions fixes the function.

Therefore, it suffices that we prove the theorem only for $f : 0^n \to 0$ and, similarly, assume all $g \in A$ be of type $0^n \to 0$. We suppose that the subset relation in $(*)$ holds. By Lemma 1, since $\mathrm{St}(g)$ is exactly the block of the equivalence relation $\sim_g^*$ containing id that the set of left cosets of $\bigcap_{g \in A} \mathrm{St}(g)$ are a finer partition of S_n than the set of left cosets of $\mathrm{St}(f)$, which are exactly the blocks of $\sim_f^*$.

Therefore, on any left coset of $\bigcap_{g\in A}\mathrm{St}(g)$ we have that f behaves like a projection operator, as the coset is entirely contained in a block of $\sim_f^*$, which in turn is contained in a block of $\sim_f$. Thus, for any permutations π we can identify the left coset of $\bigcap_{g\in A}\mathrm{St}(g)$ that π is in. Indeed, f acts uniformly on that block as a projection function, so we can make a definition similar to the above definition of f by its blocks in $\sim_f$. □

2.4 Super-symmetry and λδ-definablity

Let us return our attention to the term model $\mathscr{T}$, where members are terms with possible free variables among $x_1, x_2, \ldots$. We first state the following result of Lauchli.

Theorem 5 ([Läu70])**.** *There is a closed λδ-term F of type α if and only if there is a symmetric $F \in \mathscr{T}(\alpha)$ (recall: for terms in $\mathscr{T}(\alpha)$, symmetric means fixed under all automorphisms).*

Proof. In Lauchli [Läu70], it is stated and proved in terms of intuitionist logic: $\vdash_I \alpha$ if and only if there is an "invariant" function of type α. □

Theorem 6. *Any $F \in \mathscr{T}$ is λδ-definable if and only if it is symmetric.*

Proof. It is easy to see that every element of $\mathscr{T}$ which is λδ-definable is symmetric. To see why, note that any element is $\beta\eta\delta$-equal to a closed term, and closed terms are fixed under all automorphisms. We will just prove the converse.

Let $F \in \mathscr{T}$ by symmetric; consider F to be of type $\alpha_1 \to \cdots \to \alpha_k \to 0$. Write F as $Gx_1 \ldots x_n$, where G is closed and free variables of F are among $x_1, \ldots, x_n$. By the proposition above, we can get a closed term $H : \alpha_1 \to \cdots \to \alpha_k \to 0$, which has the long normal form

$$\lambda y_1 \ldots y_k \cdot H',$$

where H' has type 0, and free variables only among $y_1, \ldots y_k$. Consider

$$G\underbrace{H' \ldots H'}_{n \text{ times}}.$$

This is a term of type $\alpha_1 \to \cdots \to \alpha_k \to 0$ which has free variables only among $y_1, \ldots y_k$. Thus the term

$$M := \lambda y_1 \ldots y_k \cdot G\underbrace{H' \ldots H'}_{n \text{ many}} y_1 \ldots y_k : \alpha_1 \to \ldots \to \alpha_k \to 0$$

is a closed.

Recall that F is symmetric. Therefore

$$F = Gx_1 \dots x_n =_{\beta\eta\delta} Gy_1 \dots y_n,$$

for any variables $y_1, \dots, y_n : 0$. Thus, by a substitution, we have that $F =_{\beta\eta\delta} GY_1 \dots Y_n$ for any $Y_1, \dots, Y_n : 0$. Therefore, $M =_{\beta\eta\delta} F$ and is closed, thus is a $\lambda\delta$-definition of F. □

Corollary 2. *Let $h : \mathscr{T} \to \mathscr{M}$ be defined as $x_i \mapsto i$. This is called the canonical homomorphism. A function $f \in \mathscr{M}$ is $\lambda\delta$-definable if and only if there is $F \in h^{-1}(f)$ symmetric.*

Proof. Once again the forward direction is straightforward. For the backward direction, we just apply the last theorem. By the last theorem, if $F \in h^{-1}(f)$ is symmetric then it is $\lambda\delta$-definable by some closed term G. $h(G) = f$ and G is closed, therefore G is also a $\lambda\delta$-definition for f. □

Definition 20. We call a homomorphism $h : \mathscr{T}^n \to \mathscr{M}^m$ **canonical** if $x_i \mapsto i$ for all $1 \leq i \leq m$.

We say that an $F \in \mathscr{T}^n$ is **super-symmetric** if for every homomorphism $\phi : \mathscr{T}^n \to \mathscr{T}^n$, $\phi(F)$ is symmetric.

Theorem 7 ([GS14])**.** *$f \in \mathscr{M}^m$ is $\lambda\delta$-definable if and only if there is some $n > m$ and $F \in \mathscr{T}^n$ super-symmetric such that for all canonical homomorphisms $h : \mathscr{T}^n \to \mathscr{M}^m$ we have $h(F) = f$.*

Proof. The left to right direction is trivial since f being $\lambda\delta$-definable gives us a closed term which will satisfy all the requirements.

For the other direction, fix $f \in \mathscr{M}^m$ of type $\alpha_1 \to \alpha_2 \to \dots \to \alpha_k \to 0$. Suppose that $n > m$ and $F \in \mathscr{T}^n$ is super-symmetric where all homomorphisms $h : \mathscr{T}^n \to \mathscr{M}^m$ have $h(F) = f$. Write $F = F'x_1 \dots x_j$ where F' is a closed term.

The idea is as follows: we will do induction on the number of free variables on F, j. We will construct a new term M which has $j-1$ free variables and still has the property that it is super-symmetric is sent to f under all canonical homomorphisms. At the end of our construction, we will have eliminated all free variables, and will have constructed a closed term M which is sent to f under all canonical homomorphisms. But, as M will be closed, M will be a $\lambda\delta$-definition for f.

To start the induction, if $j = 1$, then $F = F'x_1$. As $n > m \geq 1$, we know $n > 1$ so that $x_n \neq x_1$. F is super-symmetric, and therefore it is

symmetric, so under the automorphism sending x_1 to x_n we know $F'x_1 = F'x_n$. As $n > m$, we have freedom with our canonical homomorphism to send x_n anywhere; in particular for any $1 \leq s \leq m$ we can define canonical homomorphism h where $h(x_n) = s$. Therefore $f = F's$ for all s. Therefore, we may replace x_1 in F by anything of type 0 and it would still be sent to f through any canonical homomorphism.

By Theorem 5, there is a closed term G of type $\alpha_1 \to \dots \alpha_k \to 0$. We can write F as

$$\lambda z_1 \dots z_k . F' x_1 z_1 \dots z_k$$

by doing η expansions. Then, replacing x_1 to form the term

$$\lambda z_1 \dots z_k . F'(G z_1 \dots z_k) z_1 \dots z_k,$$

we have a closed $\lambda\delta$-term which is equal to f.

If $j > 1$, then we wish to eliminate the variable x_j. If $j > m$ then we already have freedom to send x_j to any number in a canonical homomorphism h. Therefore, for every $1 \leq s \leq m$, by picking a canonical homomorphism which sends x_j to s we have

$$f = h(F'x_1 \dots x_j) = F'1 \dots n(h(n+1)) \dots (h(j-1))s.$$

As s is unrestricted, we can replace x_j with anything of type 0 and the above is still preserved. In particular, doing an η expansion of F gives us $F = \lambda z_1 \dots z_k . F' x_1 \dots x_j z_1 \dots z_k$ and then replaces x_j we get

$$f = h(\underbrace{\lambda z_1 \dots z_k . F' x_1 \dots x_{j-1} (F' x_1 \dots x_1 z_1 \dots z_k) z_1 \dots z_k}_{M}).$$

Note that M has only $j-1$ free variables. It remains to show that M is super-symmetric. This, however, is not hard to see. Under the map $x_i \mapsto x_1$ for all $1 \leq i \leq j$ we have that, since F is super-symmetric, $F'x_1 \dots x_1$ is symmetric, and therefore preserved under all automorphisms. Therefore, for any homomorphism $\phi : \mathscr{T}^n \to \mathscr{T}^n$ we will have $\phi(M)$ symmetric as $\phi(F)$ was symmetric and M is just F with a free variable replaced by a symmetric term.

If $1 < j \leq m < n$, we have by the symmetry of F, after applying the automorphism which sends x_j to x_n, that

$$F = F' x_1 \dots x_{j-1} x_n.$$

Now, we have the freedom to send x_n to anywhere under any canonical homomorphism, and thus we can repeat what we did above to eliminate x_n. □

2.5 Conclusion and future work

In this chapter we have established a novel proof of a folklore theorem describing necessary and sufficient conditions to definablity of functions in these finitary models in the $\lambda\delta$-calculus with $\exists$ and ι. In addition, we prove necessary and sufficient conditions for definablity in the $\lambda\delta$-calculus by this new notion of super-symmetry.

For future work, we'd like to find more natural conditions than super-symmetry. In addition, a study of the automorphisms on $\mathscr{T}^n$ would serve to be interesting. As established, there are non-trivial automorphisms, and they seem difficult to describe.

Chapter 3

Q_0 and Extensionality

3.1 The type theory Q_0

In Chapter 2, we discussed a system of type theory for first-order classical logic in the simple typed lambda calculus based on the $\lambda\delta$-calculus. In this chapter, we follow the direction of Andrews in [And02] to do higher-order logic. This system is a refinement Church's in [Chu40]. This system of Church was refined by Henkin in [Hen63] and Andrews in [And63].

Definition 21 (Types of Q_0)**.** In the system Q_0 we have two atomic types: 0 and bool. The type 0 represents the **domain of individuals**. The type bool represents the domain of boolean values.

Definition 22 (Terms of Q_0)**.** Our language is the language of the simple typed lambda calculus (see Section 1.0.2) augmented with some constants. One constant is $\iota : (0 \to \mathsf{bool}) \to 0$, which called the **description operator**; the intended semantics are to match that of Definition 16.

Another is an equality operator, similar to δ from Definition 15, except that it is a schema for each type α. We call these constants $Q_\alpha : \alpha \to \alpha \to \mathsf{bool}$, where α is a type. When it is clear from context what type we are considering, we will write Q instead of Q_α.

In the above language, we will do higher-order logic. To that end, in Figure 3.1 you can see all the usual logical symbols defined in terms of these primitives.

On the aforementioned terms, we axiomatize our system. The axioms are displayed in Figure 3.2.

Remark 9. In the formulation by Andrews [And02], the β-rule is replaced by 5 other rules essentially representing the usual combinatory axioms of

Definition	Meaning
$F^\alpha = G^\alpha$	$Q_\alpha FG$
$A^{\mathsf{bool}} \iff B^{\mathsf{bool}}$	$A = B$
$\top^{\mathsf{bool}}$	$Q_{\mathsf{bool}} = Q_{\mathsf{bool}}$
$\forall x^\alpha . A^{\mathsf{bool}}$	$(\lambda x^\alpha . \top) = (\lambda x^\alpha . A)$
$\bot^{\mathsf{bool}}$	$\forall x^{\mathsf{bool}} . x$
$\neg A^{\mathsf{bool}}$	$A \iff \bot$
$A^{\mathsf{bool}} \wedge B^{\mathsf{bool}}$	$(\lambda G^{\mathsf{bool}\to\mathsf{bool}\to\mathsf{bool}} . G \top \top) = (\lambda G^{\mathsf{bool}\to\mathsf{bool}\to\mathsf{bool}} . GAB)$
$A^{\mathsf{bool}} \Longrightarrow B^{\mathsf{bool}}$	$A \iff (A \wedge B)$
$A^{\mathsf{bool}} \vee B^{\mathsf{bool}}$	$\neg((\neg A) \wedge (\neg B))$
$\exists x^\alpha . A^{\mathsf{bool}}$	$\neg(\forall x^\alpha . \neg A)$

Figure 3.1: Definition of logical symbols in Q_0.

$((G^{\mathsf{bool}\to\mathsf{bool}} \top) \wedge (G \bot)) \iff (\forall x^{\mathsf{bool}} . Gx)$	*(Excluded Middle)*
$(x^\alpha = y^\alpha) \to (G^{\alpha\to\mathsf{bool}} x \iff Gy)$	*(Substutitivity of Equality)*
$(F^{\alpha\to\beta} = G^{\alpha\to\beta}) \iff (\forall x^\alpha . Fx = Gx)$	*(Extensionality)*
$\iota(QM^0) = M$	*(Description)*
$(\lambda x^\alpha . M^\beta) N^\alpha = M[x := N]$	*(β-rule)*

Figure 3.2: The axioms of Q_0.

lambda reduction and rules to allow α-renaming. Here, we prefer to just give one rule based on substitution and take α-equivalence to be primitive along with syntactic equality, as we did in our introduction to the lambda calculus in Section 1.

Definition 23 (Provability in Q_0)**.** We define a notion of provability $\vdash$ inductively. First we say that $\vdash M$ for any instance M of an axiom in Figure 3.2. Further, if $\vdash M = N$ and $\vdash C$ then $\vdash C'$ where C' comes from replacing a subformula M with N.

This system is conservative over sentences from first-order logic [And02]. Meaning, if ϕ is a first-order sentence which is translated into the system Q_0 using the suggestive notions from Figure 3.1 then $\vdash \phi$ according to the definition above if and only if ϕ is a theorem of first-order logic. Therefore, since provability in first order logic is undecidable [Tur36, Chu36a, Chu36b], so is provability in Q_0.

3.2 Semantics

For the semantics of Q_0 we take a type structure $\mathscr{M}$ where the type constants are interpreted as:

$$\begin{aligned} \mathscr{M}(0) &= X \text{ where } X \text{ is a set, and} \\ \mathscr{M}(\mathsf{bool}) &= \{T, F\}\,. \end{aligned}$$

We interpret terms in the usual way, but for constants we do the following:

$$\begin{aligned} [\![Q^{\alpha\to\alpha\to\mathsf{bool}}]\!]_\phi &= \text{characteristic function of equality on } \mathscr{M}(\alpha) \\ [\![\iota^{(0\to\mathsf{bool})\to 0}]\!]_\phi &= \text{any function that takes characteristic} \\ &\qquad \text{functions of singleton to the member.} \end{aligned}$$

These semantics are sound and complete for the system Q_0; for a proof, see [And02].

3.3 A model of Q_0–Ext

In this section, we prove the following.

Theorem 8 (Gunther, Statman)**.** *The axiom of extensionality is independent of the other axioms of* Q_0*.*

Proof. To do this, we define a structure $\mathscr{M}$ as follows which satisfies the axioms of Q_0 but violates extensionality. We first describe the model in which our terms our interpreted.

$$\begin{aligned}\mathscr{M}(0) &= \mathbb{N}\\ \mathscr{M}(\mathsf{bool}) &= \{T, F\}\\ \mathscr{M}(\alpha \to \beta) &= \mathbb{N} \times \{f : \mathscr{M}(\alpha) \to \mathscr{M}(\beta)\}\end{aligned}$$

We then interpret our terms in the following way:

$$\begin{aligned}[\![x]\!]_\phi &:= \phi(x)\\ [\![F^{\alpha\to\beta}G^{\alpha}]\!]_\phi &:= f(m,g) \qquad \text{if } [\![F]\!]_\phi = (n,f) \text{ and } [\![G]\!]_\phi = (m,g)\\ [\![\lambda x.M]\!]_\phi &:= (0, \lambda\!\!\lambda z.[\![M]\!]_{\phi[x:=z]})\\ [\![Q^{\alpha\to\alpha\to\mathsf{bool}}]\!] &:= \left(1, \lambda\!\!\lambda x. \left(0, \lambda\!\!\lambda y. \begin{cases} T & \text{if } x = y \\ F & \end{cases}\right)\right)\\ [\![\iota^{(0\to\mathsf{bool})\to 0}]\!] &:= \left(0, \lambda\!\!\lambda(k,f). \begin{cases} 0 & \text{if } f^{-1}(\{T\}) = \emptyset \\ y & \text{for } y = \min(f^{-1}(\{T\})) \end{cases}\right) \qquad (\ddagger)\end{aligned}$$

where ϕ is an environment. In $(\ddagger)$, we are taking the minimum element of a nonempty set of natural numbers which satisfy the predicate provided as input. The choice of the minimum is arbitrary; any function which behaves like the description operator as described in Section 3.2 would serve.

The intuition beyond this model is that we are adding fairly arbitrary labels which discern members of the model which are extensionally equal.

Clearly if M is a term of type α then fix $[\![M]\!]_\phi \in \mathscr{M}(\alpha)$ for any environment ϕ. We call the natural number in the first coordinate of the pair from elements of $\mathscr{M}(\alpha \to \beta)$ the label of the function.

We say $\vdash M$ in Q_0 if either 1) M is an axiom or 2) $\vdash M = N$ and $\vdash C$, then we can say $\vdash C'$ where C' comes from replacing an instance of M with N in C. We will show that if $\vdash M$ then $[\![M]\!]_\phi = T$ for any valuation of the variables ϕ.

Let's verify that this is a model for Q_0-Ext. First we will verify it models the axioms and the equality rule. Note that $[\![\top]\!] = T$ and $[\![\bot]\!] = F$, and moreover logical connectives are all as expected. Since the logical connectives work as expected, we will argue the truth of each inside the model using the standard interpretation of logical connectives. For example, if an axiom is $A \to B$, we can argue its truth by assuming $[\![A]\!]_\phi$ and proving $[\![B]\!]_\phi$.

Excluded middle

$$((G^{\mathsf{bool}\to\mathsf{bool}}\top) \wedge (G^{\mathsf{bool}\to\mathsf{bool}}\bot)) \iff ((\lambda x^{\mathsf{bool}}\mathbf{.} G^{\mathsf{bool}\to\mathsf{bool}}x) = (\lambda x\mathbf{.}\top))$$

Suppose $[\![G]\!]_\phi = (n, g)$. g is a function from $\{T, F\}$ to itself; therefore, do cases on four possible values of g. Each case is straightforward. Notice, the label on the term on the right-hand side is 0 for both terms.

Substutitivity of equality

$$(X^\alpha = Y^\alpha) \to (G^{\alpha\to\mathsf{bool}}X \iff G^{\alpha\to\mathsf{bool}}Y)$$

Suppose $[\![G]\!]_\phi = (n, g)$, and assume $[\![X = Y]\!]_\phi = T$. Therefore, we have $[\![X]\!]_\phi = [\![Y]\!]_\phi$, by the definition of $[\![=]\!]_\phi$. Therefore, $g\,([\![X]\!]_\phi) = g\,([\![Y]\!]_\phi)$. So $[\![GX = GY]\!]_\phi = T$.

Description

$$\iota(QY^0) = Y$$

$[\![QY]\!]_\phi = (0, \chi_{\{[\![Y]\!]_\phi\}})$, where $\chi_{\{[\![Y]\!]_\phi\}}$ is the characteristic function of $[\![Y]\!]_\phi$. By definition of $[\![\iota]\!]_\phi$, the left-hand side of the above exactly $[\![Y]\!]_\phi$.

β-reduction

$$(\lambda x\mathbf{.} M)N = M[x := N]$$

We do a familiar calculation:

$$[\![(\lambda x\mathbf{.} M)N]\!]_\phi = (\lambda\!\!\lambda z\mathbf{.}[\![M]\!]_{\phi[x:=z]})[\![N]\!]_\phi = [\![M]\!]_{\phi[x:=[\![N]\!]_\phi]}.$$

Now, we need only show that $[\![M]\!]_{\phi[x:=[\![N]\!]_\phi]} = [\![M[x := N]]\!]_\phi$. We will be explicit to show that the choices of labels to constants and abstractions will not spoil this property. We do induction on the shape of M.

If M is a variable $y \not\equiv x$ then the left-hand side and right-hand side above are both $\phi(y)$. If M is the variable x then the right and left-hand side of the above are both $[\![N]\!]_\phi$. If M is any constant then it is preserved under any variable substitution.

If M is the term PQ then

$$[\![PQ]\!]_{\phi[x:=[\![N]\!]_\phi]} = p([\![Q]\!]_{\phi[x:=[\![N]\!]_\phi]}) = p([\![Q[x := N]]\!]_\phi)$$

where $[\![P]\!]_{\phi[x:=[\![N]\!]_\phi]} = [\![P[x := N]]\!]_\phi = (n, p)$. This is the same however as $[\![P[x := N]Q[x := N]]\!]_\phi$, which is $[\![M[x := N]]\!]_\phi$ as required.

If $M \equiv \lambda y . P$ then

$$\begin{aligned}[\![\lambda y . P]\!]_{\phi[x:=[\![N]\!]_\phi]} &= (0, \lambda\!\!\lambda z . [\![P]\!]_{\phi[x:=[\![N]\!]_\phi][z:=y]}) \\ &= (0, \lambda\!\!\lambda z . [\![P[x := N]]\!]_{\phi[z:=y]}) \\ &= [\![M[x := N]]\!]_\phi\end{aligned}$$

Equality Rule

Now we verify the equality rule. We show if for all valuations ϕ we have $[\![M = N]\!]_\phi = T$ then we have $[\![C = C']\!]_\phi = T$ for all valuations ϕ where C' is obtained from substituting an instance of M in C with N. This clearly implies the result.

To do this we will do an induction on the shape of C. If C is a variable which is the same as M then after the substitution we have $M = N$, and we assumed this is evaluated to T. Similarly for constants.

If C is an application, then $C \equiv PQ$. If M is a subterm of P, then by induction hypothesis we have that $[\![P = P']\!]_\phi$ where P' is obtained from substituting an instance of M with an instance of N. Therefore, $[\![P]\!]_\phi = [\![P']\!]_\phi = (n, p)$. So, $[\![M]\!]_\phi = p([\![Q]\!]_\phi) = [\![P'Q]\!]_\phi$. Similarly if M is a subterm of Q.

If C is an abstraction, then $C \equiv \lambda x . P$, then the instance of M lives in P. By induction hypothesis, for any valuation ψ we have $[\![P]\!]_\psi = [\![P']\!]_\psi$ where P' is obtained from substituting an instance of M for N in P. In particular, the equality holds for any z which x is mapped to in ϕ. Therefore, $\lambda\!\!\lambda z . [\![P]\!]_{\phi[x:=z]} = \lambda\!\!\lambda z . [\![P']\!]_{\phi[x:=z]}$. And since the label for both C and $C' \equiv \lambda x . P'$ is 0, the result follows.

Extensionality fails

Finally, we will show that this model fails to satisfy the extensionality axiom. With some of the definitions in Figure 3.1 unwound, the extensionality axiom is

$$(F = G) \iff (\lambda x . Fx = Gx) = (\lambda x . \top).$$

For this, we take $F := Q$ and $G := \lambda x . Qx$ (where the type we are considering Q over is unimportant). We will show that the left-hand side evaluates to F but the right-hand side evaluates to T.

By the above verifications that this model satisfies the β-rule, we have that $\vdash Qx = (\lambda y . Qy)x$, and so it is easy to check that

$$[\![(\lambda x . Qx = (\lambda y . Qy)x) = (\lambda x . \top)]\!]_\phi = T.$$

However, $[\![Q]\!]_\phi$ has label 1, but $[\![\lambda x . Qx]\!]_\phi$ has label 0, therefore,

$$[\![Q = \lambda x . Qx]\!] = F.$$

Note, this also shows that η-equivalence does not hold in this model. □

3.4 Conclusion and future work

In this chapter we prove that extensionality is independent from the other axioms of Q_0. In future work, it would be nice to build a better model. Attempts by the author to find a model where the Q equality schema was interpreted non-uniformly proved unsuccessful. Similarly, it would be worth investigating whether η is coupled with extensionality in Q_0; in our model, both fail, but typically, η is weaker than extensionality, which suggests another model could be find where η succeeds and extensionality fails.

Chapter 4

Self application and polymorphism

4.1 Survey of system F

System F is another type system which is **polymorphic**, meaning that a term is actually given multiple types. Just as the simply typed lambda calculus relates to intuitionistic implicational propositional logic, system F relates to second-order intuitionistic propositional logic. In system F, the assignment of types is described by variable parameters, and to get other types one instantiates those parameters for other types (like a universally quantified formula in logic).

For a detailed survey of system F, see the books of Girard [GTL89] and Sørensen and Urzyczyn [SU06].

Definition 24. Formally, we define the set of types from F inductively as follows: a is a type, where a is a type variable; if α and β are types then $\alpha \to \beta$ is a type; and if α is a type and a is a type variable then $\forall a.\alpha$ is a type.

Remark 10. We assume that same notation conventions from Remark 1 for this type binder as we do for the lambda binder. Namely, we will assume α-equivalence for types on a syntactic level, that all bound type variables have different names than free variables, and no two bound variables have the same name. Again, the dot reminds us that bindings occur in the largest scope possible.

We now define what it means for for a term M to have type α, written $M : \alpha$, in system F. Γ is an arbitrary context; that is, a finite partial function

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ (ax) \qquad \frac{\Gamma \vdash M : \alpha \to \beta \qquad \Gamma \vdash N : \alpha}{\Gamma \vdash MN : \beta}\ (app)$$

$$\frac{\Gamma, x : \alpha \vdash M : \beta}{\Gamma \vdash \lambda x . M : \alpha \to \beta}\ (abs)$$

$$\frac{\Gamma \vdash M : \alpha \qquad a \notin \Gamma}{\Gamma \vdash M : \forall a . \alpha}\ (gen) \qquad \frac{\Gamma \vdash M : \forall a . \alpha \qquad \beta = \alpha[a := \gamma]}{\Gamma \vdash M : \beta}\ (inst)$$

Figure 4.1: Deduction rules for system F (Curry-style typing)

from term variables to types from Definition 24. We can think of Γ as a finite like of variable assignments $x : \alpha$, where each variable is distinct. The rules are given in Figure 4.1. In the rule *(inst)*, we call the type β an **instance of** α and the process **instantiation**.

As in the simply typed lambda calculus, we have presented the system in the style of Curry-style typing, where terms live in the untyped lambda calculus, as opposed to Church-style typing, where term decorations governs types. Church-style typing is more complicated in system F than it is in the simply typed case since partial type information may be insufficient to reconstruct the type for the entire term [Pfe88]. Therefore, in a Church-style approach of the system F, the terms convey addition type information, which is done by introducing a type binder and making all type instantiations explicit.

Even in a Curry-style typing system, there are other formulations that could be given. For example, there is a syntax directed presentation where the shape of the term corresponds to a unique rule that was used last. The presentation in Figure 4.1 is clearly not syntax directed since the last two rules, *(inst)* and *(gen)*, do not depend at all on the shape of M, and be used repeatedly to add and remove a quantifier on the type. This presentation, however, has the advantage that it mimics a standard presentations of intuitionistic second-order propositional logic.

The Curry-style typed version of system F enjoys the subject reduction property defined in Definition 11. In addition, it also satisfies the subterm property; that is, if M is a term such that $M : \alpha$ for some α then for every subterm M' of M there is a type α' such that $M' : \alpha'$.

System F was created by Girard [Gir72] and by Reynolds [Rey74] independently, the latter calling it the polymorphic lambda calculus. It was

proved by Girard that all terms which have types in this system are normalizing in the introductory work, which was expanded by Prawatz [Pra71] into a proof of strong normalization.

There are many different systems of typed lambda calculus where a term may be given more than one type, such as intersection typed systems. System F is different in that the family of types given to a term are all governed by parameters. It is actually at the bottom of a entire hierarchy of parametric polymorphic systems also explored by Girard. The higher-order systems allow quantification of higher-order type predicates.

It was proved by Wells [Wel99] that typability in system F is undecidable; that is, it is an undecidable problem given a term M in the untyped lambda calculus to determine if there is a type α such that $M : \alpha$. Type inhabitation is also undecidable, and is an older result by Löb [Löb76] and Gabbay [Gab74, Gab81]. This, along with the complexity of the strong normalization proof, indicate a great deal of complexity of this system over that of the simple types.

System F is more limited in its ability to type terms than non-parametric polymorphic systems, such as intersection types. For example, as proved by Giannini and Della Rocca [GDR88], not all strongly normalizing terms have a type in system F. A classical example of such a term is $22\mathsf{K}$, where 2 is the Church numeral from Definition 9. It's a theorem of Reynolds that all normal terms have types in system F. In fact, all normal forms have a type in a much weaker system based on intuitionistic implicational logic with a bottom type (with the elimination rule *ex falso*).

If α is a type of system F we can view α as a rooted binary tree, where each node is decorated with a quantified (possibly empty) set of variables, and each leaf is decorated with a possibly quantified type variable. An arrow type is a branch where the left child is the input type and the right child is the output type.

Using these binary trees, we can talk about a path from the root to a node (or, usually, a leaf). We can also talk about the depth of a variable or binding location of a variable in a type, which would correspond to its depth in the tree. We can also describe the location of a variable or binding location by its path from the root to the location. Here, we will be particular interested in the leftmost path of a type: in particular, its depth and binding location. The primary fact that we will use about depths and paths are that any free variable a in a type α will have the same path for all β which are instances of α. That is, you cannot change the location of a free variable in a type by instantiation.

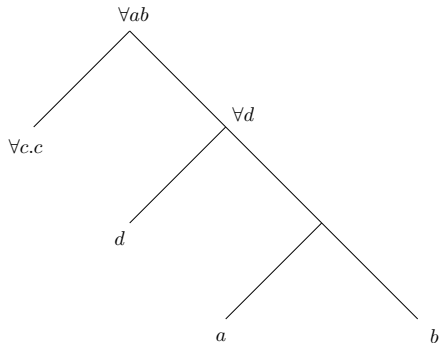

Figure 4.2: The type $\forall ab\bullet(\forall c\bullet c) \to (\forall d\bullet d \to a \to b)$

Example 1. The type

$$\forall ab\bullet(\forall c\bullet c) \to (\forall d\bullet d \to a \to b)$$

would translate to the tree in Figure 4.2. Its leftmost path is the bound variable c which is depth 1, and is bound at depth 1.

4.2 Expansions

Since system F types all normal terms but does not type all strongly normalizing terms, it is necessarily true that system F is not closed under β-expansions. We say that F permits M-expansions if for any N typable in system F, if $MN \to_\beta P$ and P has a type in system F then MN has a type in system F. Our goal is an investigation of ω-expansions: that is, the question if MM has a type, does ωM have a type?

There are no known terms M which act as counterexamples to terms typable in system F being closed under ω-expansions, and this seems like a difficult thing to resolve. If the answer is yes, it seems the proof will give a uniform construction of the type of ω given a typing of MM and the shape of M. Where this typing information would fit in is mysterious since system F lacks most general types. If the answer is no, then such a term seems difficult to construct since, on its face, being typed for being an argument for ω gives less information than one would like, which we will now discuss.

The type of ω has one primary restriction: it must be of the form $\forall \bar{a}\bullet\alpha$ where the leftmost path of the type α is a variable in $\bar{a}$. To see why, consider if the leftmost path was not bound at this level. Then, it would not be

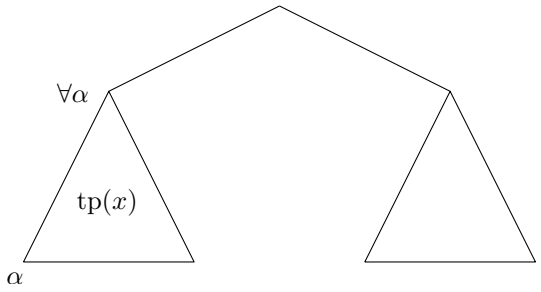

Figure 4.3: Shape of type of ω

possible to change the depth of this variable by instantiation, but in order to apply x to x, the depth of the leftmost paths must differ by 1 in two different instantiations of the type. This restriction is shown in Figure 4.3. Since the leftmost path of ω must be bound at this depth, we can say that terms M such that ωM has a type require that M can be typed with a free leftmost path.

This realization of this restriction has some power in itself. For instance, it supplies a fairly concise proof that $\omega\omega$ has no type in system F which does not resort to the fact that $\omega\omega$ it not normalizing. It follows simply from the fact that the type of the ω subterm in the argument position must be typed with a leftmost path bound at depth 0 (which means, it must be typable with a free leftmost path) which is impossible.

Note that there is nothing, a priori, restricting a term which requires the leftmost variable be bound from being self-applied. This is because it could be that there is a requirement that the variable is bound, but the binding location could be flexible and vary in either instance. For example, consider $\mathsf{I} \equiv \lambda x_{\bullet} x$ which can be typed in the following ways:

$$\mathsf{I} : (\forall\alpha_{\bullet}\alpha) \to \beta$$
$$\mathsf{I} : ((\forall\alpha_{\bullet}\alpha) \to \beta) \to ((\forall\alpha_{\bullet}\alpha) \to \beta).$$

Of course, II is typable in this way where both types of I have bound leftmost path. But, $\omega\mathsf{I}$ certainly does not have a type with either type of I above.

The remainder of the chapter will be to show that finding such a term which is normal is not possible.

Theorem 9 (Gunther, Statman)**.** *If M is a normal term that requires a bound leftmost path then MM does not have a type in system F. Furthermore, MM is not even strongly normalizing.*

4.3 A non-normalizing class

We go on a brief diversion to explore some interesting normal terms which all fail to normalize under self-application in the untyped lambda calculus. We will then discuss the relationship between terms of this class and requiring a bound leftmost path in system F.

For this we first develop the idea of an important variable. Consider a normal combinator M. Then $M \equiv \lambda x.P$.

Remark 11. Before establishing the definition of an important variable, it is worth pointing out that important variables are not actually variables. They are bound variables, and because we are associating terms up to α-equivalence, we are actually talking about binding sites. Despite this, it is very important that we be able to talk about binding sites easily, which is is why we are conflating variable names with binding sites in the below definitions and proofs.

Definition 25 (Important variable)**.** We say that a variable z is **important** if:

- $z \equiv x$ where x is the outer bound variable, or
- $P \equiv C\,[y\lambda z.Q]$ where C is any one-hole context and y important.

Namely, the outer bound variable is important, this causes the importance of other variables via application of that variable to the binding site of another.

Definition 26 (Influence tree)**.** The collection of important variables can be partially ordered by which variables cause other variable importance: we say $z \leq y$ if y appears in the proof of the importance of z.

This partial order makes a rooted tree, where the root is the outer bound variable x, and the child of a variable y are all variables z for which $y\lambda z.Q$ is a subterm. We will call this tree the **influence tree** of a normal term.

Example 2. If M is the term

$$\lambda x.x\lambda y.y\,(\lambda v.v)\,\big(y\lambda z.z\,(\lambda w.wy)\,\big)$$

then its influence tree is given in Figure 4.4. Note, the important variables w and y are applied to each other.

The motivation of this definition is it gives us a necessary condition for strong normalization of a normal term self-applied.

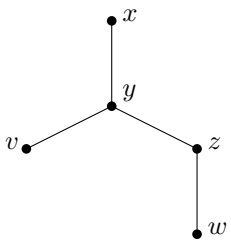

Figure 4.4: Influence tree of a term.

Theorem 10 (Gunther, Statman)**.** *If M is a normal term with two important variables applied to each other then MM is not strongly normalizing.*

To prove this, we will prove that a class of terms in the untyped lambda calculus is not strongly normalizing.

We say that a two place context P is **eventually applicative** if

- $P[x, y] \equiv xy$ or $P[x, y] \equiv yx$, or
- $P[x, y] \equiv x\lambda z.Q[y, z]$, where Q is eventually applicative, or
- $P[x, y] \equiv y\lambda z.Q[x, z]$, where Q is eventually applicative.

Then we define a class of terms which we write $\mathscr{C}$ inductively as follows:

- $\lambda x.P[x, x] \in \mathscr{C}$ if P is eventually applicative;
- $\lambda x.P[x, T] \in \mathscr{C}$ if P is eventually applicative and $T \in \mathscr{C}$; and
- $\lambda x.P[T, x] \in \mathscr{C}$ if P is eventually applicative and $T \in \mathscr{C}$.

Note that every term $M \in \mathscr{C}$ has two important variables applied to each other. Further, M is a term in the λI-calculus; that is, there are no bound variables that do not appear in the term which is abstracted.

Lemma 3. *If $M, N \in \mathscr{C}$ and P is eventually applicative then*

$$P[M, N] \to_\beta Q[M', N']$$

where Q is eventually applicative and $M', N' \in \mathscr{C}$.

Proof. This we do by cases on the context P and subcases on on term M (or N).

First consider the case where $P[x, y] \equiv xy$ (or, analogously, yx). Then we have that $P[M, N] \equiv MN$. We now do subcases on M. If $M \equiv \lambda x.Q[x, x]$

where Q is eventually applicative then $MN \to_\beta Q[N,N]$, as required. Otherwise, if $M \equiv \lambda x.Q[x,T]$ where Q is eventually applicative and $T \in \mathscr{C}$ then $MN \to_\beta Q[N,T]$, as required. $M \equiv \lambda x.Q[T,x]$ is analogous.

In the case where $P[x,y] \equiv x\lambda z.Q[y,z]$ then $P[M,N] \equiv M\lambda z.Q[N,z]$. This reduces to the previous case as M and $\lambda z.Q[N,z]$ are in $\mathscr{C}$ by using the trivial eventually applicative context $R[x,y] = xy$.

In the case where $P[x,y] \equiv y\lambda z.Q[x,z]$ then $P[M,N] \equiv N\lambda z.Q[M,z]$ which similarly reduces to the first case. □

Lemma 4. *If $M, N \in \mathscr{C}$ then $P[M,N]$ does not have a normal form, for any eventually applicative context P.*

Proof. Begin by noting, for any eventually applicative context P, $P[M,N]$ is never normal since all $M, N \in \mathscr{C}$ are abstractions.

Furthermore, by the Lemma 3 we have that $P[M,N] \to_\beta Q[M',N']$ which itself is not normal, and has the same property. This shows that $P[M,N]$ is not strongly normalizing as it has an infinite reduction strategy.

As $P[M,N]$ is a λI-term, strong normalization and normalization coincide (see Remark 3). □

Lemma 5. *If $M, N \in \mathscr{C}$ then MN does not that have a normal form.*

Proof. Set $P[x,y] :\equiv xy$ and apply Lemma 4. □

Now, we have the appropriate infrastructure to prove Theorem 10.

Proof of Theorem 10. Consider a term M with two important variables applied. Call these variables y and z. We can build a corresponding 'skeleton term' M' from the influence tree of M and the binding order. This skeleton term only has variables that are in the influence tree of M that are ancestors of either y or z.

We build this skeleton term inductively as follows: $M \equiv \lambda x.P$. If $P = C[xx]$ (so $y \equiv z \equiv x$) then the skeleton term is simply $\lambda x.xx$.

Otherwise, $P \equiv C[x\lambda w.Q]$ where w is an important variable that is the first bound important variable which is either an ancestor of z or y in the influence tree of M. Then the skeleton term of M is $\lambda x.x\lambda w.Q'$ where Q' is the skeleton term of Q under the assumption that x and w are both important.

Denote the skeleton term of M as M'. Note that $M' \in \mathscr{C}$, so $M'M'$ is not normalizing by Lemma 5. All applications in $M'M'$ have a corresponding application in MM, and furthermore, this correspondence continues after any contractions. Therefore the infinite reduction sequence that exists in

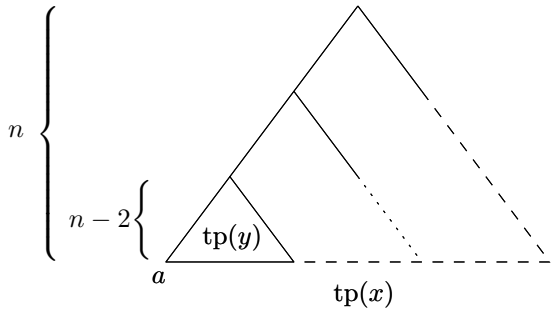

Figure 4.5: Depth of important variables leftmost path.

$M'M'$ can be emulated in MM, and so MM is not strongly normalizing, as required. □

So we have shown a class of normal terms whose self-application cannot have types in system F (for lack of being strongly normalizing): those are the normal terms which have two important variables applied to each other. These terms actually fail syntactically to be well-typed for ω based on the shape of their types in system F.

Theorem 11 (Gunther, Statman)**.** *If M is a normal term with two important variables applied then M cannot be typed in system F with a free leftmost path.*

Proof. Let $M \equiv \lambda x.P$ and suppose that there is a typing of M without a bound leftmost path. Then the leftmost path of the type of x is free, and it has some depth n which is fixed under all instantiations.

By a straightforward induction on the influence tree of the term, all important variables must also have a free leftmost path.

We next claim that the parity the depth of the leftmost path of all the important variables is the same as the parity of x. The idea of the proof is in Figure 4.5. We can do induction on the influence tree, noting that if a variable x causes the importance of y then the depth of the leftmost path of y is exactly 2 less than the depth of the leftmost path of x, and these depths are unaffected by other instantiations.

Therefore, since all important variables have a free leftmost path of the same parity, it is not possible that they are ever applied to each other as that would require a difference in leftmost path depth of exactly 1. □

4.4 Avoiding a bound leftmost path

In the last section, we showed a class of normal combinators that requires a bound leftmost path, and interestingly cannot be self-applied. We will now show that those are exactly the normal terms that require a bound leftmost path, thus completing the proof Theorem 9. We will do this by typing all the other terms with a free leftmost path.

Theorem 12 (Gunther, Statman)**.** *If M is a normal combinator in which no two important variables are applied then there is a typing of M which has a free leftmost path.*

Proof. We will inductively type M maintaining the following:

1. All unimportant variables will get type $\bot := \forall a.a$.
2. All important variables will get some type in this inductive class:
   - $a \to \bot$, and
   - $\forall b.(\alpha \to b) \to \bot$ where α is in the class.

Call the complexity of a type in this class the length of the proof that the type is in this class.

3. If z is an important variable with complexity n then we may alter the typing of the term to give z a type with any complexity $m \geq n$. This allows important variables who are siblings in the influence tree to be given the same type.

We initially assign to each leaf variable of the influence tree the type $a \to \bot$. If z is an leaf variable of influence tree then we have $M \equiv C[\lambda z.P]$, where P is the entire term that z abstracts. We initially assign z type $a \to \bot$, and will now verify that this is valid way to type z so that all the invariants above are satisfied.

We claim we can type P to be one of the following: either $\bot \to \cdots \to \bot$ or $\bot \to \cdots \to \bot \to \alpha$ where α is the type of an important variable which is either the type of z or one in which we can determine independently from the type of z.

To see why, there are only 4 possibilities for the form of P: it is either an important variable, an unimportant variable, an application, or an abstraction of an unimportant variable. In the case when it is an unimportant variable, the type of P with be $\bot$. If it is an important variable, it may be z, which is given a type of the form above, or some other important variable

w. This variable w will be assigned some type α, which we will verify when we assign a type for w that it will not require knowledge of the type of P to properly type.

If it is an application, the P will be $\bot$ if we fulfill our obligations in typing all important variables since any application will either be between unimportant variables (all given type $\bot$) or an important variable with a type in the above class and an unimportant variable (whose output can be given type $\bot$). Otherwise, it is an abstraction, in which case it is $\bot \to \phi$ for some ϕ where ϕ has the same kind of form as the possible types of P.

Further note, we can make the type of z as complex as we want as it is never applied to anything other then a term of type $\bot$; it is never applied to a lambda as then it wouldn't be a leaf important variable, and it is not applied to an important variable as we are assuming no two important variables are applied. Therefore it is either applied to an application or an unimportant variable, which both have type $\bot$.

Suppose, for induction, that we are typing an important variable z which causes the importance of y_1, y_2, ..., y_n, which have been given the types α_{y_1}, α_{y_2}, ..., α_{y_n}. By induction using invariant 3, by taking the max of their complexities, we can assume that they can all be typed with the same type, η. Therefore, the terms that these variables abstract all look like

$$\lambda y_i w_1 \dots w_k \bullet P_i : \eta \to \phi_i.$$

for some types ϕ_i. Then type

$$z : \forall c \bullet (\eta \to c) \to \bot.$$

Note that z can accept any of these for input. Further note that this type of z can be done without affecting the types of any of the important variables already decided (after altering their complexity, of course), which we were obligated to show from the base case where the term P was an important variable.

All unimportant variables still have type $\bot$, the type of any important variables still satisfies invariant 2, and as we may increase the complexity of η arbitrarily, we can increase the complexity of the type of z arbitrarily, preserving the 3rd invariant. □

It is not true that requiring a bound leftmost path is equivalent to the term being self-applicable in system F. Consider the normal term

$$M \equiv \lambda x \bullet x(\lambda uvw \bullet vI)(\lambda y \bullet y\omega\omega)$$

The above typing construction gives us a typing of this term with a free leftmost path since there are no two important variables applied. However, MM is not strongly normalizing.

The problem of determining if a term is self-applicable, or if ωM has a type for a normal term M is actually undecidable.

Theorem 13 (Gunther, Statman)**.** *The problem of deciding, given a normal term M if MM has a type in system F is undecidable.*

Proof. We will reduce this problem to deciding if a term has a type in system F, which is undecidable by a result of Wells [Wel99]. Let N be an arbitrary term. Every redex in N, $\ldots(\lambda x.P)Q\ldots$ can be transformed as $\ldots z(\lambda x.P)Q\ldots$, to create a new term $N^\star$. Note that $N^\star\mathsf{I} \twoheadrightarrow_\beta N$.

Consider the normal term

$$M \equiv \lambda x.x(\lambda uvw.v\mathsf{I})N^\star.$$

We have

$$MM \to_\beta M(\lambda uvw.v\mathsf{I})N^\star \to_\beta (\lambda uvw.v\mathsf{I})(\lambda uvw.v\mathsf{I})N^\star N^\star \to_\beta N^\star\mathsf{I} \to_\beta N.$$

Therefore, if MM has a type then N has a type. It's only left to show that if N has a type then MM has a type. Let $N : \alpha$, and let σ_I denote the polymorphic type for I in system F: $\forall a.a \to a$.

First note, if $N : \alpha$ then we can type $N^\star : \sigma_\mathsf{I} \to \alpha$. Then we can assign the following type to M:

$$M : \Big(\forall a.a \to (\sigma_\mathsf{I} \to \alpha) \to (\sigma_\mathsf{I} \to \alpha) \to \alpha\Big) \to (\sigma_\mathsf{I} \to \alpha) \to \alpha.$$

Similarly, we can type M as

$$M : \Big(\big(\forall a.a \to (\sigma_\mathsf{I} \to \alpha) \to (\sigma_\mathsf{I} \to \alpha) \to \alpha\big) \to (\sigma_\mathsf{I} \to \alpha) \to \alpha\Big) \to \alpha,$$

since we can type

$$\lambda uvw.v\mathsf{I} : \forall a.a \to (\sigma_\mathsf{I} \to \alpha) \to (\sigma_\mathsf{I} \to \alpha) \to \alpha.$$

So, MM is properly typed of type α. □

Theorem 14 (Gunther, Statman)**.** *The problem of deciding, given a normal term M if ωM has a type in system F is undecidable.*

Proof. As above, we reduce this problem to typability of an arbitrary term in system F. Let N be an arbitrary term, and again consider $N^\star : \sigma_{\mathsf{I}} \to \alpha$. Use the same M has above. As before, it's clear that if ωM has a type then N has a type.

Now, type the x in M as

$$x : \big(a \to (\sigma_{\mathsf{I}} \to \alpha) \to (\sigma_{\mathsf{I}} \to \alpha)\big) \to (\sigma_{\mathsf{I}} \to \alpha) \to (\sigma_{\mathsf{I}} \to \alpha)$$

and

$$\lambda uvw.v\mathsf{I} : a \to (\sigma_{\mathsf{I}} \to \alpha) \to \sigma_{\mathsf{I}} \to \alpha.$$

Then

$$M : \Big(\big(a \to (\sigma_{\mathsf{I}} \to \alpha) \to (\sigma_{\mathsf{I}} \to \alpha)\big) \to (\sigma_{\mathsf{I}} \to \alpha) \to (\sigma_{\mathsf{I}} \to \alpha)\Big) \to \sigma_{\mathsf{I}} \to \alpha$$

□

4.5 Conclusion and future work

In this chapter we explored a particular class of normal terms which are typable in system F with a free leftmost path. We did this in the hope that it gives us some insights in constructing counterexamples to the set of typable terms in system F being closed under ω-expansions. Further, we proved some relevant problems are undecidable.

For future work, it would be nice to extend this result to non-normal terms. This is not straightforward as it seems that we'd need some kind of generalization of important variables, or another approach all together. We might also look at the original question in higher-order parametric polymorphic systems (e.g. F_ω).

Bibliography

[And63] Peter Andrews. A reduction of the axioms for the theory of propositional types. *Fundamenta Mathematicae*, 52(3):345–350, 1963. 27

[And72] Peter B. Andrews. General models, descriptions, and choice in type theory. *Journal of Symbolic Logic*, 37(2):385–394, 1972. iii

[And02] Peter B Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, volume 27. Springer Science & Business Media, 2002. iii, 27, 29

[Bar84] Hendrik Pieter Barendregt. *The lambda calculus*, volume 3. North-Holland Amsterdam, 1984. 1, 2

[BDS13] Hendrik Pieter Barendregt, Wil Dekkers, and Richard Statman. *Lambda calculus with types*. Cambridge University Press, 2013. 1, 9, 17

[CFC74] Haskell B Curry, Robert Feys, and William Craig. *Combinatory Logic*, volume 1. North-Holland, 1974. 8

[Chu32] Alonzo Church. A set of postulates for the foundation of logic. *Annals of mathematics*, pages 346–366, 1932. 1, 6

[Chu36a] Alonzo Church. A note on the entscheidungsproblem. *The journal of symbolic logic*, 1(01):40–41, 1936. 29

[Chu36b] Alonzo Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, pages 345–363, 1936. 1, 4, 29

[Chu40] Alonzo Church. A formulation of the simple theory of types. *The journal of symbolic logic*, 5(02):56–68, 1940. 6, 27

[Chu41] Alonzo Church. *The calculi of lambda-conversion*. Princeton University Press, 1941. 6, 13

[CR36] Alonzo Church and J Barkley Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, 1936. 4

[Cur34] Haskell B Curry. Functionality in combinatory logic. *Proceedings of the National Academy of Sciences of the United States of America*, 20(11):584, 1934. 8

[Cur41] Haskell B Curry. Consistency and completeness of the theory of combinators. *The Journal of Symbolic Logic*, 6(02):54–61, 1941. 5

[Fri75] Harvey Friedman. Equality between functionals. In *Logic Colloquium*, pages 22–37. Springer, 1975. 10

[Gab74] Dov M Gabbay. On 2nd order intuitionistic propositional calculus with full comprehension. *Archive for Mathematical Logic*, 16(3):177–186, 1974. 37

[Gab81] Dov M Gabbay. *Semantical investigations in Heyting's intuitionistic logic*, volume 148. Taylor & Francis, 1981. 37

[Gan80] Robin O Gandy. Proofs of strong normalization. In Haskell B Curry, J Roger Hindley, and Jonathan Paul Seldin, editors, *To HB Curry: essays on combinatory logic, lambda calculus, and formalism*, pages 457–477. Academic Press, 1980. 8

[GDR88] Paola Giannini and Simona Ronchi Della Rocca. Characterization of typings in polymorphic type discipline. In *Logic in Computer Science, 1988. LICS'88., Proceedings of the Third Annual Symposium on*, pages 61–70. IEEE, 1988. 37

[Gen35] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935. 8

[Gir72] Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique dordre supérieur*. PhD thesis, Université Paris VII, 1972. iii, 36

[GS14] William Gunther and Richard Statman. Reflections on a theorem of Henkin. In *The Life and Work of Leon Henkin*, pages 203–216. Springer, 2014. 24

[GTL89] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*, volume 7. Cambridge University Press Cambridge, 1989. 35

[Hen63] Leon Henkin. A theory of prepositional types. *Fundamenta Mathematicae*, 52(3):323–344, 1963. 15, 27

[Hin69] Roger Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the american mathematical society*, pages 29–60, 1969. 9

[How80] William A Howard. The formulae-as-types notion of construction. In Haskell B Curry, J Roger Hindley, and Jonathan Paul Seldin, editors, *To HB Curry: essays on combinatory logic, lambda calculus, and formalism*, pages 479–490. Academic Press, 1980. 8

[Kle35] Stephen Cole Kleene. A theory of positive integers in formal logic. part I. *American journal of mathematics*, pages 153–173, 1935. 6

[KR35] Stephen C Kleene and J Barkley Rosser. The inconsistency of certain formal logics. *Annals of Mathematics*, pages 630–636, 1935. 1

[Läu70] Hans Läuchli. An abstract notion of realizability for which intuitionistic predicate calculus is complete. *Studies in Logic and the Foundations of Mathematics*, 60:227–234, 1970. 23

[Löb76] Martin H Löb. Embedding first order predicate logic in fragments of intuitionistic logic. *Journal of Symbolic Logic*, pages 705–718, 1976. 37

[Mil78] Robin Milner. A theory of type polymorphism in programming. *Journal of computer and system sciences*, 17(3):348–375, 1978. 9

[Pfe88] Frank Pfenning. Partial polymorphic type inference and higher-order unification. In *Proceedings of the 1988 ACM conference on LISP and functional programming*, pages 153–163. ACM, 1988. 36

[Pra71] Dag Prawitz. Ideas and results in proof theory. *Studies in Logic and the Foundations of Mathematics*, 63:235–307, 1971. 37

[Rey74] John C Reynolds. Towards a theory of type structure. In *Programming Symposium*, pages 408–425. Springer, 1974. iii, 36

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965. 9

[Sta82] Richard Statman. Completeness, invariance and lambda-definability. *Journal of Symbolic Logic*, 47(1):17–26, 1982. 17

[Sta00] Richard Statman. Church's lambda delta calculus. In *Logic for Programming and Automated Reasoning*, pages 293–307. Springer, 2000. 13, 14

[SU06] Morten Heine Sørensen and Pawel Urzyczyn. *Lectures on the Curry-Howard isomorphism*, volume 149. Elsevier, 2006. 35

[Tai67] William W Tait. Intensional interpretations of functionals of finite type I. *The Journal of Symbolic Logic*, 32(02):198–212, 1967. 8

[Tur36] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *J. of Math*, 58(345-363):5, 1936. 29

[Tur37] Alan M Turing. Computability and λ-definability. *The Journal of Symbolic Logic*, 2(04):153–163, 1937. 6

[vBD01] Johan van Benthem and Kees Doets. Higher-order logic. *Handbook of Philosophical Logic*, 1:189–244, 2001. 17

[Wel99] Joe B Wells. Typability and type checking in system F are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1):111–156, 1999. 37, 46

Index