The Calculi of Lambda-Conversion by Alonzo Church Annotated Notes and Ramblings of a Tired Student

Prepared by Ryan Flannery

Contents

0 Pre-Introduction

What follows is a brief introduction to set theory,

How does set theory do this? Simple. Consider the following constructions...

0.1 Numbers

In set theory, we posit the existence of the empty set, denoted $\emptyset$, and use this to build a representation of the natural numbers. 0 (zero), is appropriately represented by $\emptyset$, and every natural number n is the set of all *previous* natural numbers. So...

$$\begin{array}{rcll}
\text{Natural Number} & & \text{Set Representation} & (0.1.1)\\
0_{set} & = & \emptyset & (0.1.2)\\
1_{set} & = & \{\emptyset\} & (0.1.3)\\
2_{set} & = & \{\emptyset, \{\emptyset\}\} & (0.1.4)\\
3_{set} & = & \{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\} & (0.1.5)\\
 & \vdots & & (0.1.6)\\
n_{set} & = & \{0_{\text{set}}, 1_{\text{set}}, \ldots, n-1_{\text{set}}\} & (0.1.7)
\end{array}$$

0.2 Ordered Lists (Tuples)

TUPLES!

0.3 Relations

Relations, also, are represented by sets.

$$\leq = \{(a,b) \mid a \text{ is less than or equal to } b\} \tag{0.3.1}$$

The binary relation "$\leq$" would then be represented by the following set, where all tuples of natural numbers would be replaced with *their* set representations.

$$\begin{array}{rll}
\leq_{set} = \{ & (0,0), (0,1), (0,2), (0,3), \ldots & (0.3.2)\\
 & (1,1), (1,2), (1,3), (1,4), \ldots & (0.3.3)\\
 & (2,2), (2,3), (2,4), (2,5), \ldots & (0.3.4)\\
\vdots & & (0.3.5)\\
\} & \} & (0.3.6)
\end{array}$$

Any statement of the form $x \leq y$ could then be interpretted as $(x,y) \in \leq$.

0.4 Functions

Similarly, we can represent *functions* as sets. An n-ary function can be represented as an $n+1$-ary relation. Look a the well known binary function $+$ (plus), which is usually written in the form $a + b = c$. We can *represent* this as a ternary relation,

$$+_{set} = \{(a,b,c) \mid \text{the sum of } a \text{ and } b \text{ is equal to } c\} \tag{0.4.1}$$

The set representation of $+$ is then the following set.

$$\begin{array}{rll}
+_{set} = \{ & (0,0,0), (0,1,1), (0,2,2), (0,3,3), \ldots & (0.4.2)\\
 & (1,0,1), (1,1,2), (1,2,3), (1,3,4), \ldots & (0.4.3)\\
 & (2,0,2), (2,1,3), (2,2,4), (2,3,5), \ldots & (0.4.4)\\
\vdots & & (0.4.5)\\
 & \} & (0.4.6)
\end{array}$$

0.5 Who cares?

Thus, in set theory, everytime we talk about numbers, ordered lists, relations, and functions, we're *really* talking about sets. *Everything is a set*, and the basic "thing" we talk about, and construct more complicated "things" from, is a set.

Although this representation maybe easy to work with, it's not necessarily intuitive, especially when it comes to functions. Alonzo Church, the creator of Lambda calculus, thought that conceptually, functions are more like *actions*, where given a function f and some input, and the function performs some action (which is specied in the very definition of f) until an output is produced. Alonzo Church created Lambda calculus as a system of mathematics that reflects this approach. As an extreme opposite to set theory, the basic "thing" in Lambda calculus *is the function*, and all other things (including numbers, ordered lists, relations, and sets) are represented through *functions*.

This representation of things may seem just as abstract and unintuitive as the representation used in set theory, but it has a certain appeal when working with algorithms, where answering

1 Introduction

1.1 Functions in Lambda Calculus

The underlying concept of Lambda Calculus (from here denoted by λ-Calculus) is the *function.* Although not a new concept entirely, it is how λ-Calculus treats functions that distinguishes it from other formal systems.

Definition 1.1.1 (Function)**.** A *function* is a rule of correspondence by which when anything is given (as *argument*) another thing (the *value* of the function for *that* argument) may be obtained. More simply, a function is an *action* that is applied to one thing (the argument) to obtain another thing (the value). A function need not be applicable to everything; a function may only make sense for a certain set of arguments, called the *range of arguments.* The set of all values obtained from a applying all possible arguments to a function is called the *range of values.*

Unlike set theory, where functions are treated as sets of ordered tuples, in λ-Calculus functions are treated as *actions*, where applying the function to something (arguments) yields something else (a value). In pure λ-Calculus, everything is be treated as a function. Even numbers, as we'll soon see, are functions.

Notation 1.1.2. Let f denote a function. Then $(f\alpha)$ denotes the value of f for the argument α.
We also adopt the following shorthand: $(f\alpha) = f\alpha$.

Note that the treatment of functions in λ-Calculus (as an action) allows a function to be applied to itself. That is, for a function f, there is nothing preventing f from being in its own range.

Two fundamental function in λ-Calculus are defined as follows. . .

Definition 1.1.3 ($\boldsymbol{I}$, the identity function)**.** $\boldsymbol{I}$ is defined as follows: $(\boldsymbol{I}x)$ is x, whatever x may be.

Definition 1.1.4 ($\boldsymbol{H}$)**.** $\boldsymbol{H}$ is defined as the function that always returns the identity function. That is, $(\boldsymbol{H}x)$ is $\boldsymbol{I}$ for whatever x may be.

1.2 Notions of Equality

The concept of a function in λ-Calculus allows the notion of equality between functions to be defined in two separate manners. The first deals solely with the result of the action performed by a function and the second deals with how that action is performed.

Definition 1.2.1 (Extensional Equality)**.** Two functions, f and g, are said to be in *extension* if they have the same range of arguments and for every element α that belongs to this range, $(f\alpha)$ is the same as $(g\alpha)$.

Let $f = x + 1$ and $g = (x + 3) - 2$. Clearly, for every possible numerical argument α, $(f\alpha) = (g\alpha)$. Yet the manner in which f and g reach their value is slightly different. This brings us to the next notion of equality. . .

Definition 1.2.2 (Intensional Equality)**.** Two functions are said to be functions in *intension* if they are extensionally equivalent and identical in the manner by which they produce some value provided some argument(s).

1.3 Functions of Several Variables

Functions in λ-Calculus can take more than one argument, however the meaning is interpreted slightly different from our previous notions of functions. In λ-Calculus, a function of $n+1$ arguments is treated as a function of one argument whose values are functions of (up to) n arguments.

For example, let f be a function of two arguments. Then $((fa)b)$ is how we would interpret providing two arguments, a and b to f.

Notation 1.3.1. We adopt the following shorthand: $((fa)b) = (fab) = fab$.

Notice the following: let f denote a function of 3 variables. Then $fabc$ denotes the value of f for the arguments a, b, and c. Given our interpretation of functions, note that fa is a function whose value is a function of two variables, fab is a function whose value is a function of one variable, and $fabc$ denotes an actual value (potentially another function of any number of arguments). In this respect, we may provide as many arguments as we wish to any function!

Definition 1.3.2 ($\boldsymbol{K}$, the constancy function)**.** $\boldsymbol{K}$ is defined such that Kxy is x for whatever x and y may be.

Note that $\boldsymbol{KII}$ is $\boldsymbol{I}$ and $\boldsymbol{KHI}$ is $\boldsymbol{H}$. More interestingly, note that $\boldsymbol{KI}$ is $\boldsymbol{H}$ and that $\boldsymbol{KK}$ is, simply, $\boldsymbol{K}$.

Next, we define the following functions...

Definition 1.3.3 ($\boldsymbol{T}$)**.** $\boldsymbol{T}xf$ is (fx)

Definition 1.3.4 ($\boldsymbol{J}$)**.** $\boldsymbol{J}fxyz$ is $fx(fzy)$

Definition 1.3.5 ($\boldsymbol{B}$, the product function)**.** $\boldsymbol{B}fgx$ is $f(gx)$

Definition 1.3.6 ($\boldsymbol{C}$, the converse function)**.** $\boldsymbol{C}fxy$ is (fyx)

Definition 1.3.7 ($\boldsymbol{D}$)**.** $\boldsymbol{D}x$ is (xx)

Definition 1.3.8 ($\boldsymbol{W}$)**.** $\boldsymbol{W}fx$ is (fxx)

Definition 1.3.9 ($\boldsymbol{S}$)**.** $\boldsymbol{S}nfx$ is $f(nfx)$
NOTE: This is distinctly different from the function $\boldsymbol{S}$ defined in Barendregt's text. There, $\boldsymbol{S}fgx$ is $fx(gx)$.

1.4 Abstraction

Notation 1.4.1. Let $\boldsymbol{M}$ denote an *expression* containing a variable x that is free (that is, the meaning of $\boldsymbol{M}$ depends upon the value of x). Then $(\lambda x\boldsymbol{M})$ denotes a *function* whose value for some argument α is denoted by substituting all occurrences of α in $\boldsymbol{M}$.
Notice that if $\boldsymbol{M}$ does not contain the symbol x, then the value of $(\lambda x\boldsymbol{M})$ is constant and equal to $\boldsymbol{M}$.
Also, although x is free in $\boldsymbol{M}$, it is *bound* in $(\lambda x\boldsymbol{M})$ and that λx is an incomplete expression! (λx has no meaning on its own.

In the above, note the distinction between an expression and a function. Take the following two expressions, $\boldsymbol{M} = (x+x)$ and $\boldsymbol{N} = (y+y)$. The equation $\boldsymbol{M} = \boldsymbol{N}$ then expresses a relation between the numbers denoted by x and y, and its truth depends on determining x and y.
Now look at $(\lambda x\boldsymbol{M}) = (\lambda y\boldsymbol{N})$:

$$(\lambda x(x+x)) = (\lambda y(y+y)) \tag{1.4.1}$$

This equation denotes something entirely different; it says that the two functions $(\lambda x\boldsymbol{M})$ and $(\lambda y\boldsymbol{N})$ are equivalent (which is true) and makes no mention of the values of x or y.

When we added the λx to the expression $\boldsymbol{M}$, we created a function. This operation is called *abstraction* and the symbol λx is called the *abstract operator*.

Notation 1.4.2. The expression $(\lambda x(\lambda y\boldsymbol{M}))$ is abbreviated $(\lambda xy.\boldsymbol{M})$ and denotes a function whose value for the argument x is denoted by $(\lambda y\boldsymbol{M})$.

2 Lambda-Conversion

2.1 Primitive Symbols and Formulas

Now we are ready to lay out the language of λ-Calculus. It consists of the following primitive symbols called *improper symbols*:

$$\lambda \text{ (the abstraction operator), } (,\ )$$

and an infinite list of *variables* which will use the English alphabet:

$$a, b, c, \ldots, x, y, z, \overline{a}, \overline{b}, \ldots, \overline{\overline{a}}, \ldots$$

Definition 2.1.1 (Formula)**.** A formula is any finite sequence of primitive symbols.

Definition 2.1.2 (Well-Formed Formula)**.** A well-formed formula is any formula where all variables occur either free or bound, according to the following rules:

1. A variable x is a well-formed formula and the occurrence of x in this formula (itself) is free.
2. If $\boldsymbol{M}$ and $\boldsymbol{N}$ are well-formed, then $(\boldsymbol{MN})$ is also well-formed. Any variables occurring free (or bound) in $\boldsymbol{M}$ or $\boldsymbol{N}$ are considered free (or bound) in $(\boldsymbol{MN})$.
3. If $\boldsymbol{M}$ is well-formed and contains the free variable x, then $(\lambda x\boldsymbol{M})$ is also well-formed and all occurrences of x in $(\lambda x\boldsymbol{M})$ are bound.

Notation 2.1.3. In λ-Calculus, bold capital letters ($\boldsymbol{M, N, A, B, \ldots}$) to stand for formulas, and bold small letters ($\boldsymbol{a, b, c, \ldots}$) to stand for variables.
Unless otherwise indicated in a particular case, all formulas are assumed to be well-formed.

We now introduce notation that we will use when the concept of substitution comes in...

Notation 2.1.4. Let $S^x_{\boldsymbol{N}}\boldsymbol{M}|$ denote the formula that results when all occurrences of x in $\boldsymbol{M}$ are replaced with $\boldsymbol{N}$.
This is equivalent to Barendregt's syntax: $\boldsymbol{M}[x := \boldsymbol{N}]$. Since Barendregt's syntax is significantly easier and faster, it shall be used henceforth in these notes.

The $\rightarrow$ may be read as "stands for" (among other things, as we'll see later) and is used to expand formulas, replacing some sub-term with the meaning of that sub-term. For example, the function $\boldsymbol{I}$ defined above is represented formally by the function λaa, so...

$$\boldsymbol{I} \rightarrow (\lambda aa) \tag{2.1.1}$$

And, accordingly,

$$(\boldsymbol{II}) \rightarrow ((\lambda aa)(\lambda aa)) \tag{2.1.2}$$

We now make the following definitions:

Definition 2.1.5.

$$\begin{aligned}
[\boldsymbol{M}+\boldsymbol{N}] &\rightarrow (\lambda a(\lambda b((\boldsymbol{M}a)((\boldsymbol{N}a)b)))) && (2.1.3)\\
[\boldsymbol{M}\times\boldsymbol{N}] &\rightarrow (\lambda a(\boldsymbol{M}(\boldsymbol{N}))) && (2.1.4)\\
[\boldsymbol{M}^{\boldsymbol{N}}] &\rightarrow (\boldsymbol{NM}) && (2.1.5)
\end{aligned}$$

Notation 2.1.6. For convenience, the parentheses will often be omitted for large formulas. We should keep in mind when replacing the parentheses that grouping in λ-Calculus is *left-associative.* That is, $\boldsymbol{MNS}$ is really $((\boldsymbol{MN})\boldsymbol{S})$.
Similarly for expressions with brackets, $x + y + z$ is really $[[x + y] + z]$.

One bit of confusion that may occur when evaluating expressions in λ-Calculus is in formulas of the form $(\lambda x\boldsymbol{M})$. If we let $\boldsymbol{M} = xyz$, then we have $(\lambda xxyz)$. This is ambiguous; where does the abstraction operator stop and the expression begin? To solve this, we use the ".".

Notation 2.1.7. When omitting a set of parentheses or brackets may cause ambiguity, use a "." in place of the left parenthesis or bracket. It is to be interpreted as follows: replace the "." with a left parentheses and insert a right parenthesis as far the right as possible. So, for example: $(\lambda x.\boldsymbol{MN}) \rightarrow (\lambda x(\boldsymbol{MN}))$, $(\lambda xy.\boldsymbol{MN}) \rightarrow (\lambda x(\lambda y(\boldsymbol{MN})))$, and $(\lambda x.\lambda y.\boldsymbol{MN}) \rightarrow (\lambda x(\lambda y(\boldsymbol{MN})))$

Now that the formal language of λ-Calculus has been established, here are the formal definitions for the functions defined thus far:

$$\boldsymbol{I} \rightarrow \lambda a.a \tag{2.1.6}$$
$$\boldsymbol{H} \rightarrow \lambda a.I \tag{2.1.7}$$
$$\boldsymbol{T} \rightarrow \lambda xf.fx \tag{2.1.8}$$
$$\boldsymbol{J} \rightarrow \lambda fxyz.fx(fzy) \tag{2.1.9}$$
$$\boldsymbol{B} \rightarrow \lambda fgx.f(gx) \tag{2.1.10}$$
$$\boldsymbol{C} \rightarrow \lambda fxy.fyx \tag{2.1.11}$$
$$\boldsymbol{D} \rightarrow \lambda x.xx \tag{2.1.12}$$
$$\boldsymbol{W} \rightarrow \lambda fx.fxx \tag{2.1.13}$$
$$\boldsymbol{S} \rightarrow \lambda nfx.f(nfx) \tag{2.1.14}$$

2.2 Conversion

We now introduce the fundamental operations on well-formed formulas in λ-Calculus: conversion. Although there are many forms of conversion, these are the three we begin with:

I Given a formula, we can replace any part $\boldsymbol{M}$ with $\boldsymbol{M}[x := y]$ as long as x is not a free variable of $\boldsymbol{M}$ and y does not occur in $\boldsymbol{M}$. Notice, all we are doing here is changing variable names.
This method of conversion is often called **α-conversion**.

II Given a formula, we can replace any part $((\lambda x\boldsymbol{M})\boldsymbol{N})$ with $\boldsymbol{M}[x := \boldsymbol{N}]$ as long as the bound variables of $\boldsymbol{M}$ are distinct both from x and from the free variables of $\boldsymbol{N}$. Application of this rule is called Contraction.
This method of conversion is often called **β-conversion**.

III Given a formula, we can replace any part $\boldsymbol{M}[x := \boldsymbol{N}]$ with $((\lambda x\boldsymbol{M})\boldsymbol{N})$ as long as $((\lambda x\boldsymbol{M})\boldsymbol{N})$ is well-formed and the bound variables of $\boldsymbol{N}$ are distinct from both x and the free variables of $\boldsymbol{N}$. *NOTE: where the previous rule provides a sort of reduction, this rule provides the opposite: expansion.*

Definition 2.2.1. A *part* of a formula is any consecutive well-formed part that does not immediately follow an occurrence of the symbol λ.

Notice, Rule I can convert $ab(\lambda aa)(\lambda aa)$ into $ab(\lambda bb)(\lambda aa)$, and Rule III can convert (λaa) into $(\lambda a.(\lambda aa)a)$.
Rules I-III are *definite*, in that given any two formulas $\boldsymbol{M}$ and $\boldsymbol{N}$, we can determine if $\boldsymbol{N}$ can be derived from $\boldsymbol{M}$ using one of the rules and, if so, which one.

Definition 2.2.2 (Immediately Convertible)**.** If $\boldsymbol{M}$ can be converted into $\boldsymbol{N}$ by the application of one of rules I-III then we say $\boldsymbol{M}$ *immediately converts* to $\boldsymbol{N}$, written $\boldsymbol{M}$ *imc* $\boldsymbol{N}$, or $\boldsymbol{M} \rightarrow \boldsymbol{N}$.

Definition 2.2.3 (Convertible)**.** If $\boldsymbol{M}$ can be converted into $\boldsymbol{N}$ by the application of a finite number of rules I-III then we say $\boldsymbol{M}$ *converts* to $\boldsymbol{N}$, written $\boldsymbol{M}$ *conv* $\boldsymbol{N}$, or $\boldsymbol{M} \twoheadrightarrow \boldsymbol{N}$.

Definition 2.2.4 (Interconvertible)**.** If $\boldsymbol{M} \twoheadrightarrow \boldsymbol{N}$, then we say $\boldsymbol{M}$ and $\boldsymbol{N}$ are interconvertible.

Notice that conversion (as defined thus far) is transitive, symmetric, and reflexive (proof omitted).

Definition 2.2.5 (Expansion)**.** A conversion that contains no applications of Rule II and exactly one application os Rule III is called expansion.

Definition 2.2.6 (Reduction)**.** A conversion that contains no applications of Rule III and exactly one application os Rule II is called reduction. Further, we say that $\boldsymbol{M}$ *reduces* to $\boldsymbol{N}$ ($\boldsymbol{M}$ *red* $\boldsymbol{N}$ for short) if, by a finite number of reductions, $\boldsymbol{M}$ converts to $\boldsymbol{N}$.

Definition 2.2.7 (Normal Form)**.** A well-formed formula is said to be in normal-form if it contains no part of the form $((\lambda x\boldsymbol{M})\boldsymbol{N})$.

Definition 2.2.8 (Principle Normal Form)**.** A well-formed formula is said to be in principle normal-form if it is in normal form and the bound variables occur in alphabetical ordering from left to right. Note that any formula in normal form can thus be converted into principle normal form by a finite number of applications of rule I.

2.3 (Selections from) Fundamental Theorems on Well-Formed Formulas and on the Normal Form

Below is only a few of the theorems provided in §7 of Church's text.

Theorem 2.3.1. *If $\boldsymbol{M}$ is well-formed and $\boldsymbol{M} \twoheadrightarrow \boldsymbol{N}$ then $\boldsymbol{N}$ is well-formed.*

Theorem 2.3.2. *If $\boldsymbol{M}$ is well-formed and $\boldsymbol{M} \twoheadrightarrow \boldsymbol{N}$ then $\boldsymbol{M}$ and $\boldsymbol{N}$ have the same free variables.*

Just as variables occur either free or bound in a formula, we define similar notions for parts of formula. A well-formed part $\boldsymbol{P}$ of a formula $\boldsymbol{K}$ is a free occurrence of $\boldsymbol{P}$ in $\boldsymbol{K}$ if every free occurrence of a variable in $\boldsymbol{P}$ is also a free occurrence of that variable in $\boldsymbol{K}$. Otherwise, $\boldsymbol{P}$ is bound in $\boldsymbol{K}$.

Further, we extend the syntax $\boldsymbol{M}[x := \boldsymbol{N}]$ to handle this new notion of free/bound parts.

Notation 2.3.3. Let $\boldsymbol{M}[\boldsymbol{N} := \boldsymbol{P}]$ denote the formula that results when all occurrences of $\boldsymbol{N}$ in $\boldsymbol{M}$ are replaced with $\boldsymbol{P}$.

Theorem 2.3.4. *If $\boldsymbol{M} \twoheadrightarrow \boldsymbol{N}$ then there is a conversion of $\boldsymbol{M}$ into $\boldsymbol{N}$ in which no expansion precedes any reduction.*

Theorem 2.3.5. *If $\boldsymbol{N}$ is a normal form of $\boldsymbol{M}$, then there is a sequence of conversions from $\boldsymbol{M}$ into $\boldsymbol{N}$ using only rules I and II (reduction).*

Theorem 2.3.6. *If $\boldsymbol{M}$ has a normal form, its normal form is unique within applications of rule I.*

Theorem 2.3.7. *If $\boldsymbol{M}$ has a normal form, it has a unique principle normal form. Notice that once a normal form is reached, a finite number of α-conversions will produce a formula in principle normal form.*

Theorem 2.3.8. *If $\boldsymbol{N}$ is a normal form of $\boldsymbol{M}$, then there is a number m such that any sequence of reductions starting from $\boldsymbol{M}$ will lead to $\boldsymbol{N}$ (within applications of rule I) after at most m reductions.*
Basically, if $\boldsymbol{N}$ has a normal form then there is an upper bound, m, on the number of reductions taken to obtain the normal form.

Theorem 2.3.9. *If $\boldsymbol{M}$ has a normal form, every well-formed part of $\boldsymbol{M}$ has a normal form.*

3 Lambda-Definability

3.1 Lambda-Definability of Functions of Positive Integers

In λ-Calculus, everything is a function. Even numbers. The natural numbers (1, 2, 3, ...) are represented in the following fashion:

Definition 3.1.1 (Natural Numbers)**.** We define the natural numbers as follows:

$$1 \rightarrow \lambda ab.ab \tag{3.1.1}$$
$$2 \rightarrow \lambda ab.a(ab) \tag{3.1.2}$$
$$3 \rightarrow \lambda ab.a(a(ab)) \tag{3.1.3}$$
$$\vdots$$

Notice, any number x can be defined as the function that takes two arguments, f and y, and it applies f x-times to y.

To avoid confusion, numbers of multiple digits will have a bar over them to distinguish them from sequences of single-digit numbers. For example, we will use $\overline{11}$ to denote eleven and 11 to denote the function 1 applied to itself.

Definition 3.1.2 (λ-Definable)**.** A function F of n arguments, all of which are natural numbers, is λ-definable if there is a formula $\boldsymbol{F}$ such that the following requirements are met:

1. If $m_1, m_2, \ldots, m_n, l$ are positive integers and $Fm_1m_2 \ldots m_n = l$, and $\boldsymbol{M}_1, \boldsymbol{M}_2 \ldots \boldsymbol{M}_n, \boldsymbol{L}$ represent the integers $m_1, m_2, \ldots, m_n, l$ respectively, then $\boldsymbol{F}\boldsymbol{M}_1\boldsymbol{M}_2 \ldots \boldsymbol{M}_n \twoheadrightarrow \boldsymbol{L}$
2. If the function F has no value for the positive integers $m_1, m_2, \ldots, m_n$ as arguments, and $\boldsymbol{M}_1, \boldsymbol{M}_2, \ldots \boldsymbol{M}_n$ represent $m_1, m_2, \ldots, m_n$ respectively, then $\boldsymbol{F}\boldsymbol{M}_1\boldsymbol{M}_2 \ldots \boldsymbol{M}_n$ has no normal form.

Church then provides the following λ-definitions for common functions:

$$\begin{aligned} \textit{Successor}: &\quad \lambda abc.b(abc) && (3.1.4)\\ \textit{Addition}: &\quad \lambda ab.a+b && (3.1.5)\\ \textit{Multiplication}: &\quad \lambda ab.a\times b && (3.1.6)\\ \textit{Exponentiation}: &\quad \lambda ab.a^b && (3.1.7)\end{aligned}$$

Where "+", "×", and exponentiation are to be interpreted as defined in Definition 2.1.5.

For the rest of Church's text, the successor function is abbreviated using $\boldsymbol{S}$ and is not to be confused with the previous definition for $\boldsymbol{S}$. How we are not to confuse the two functions is beyond me, however it seems that in almost all contexts for the rest of the text, $\boldsymbol{S}$ will mean "successor".

3.2 Ordered Pairs & Triads and the Predecessor Function

It is often useful to work with ordered groups of values. To do so in λ-Calculus, we introduce the following notation for ordered pairs and triads. It's easy to see how to extend the syntax for ordered n-tuples.

Notation 3.2.1.

$$\begin{aligned} [\boldsymbol{M},\boldsymbol{N}] &\rightarrow \lambda a.a\boldsymbol{MN} && (3.2.1)\\ [\boldsymbol{L},\boldsymbol{M},\boldsymbol{N}] &\rightarrow \lambda a.a\boldsymbol{LMN} && (3.2.2)\end{aligned}$$

Church then defines the following functions which extract particular elements from an ordered pair/triad:

Definition 3.2.2. For pairs, we have...

$$\begin{aligned} 2_1 &\rightarrow \lambda a.a(\lambda bc.c\boldsymbol{I}b) && (3.2.3)\\ 2_2 &\rightarrow \lambda a.a(\lambda bc.b\boldsymbol{I}c) && (3.2.4)\end{aligned}$$

For triads, we have...

$$\begin{aligned} 3_1 &\rightarrow \lambda a.a(\lambda bcd.c\boldsymbol{I}d\boldsymbol{I}b) && (3.2.5)\\ 3_2 &\rightarrow \lambda a.a(\lambda bcd.b\boldsymbol{I}d\boldsymbol{I}c) && (3.2.6)\\ 3_3 &\rightarrow \lambda a.a(\lambda bcd.d\boldsymbol{I}c\boldsymbol{I}d) && (3.2.7)\end{aligned}$$

So, for example, $2_1[\boldsymbol{M},\boldsymbol{N}] \twoheadrightarrow \boldsymbol{M}$, $2_2[\boldsymbol{M},\boldsymbol{N}] \twoheadrightarrow \boldsymbol{N}$, $3_1[\boldsymbol{L},\boldsymbol{M},\boldsymbol{N}] \twoheadrightarrow \boldsymbol{L}$, $3_2[\boldsymbol{L},\boldsymbol{M},\boldsymbol{N}] \twoheadrightarrow \boldsymbol{M}$, and $3_3[\boldsymbol{L},\boldsymbol{M},\boldsymbol{N}] \twoheadrightarrow \boldsymbol{N}$.

The predecessor function is then defined as follows:

Definition 3.2.3 (Predecessor Function)**.** For any integer n other than 1, Pn has a value of $n-1$. For 1, $P1$ has a value of 1.

$$\boldsymbol{P} \rightarrow \lambda a.3_3(a(\lambda b[\boldsymbol{S}(3_1b),\ 3_1b,\ 3_2b])[1,\ 1,\ 1]) \qquad (3.2.8)$$

Using the predecessor function, a form of subtraction can be defined:

Definition 3.2.4 (Subtraction)**.** For any two integers a and b, $a \dot{-} b$ has a value of 1 if $a < b$ and $a - b$ otherwise.

$$[\boldsymbol{M} \dot{-} \boldsymbol{N}] \rightarrow \boldsymbol{NPM} \qquad (3.2.9)$$

Using the predecessor function, we can then define the typical *min* and *max* functions as follows:

Definition 3.2.5 (Min/Max Functions)**.** The functions that return the greatest of two positive integers and the lesser of two positive integers are defined as follows:

$$\begin{aligned} \mathit{min} &\rightarrow \lambda ab.\boldsymbol{S} \dot{-} .Sb \dot{-} a && (3.2.10)\\ \mathit{max} &\rightarrow \lambda ab.[a+b] \dot{-} \mathit{min}\ ab && (3.2.11)\end{aligned}$$

Later, we'll make use of the following few functions:

Definition 3.2.6 (Z and Z')**.** To define Z and Z', we first need the following:

$$\mathfrak{L} \rightarrow \lambda\beta.\beta(\lambda c\beta\lambda d[dPc(\lambda e.e1\boldsymbol{I})(\lambda fg.fg5)c,\ dPc(\lambda h.h1\boldsymbol{I}5)(\lambda ijk.kij(\lambda l.l1))d]) \tag{3.2.12}$$
$$\mathfrak{U} \rightarrow \lambda\alpha.\alpha\mathfrak{L}[1,\ 1] \tag{3.2.13}$$

Now, we may define Z and Z' as follows:

$$Z \rightarrow \lambda\alpha.2_2(\mathfrak{U}\alpha) \tag{3.2.14}$$
$$Z' \rightarrow \lambda\alpha.\mathfrak{U}\alpha(\lambda\beta c.b\dot{-}c) \tag{3.2.15}$$

Notice the behavior of Z and Z' with integer arguments...

$$\begin{array}{ll} Z1 \twoheadrightarrow 1 & Z'1 \twoheadrightarrow 1 \\ Z2 \twoheadrightarrow 1 & Z'2 \twoheadrightarrow 2 \\ Z3 \twoheadrightarrow 2 & Z'3 \twoheadrightarrow 1 \\ Z4 \twoheadrightarrow 1 & Z'4 \twoheadrightarrow 3 \\ Z5 \twoheadrightarrow 2 & Z'5 \twoheadrightarrow 2 \\ Z6 \twoheadrightarrow 3 & Z'6 \twoheadrightarrow 1 \\ Z7 \twoheadrightarrow 1 & Z'7 \twoheadrightarrow 4 \end{array}$$

In this sense, we can use Z and Z' prime to enumerate the natural numbers. Indeed, the infinite sequence of ordered pairs:

$$[Z1,\ Z'1],\ [Z2,\ Z'2],\ [Z3,\ Z'3], \ldots \tag{3.2.16}$$

contains all ordered pairs of positive integers with no repetitions.

The parity function is defined as follows:

Definition 3.2.7 (Parity Function)**.** The parity of a positive integer is 1 if the integer is odd and 2 if it is even. It is defined by the following function:

$$par \rightarrow \lambda a.a(\lambda b.3 \dot{-} b)2 \tag{3.2.17}$$

3.3 Propositional Functions

Definition 3.3.1 (Propositional Function)**.** Any function whose values are truth values (true / false).

Definition 3.3.2 (Property)**.** A propositional function of one variable.

Definition 3.3.3 (Relation)**.** A propositional function of two variables.

In pure λ-Calculus, we do not have such primitives as "true" and "false" (all we have are functions). So, to work with propositional functions, we have the notion of a characteristic function:

Definition 3.3.4 (Characteristic Function)**.** The characteristic function associated with a propositional function is the function whose value is 2 (the λ-function) when the value of the propositional function is true, whose value is 1 when the propositional function is false, and which has no value otherwise.

Note that the choice of 2 and 1 in the above definition is arbitrary. A propositional function of natural numbers is said to be λ-definable if the associated characteristic function is λ-definable. Some examples of λ-definable characteristic functions follow:

Definition 3.3.5 (Strict Greater-Than and Equality for Natural Numbers)**.**

$$exc \rightarrow \lambda ab.min\ 2\ [Sa \dot{-} b] \tag{3.3.1}$$
$$eq \rightarrow \lambda ab.4 \dot{-}\ .\ exc\ ab + exc\ ba \tag{3.3.2}$$

Let R be a λ-definable propositional function of $n+1$ integer arguments. In the case that R has a value for every set of arguments, the function F can be described by saying that $Fx_1x_2\ldots x_n$ is the least positive integer y such that $Rx_1x_2\ldots x_ny$ holds.

Let $\mathfrak{G}$ be defined as follows:

Definition 3.3.6 ($\mathfrak{G}$)**.**

$$\mathfrak{G} \rightarrow \lambda n.n(\lambda r.r(\lambda s.s1\boldsymbol{II}(\lambda xgt.g1(tx)\boldsymbol{I}x)))(\lambda f.f\boldsymbol{I}1\boldsymbol{II})(\lambda xgt.g(t(\boldsymbol{S}x))(\boldsymbol{S}x)gt) \tag{3.3.3}$$

(Quite a mouthful!) Next, notice the following...

$$\begin{array}{lcl} \mathfrak{G}1\boldsymbol{N}\mathfrak{G}\boldsymbol{T} & red & \mathfrak{G}(\boldsymbol{T}(\boldsymbol{SN}))(\boldsymbol{SN})\mathfrak{G}\boldsymbol{T} \end{array} \tag{3.3.4}$$
$$\begin{array}{lcl} \mathfrak{G}2\boldsymbol{N}\mathfrak{G}\boldsymbol{T} & red & \boldsymbol{N} \end{array} \tag{3.3.5}$$

NOTE: In the above two equations, $\boldsymbol{S}$ is the successor function. Using $\mathfrak{G}$, we can define the function $\mathfrak{p}$ as follows:

Definition 3.3.7 ($\mathfrak{p}$, the Kleene $\mathfrak{p}$ function)**.**

$$\mathfrak{p} \rightarrow \lambda tx.\mathfrak{G}(tx)x\mathfrak{G}t \tag{3.3.6}$$

Note that if $\boldsymbol{N}$ represents a positive integer and $\boldsymbol{TN}$ converts to either 1 or 2 (i.e. $\boldsymbol{T}$ is simply a characteristic function), then $\mathfrak{p}\boldsymbol{TN}$ *red* $\boldsymbol{N}$ if $\boldsymbol{TN} \twoheadrightarrow 2$ and $\mathfrak{p}\boldsymbol{TN} \twoheadrightarrow \mathfrak{p}\boldsymbol{T}(\boldsymbol{SN})$ if $\boldsymbol{TN} \twoheadrightarrow 1$. Further, if $\boldsymbol{TN}$ has no normal form then $\mathfrak{p}\boldsymbol{TN}$ has no normal form.

At first glance, the $\mathfrak{G}$ and $\mathfrak{p}$ functions seem too much to comprehend. A closer look at what they do, however, makes them much easier to understand. The $\mathfrak{p}$ function is a "generic" version of the two equations in 3.3.4 and 3.3.5. That is, for a given characteristic function $\boldsymbol{T}$ and a natural number $\boldsymbol{N}$, $\mathfrak{p}\boldsymbol{TN}$ will reduce to either equation 3.3.4 or 3.3.5, depending on what $\boldsymbol{TN}$ reduces to (remember, $\boldsymbol{T}$ is a characteristic function, so it always reduces to either 1 or 2). Equation 3.3.5 always reduces to simply $\boldsymbol{N}$. Equation 3.3.4, however (and this is where it gets interesting) reduces to, essentially, $\mathfrak{p}\boldsymbol{T}(\boldsymbol{N}+1)$! It effectively increases $\boldsymbol{N}$ and starts over!

The Kleene function, then, provides a convenient way of working with propositional functions on the natural numbers. Say $\boldsymbol{N}$ represents some positive integer n and $\boldsymbol{T}$ λ-defines the characteristic function associated with some property T of positive integers. $\mathfrak{p}\boldsymbol{TN}$ is then convertible into the formula that represents the least positive y, not less than n, such that $\boldsymbol{T}y$ holds (given that such a y exists).

The Kleene function can be used to provide a more intuitive definition for division and subtraction, which we shall denote by $-$ as opposed to the previous $\dot{-}$.

Definition 3.3.8.

$$[\boldsymbol{M} - \boldsymbol{N}] \quad \rightarrow \quad \mathfrak{p}(\lambda a.eq\ \boldsymbol{M}\ [\boldsymbol{N} + a])1 \tag{3.3.7}$$
$$[\boldsymbol{M} \div \boldsymbol{N}] \quad \rightarrow \quad \mathfrak{p}(\lambda a.eq\ \boldsymbol{M}\ [\boldsymbol{N} \times a])1 \tag{3.3.8}$$

Note that $x - y$ has no value if $x < y$ and $x \div y$ has no value unless x is a positive integer multiple of y.

3.4 Definition by Recursion

Previously, the *composition* of two functions, $(f \circ g)\overline{x}$, was defined as $f(g\overline{x}))$. Now, the notion of composition is extended for any number of functions.

Definition 3.4.1 (Composition)**.** A function F of n positive integer arguments is said to be defined by composition in terms of functions G and $H_1, H_2, \ldots, H_m$ of positive integers if F is defined as follows:

$$Fx_1x_2\ldots x_n = G(H_1x_1x_2\ldots x_n)(H_2x_1x_2\ldots x_n)\ldots(H_mx_1x_2\ldots x_n) \tag{3.4.1}$$

Notice that if $n = m = 1$, then we have $Fx = G(Hx)$, which corresponds to our previous notion of composition for two functions.

Definition 3.4.2 (Primitive Recursion)**.** A function F of $n+1$ positive integer arguments is said to be defined by primitive recursion in terms of functions G_1 and G_2 of positive integers if F is defined as follows:

$$Fx_1x_2\ldots x_n1 \quad = \quad G_1x_1x_2\ldots x_n \tag{3.4.2}$$
$$Fx_1x_2\ldots x_n(y+1) \quad = \quad G_2x_1x_2\ldots x_ny(Fx_1x_2\ldots x_ny) \tag{3.4.3}$$

Note that if we allow $n = 0$, then G_1 is simply some constant positive integer α.

Using this notion of Primitive Recursion, the class of all primitive recursive functions of positive integers can now be defined.

Definition 3.4.3 (Primitive Recursive Functions)**.** We use the following criteria to define all primitive recursive functions:

1. Any function F that is equivalent to one of the following 3 functions is primitive recursive:
   (a) The function C such that $Cx = 1$ for every positive integer x
   (b) The successor function of positive integers
   (c) The function U_i^n, where n and i are any positive integers and $i \leq n$, such that $U_i^n x_1 x_2 \dots x_n = x_i$ are primitive recursive.
2. If the function F of n arguments is defined by composition in terms of $G, H_1, \dots, H_m$, and if $G, H_1, \dots, H_m$ are primitive recursive, then F is primitive recursive.
3. If the function F of $n+1$ arguments is defined by primitive recursion in terms of functions G_1 and G_2, and if G_1 and G_2 are primitive recursive, then F is primitive recursive.

The class of primitive functions includes all of the common numerical functions, including determining the greatest common divisor, determining quotient and remainder in integer division, and finding the x^{th} prime number.

Theorem 3.4.4. *Every primitive recursive function of positive integers is λ-definable.*

Now, we have two methods for defining functions: composition and primitive recursion. For both, any particular value of a function F (defined by either method) can be found by a finite number of reductions. By keeping this requirement, we can extend the notion of recursion in the following fashions.

Definition 3.4.5 (General Recursion)**.** A function F is generally recursive if it meets the requirement (1) of the primitive recursive functions and if all defined functions composing F have a value for every set of integer arguments.

and...

Definition 3.4.6 (Partial Recursion)**.** A function F is partially recursive if it only meets the first requirement of the primitive recursive functions.

Theorem 3.4.7. *Every partial recursive function of positive integers is λ-definable.*

Theorem 3.4.8. *Every λ-definable function of positive integers is partially recursive.*

Recursion is a powerful notion in λ-Calculus, as it is elsewhere, and as proven above the notion of recursion is equivalent to many other notions. Other equivalent notions are *effective calculability*, *computability*, and the notion of a *finite combinatory process*. Some of these notions are defined below.

Definition 3.4.9 (Effectively Calculable Function of Positive Integers)**.** A function F is effectively calculable if there exists a method of calculating the value of F given any set of positive integer arguments.
Note that this definition is equivalent to λ-definability! It is simply a more intuitive definition of λ-definability.

Definition 3.4.10 (Computable)**.** A function F is computable (roughly speaking) if it is possible to make a finite calculating machine capable of computing any required value of the function (Turing).

So, what functions aren't recursive? Equivalently, what functions aren't effectively calculable? An example of such of function follows. If the set of well-formed formulas of λ-Calculus is enumerated in a straightforward way, and if F is the function such that F is 2 if the x^{th} formula has a normal form and 1 if it does not, then F is not λ-definable. This leads to the conclusion that the condition of having a normal form is not sufficient for effective calculability.

4 Combinations & Gödel Numbers

4.1 Combinations

Definition 4.1.1 (S-Combinations)**.** Let s be any set of well-formed formulas. A formula is called an s-combination if it is defined by the following two rules:

1. Any formula of the set s, and any variable standing alone is an s-combination.
2. If $\boldsymbol{A}$ and $\boldsymbol{B}$ are s-combinations, then $\boldsymbol{AB}$ is an s-combination.

The set of all s-combinations is called the "class of s-combinations".

Church notes that for the cases we are interested in, the formulas of s will not contain free variables and none will be of the form $\boldsymbol{AB}$. In this case, any *term* of an s-combination will be either a free variable or one of the formulas of s. If $s = \emptyset$, then the s-combinations will be call *combinations of variables.*

If s consists of the two formulas $\boldsymbol{I}$ and $\boldsymbol{J}$ defined as follows:

$$\boldsymbol{I} \rightarrow \lambda a.a \tag{4.1.1}$$
$$\boldsymbol{J} \rightarrow \lambda abcd.ab(adc) \tag{4.1.2}$$

then the s-combinations will simply be called ***combinations***. *NOTE: These definitions of $\boldsymbol{I}$ and $\boldsymbol{J}$ are the same as presented earlier*

A function that will prove useful when dealing with combinations is τ, defined as follows:

Definition 4.1.2 (τ)**.** Let...

$$\tau \rightarrow \boldsymbol{JII} \tag{4.1.3}$$

And notice that $\tau \twoheadrightarrow \lambda ab.ba$, or, $\tau\boldsymbol{AB} \twoheadrightarrow \boldsymbol{BA}$.

If $\boldsymbol{M}$ is any combination containing x as a free variable, then we define an associated combination $\lambda_x\boldsymbol{M}|$, which does not contain x as a free variable, as follows:

1. $\lambda_x x|$ is $\boldsymbol{I}$
2. If $\boldsymbol{B}$ contains x as a free variable and $\boldsymbol{A}$ does not, then $\lambda_x\boldsymbol{AB}|$ is $\boldsymbol{J}\tau\lambda_x\boldsymbol{B}|(\boldsymbol{JIA})$
3. If $\boldsymbol{A}$ contains x as a free variable and $\boldsymbol{B}$ does not, then $\lambda_x\boldsymbol{AB}|$ is $\boldsymbol{J}\tau\boldsymbol{B}\lambda_x\boldsymbol{A}|$
4. If both $\boldsymbol{A}$ and $\boldsymbol{B}$ contain x as a free variable, then $\lambda_x\boldsymbol{AB}|$ is $\boldsymbol{J}\tau\tau(\boldsymbol{JI}(\boldsymbol{J}\tau\tau(\boldsymbol{J}\tau\lambda_x\boldsymbol{B}|(\boldsymbol{J}\tau\lambda_x\boldsymbol{A}|\boldsymbol{J}))))$

The next theorem is easily seen from the construction presented above

Theorem 4.1.3. *If $\boldsymbol{M}$ is a combination containing x as a free variable, then $\lambda_x\boldsymbol{M}| \twoheadrightarrow \lambda x\boldsymbol{M}$.*

Now, we can define the notion of *combinations that belong to a well- formed formula...*

Definition 4.1.4 (Combinations Belonging to a Well-Formed Formula)**.** 1. The combination belonging to a variable x is x.

2. The combination belonging to $\boldsymbol{FA}$ is $\boldsymbol{F}'\boldsymbol{A}'$, where $\boldsymbol{F}'$ and $\boldsymbol{A}'$ are the combinations belonging to $\boldsymbol{F}$ and $\boldsymbol{A}$ respectively.
3. The combination belonging to $\lambda x\boldsymbol{M}$ is $\lambda_x\boldsymbol{M}'|$, where $\boldsymbol{M}'$ is the combination belonging to $\boldsymbol{M}$.

The following theorem is key when dealing with combinations...

Theorem 4.1.5. *Every well-formed formula is convertible into a combination belonging to it.*

Theorem 4.1.6. *The combinations belonging to $\boldsymbol{M}$ and the combinations belonging to $\boldsymbol{N}$ are identical if and only if $\boldsymbol{M} \twoheadrightarrow \boldsymbol{N}$ using only Rule (1).*

4.2 Primitive Sets of Formulas

Definition 4.2.1 (Primitive Set)**.** A set s of well-formed formulas is called a primitive set if the following conditions are met...

1. The formulas of s contain no free variables.
2. None of the formulas are of the form $\boldsymbol{AB}$.
3. Every well-formed formula of s is convertible into an s-combination.

Examples of primitive sets include the set $\boldsymbol{I}, \boldsymbol{J}$, $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{W}, \boldsymbol{I}$, and $\boldsymbol{B}, \boldsymbol{T}, \boldsymbol{D}, \boldsymbol{I}$. One way of showing that a set of formulas, say s, is primitive is to express all of the formulas of another primitive set in terms of only the formulas in s.

Definition 4.2.2 (Primitive Independence)**.** A primitive set of formulas is said to be independent if it ceases to be a primitive set upon omission of any one of the formulas.

Notice that any set containing both $\boldsymbol{I}$ and $\boldsymbol{J}$ is a primitive recursive set as long as the other formulas in the set meet the above requirements.

4.3 An Application of the Theory of Combinations

Theorem 4.3.1. *If $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ contain no free variables then a formula $\boldsymbol{L}$ can be found such that $\boldsymbol{L}1 \twoheadrightarrow \boldsymbol{A}_1$ and $\boldsymbol{L}2 \twoheadrightarrow \boldsymbol{A}_2$.*

This theorem can be generalized into the following through induction...

Theorem 4.3.2. *If $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_n$ contain no free variables, a formula $\boldsymbol{L}$ can be found such that $\boldsymbol{L}1 \twoheadrightarrow \boldsymbol{A}_1$, $\boldsymbol{L}2 \twoheadrightarrow \boldsymbol{A}_2$,..., $\boldsymbol{L}\boldsymbol{N} \twoheadrightarrow \boldsymbol{A}_n$ (where $\boldsymbol{N}$ is the formula representing the positive integer n).*

What follows is an example construction of such a formula for $\boldsymbol{A}_1$,..., $\boldsymbol{A}_5$, such that $\boldsymbol{Q}1 \twoheadrightarrow \boldsymbol{A}_1$, $\boldsymbol{Q}2 \twoheadrightarrow \boldsymbol{A}_2$,..., $\boldsymbol{Q}5 \twoheadrightarrow \boldsymbol{A}_5$. Define $\boldsymbol{Q}$ as follows:

$$\boldsymbol{Q} \rightarrow \lambda i.\boldsymbol{G}_1[3 \dot{-} 1](Pi) \tag{4.3.1}$$
$$\boldsymbol{Q}_2 \rightarrow \lambda i.\boldsymbol{G}_2[3 \dot{-} 1](Pi) \tag{4.3.2}$$
$$\boldsymbol{Q}_3 \rightarrow \lambda i.\boldsymbol{G}_3[3 \dot{-} 1](Pi) \tag{4.3.3}$$

Where

$$\boldsymbol{G}_1 \rightarrow \lambda n.n(\lambda x.x(\lambda y.y\boldsymbol{I}\boldsymbol{B}_2'))(\lambda z.z\boldsymbol{I}\boldsymbol{I})\boldsymbol{B}_1'\boldsymbol{J} \tag{4.3.4}$$
$$\boldsymbol{G}_2 \rightarrow \lambda n.n(\lambda x.x(\lambda y.y\boldsymbol{I}\boldsymbol{B}_2''))(\lambda z.z\boldsymbol{I}\boldsymbol{I})\boldsymbol{B}_1''\boldsymbol{J} \tag{4.3.5}$$
$$\boldsymbol{G}_3 \rightarrow \lambda n.n(\lambda x.x(\lambda y.y\boldsymbol{I}\boldsymbol{B}_2'''))(\lambda z.z\boldsymbol{I}\boldsymbol{I})\boldsymbol{B}_1'''\boldsymbol{J} \tag{4.3.6}$$

and

$$\begin{array}{ccccc}
\boldsymbol{B}_1'\boldsymbol{I} \rightarrow \boldsymbol{I} & \boldsymbol{B}_1'\boldsymbol{J} \rightarrow \boldsymbol{Q}_2 & \boldsymbol{B}_2'\boldsymbol{I} \rightarrow \boldsymbol{I} & \boldsymbol{B}_2'\boldsymbol{J} \rightarrow \boldsymbol{L}_2' & \boldsymbol{L}_2'1 \rightarrow \boldsymbol{A}_1 \\
\boldsymbol{B}_1''\boldsymbol{I} \rightarrow \boldsymbol{I} & \boldsymbol{B}_1''\boldsymbol{J} \rightarrow \boldsymbol{Q}_3 & \boldsymbol{B}_2''\boldsymbol{I} \rightarrow \boldsymbol{I} & \boldsymbol{B}_2''\boldsymbol{J} \rightarrow \boldsymbol{L}_2'' & \boldsymbol{L}_2''1 \rightarrow \boldsymbol{A}_2 \\
\boldsymbol{B}_1'''\boldsymbol{I} \rightarrow \boldsymbol{I} & \boldsymbol{B}_1'''\boldsymbol{J} \rightarrow \boldsymbol{L}_1''' & \boldsymbol{B}_2'''\boldsymbol{I} \rightarrow \boldsymbol{I} & \boldsymbol{B}_2'''\boldsymbol{J} \rightarrow \boldsymbol{L}_2''' & \boldsymbol{L}_2'''1 \rightarrow \boldsymbol{A}_3 \\
 & \boldsymbol{L}_1'''1 \rightarrow \boldsymbol{A}_4 & & \boldsymbol{L}_1'''2 \rightarrow \boldsymbol{A}_5 &
\end{array}$$

Now, notice the following:
$\boldsymbol{Q}1 \rightarrow (\boldsymbol{G}_1 2)1 \rightarrow (\boldsymbol{B}_2'\boldsymbol{J})1 \rightarrow \boldsymbol{L}_2'1 \rightarrow \boldsymbol{A}_1$
$\boldsymbol{Q}2 \rightarrow (\boldsymbol{G}_1 1)1 \rightarrow (\boldsymbol{B}_1'\boldsymbol{J})1 \rightarrow \boldsymbol{Q}_2 1 \rightarrow (\boldsymbol{G}_2 2)1 \rightarrow (\boldsymbol{B}_2''\boldsymbol{J})1 \rightarrow \boldsymbol{L}_2''1 \rightarrow \boldsymbol{A}_2$
$\boldsymbol{Q}3 \rightarrow (\boldsymbol{G}_1 1)2 \rightarrow (\boldsymbol{B}_1'\boldsymbol{J})2 \rightarrow \boldsymbol{Q}_2 2 \rightarrow (\boldsymbol{G}_2 1)1 \rightarrow (\boldsymbol{B}_1''\boldsymbol{J})1 \rightarrow \boldsymbol{Q}_3 1 \rightarrow (\boldsymbol{G}_3 2)1 \rightarrow (\boldsymbol{B}_2'''\boldsymbol{J})1 \rightarrow \boldsymbol{L}_2'''1 \rightarrow \boldsymbol{A}_3$
$\boldsymbol{Q}4 \rightarrow (\boldsymbol{G}_1 1)3 \rightarrow (\boldsymbol{B}_1'\boldsymbol{J})3 \rightarrow \boldsymbol{Q}_2 3 \rightarrow (\boldsymbol{G}_2 1)2 \rightarrow (\boldsymbol{B}_1''\boldsymbol{J})2 \rightarrow \boldsymbol{Q}_3 2 \rightarrow (\boldsymbol{G}_3 1)1 \rightarrow (\boldsymbol{B}_1'''\boldsymbol{J})1 \rightarrow \boldsymbol{L}_1'''1 \rightarrow \boldsymbol{A}_4$
$\boldsymbol{Q}5 \rightarrow (\boldsymbol{G}_1 1)4 \rightarrow (\boldsymbol{B}_1'\boldsymbol{J})4 \rightarrow \boldsymbol{Q}_2 4 \rightarrow (\boldsymbol{G}_2 1)3 \rightarrow (\boldsymbol{B}_1''\boldsymbol{J})3 \rightarrow \boldsymbol{Q}_3 3 \rightarrow (\boldsymbol{G}_3 1)2 \rightarrow (\boldsymbol{B}_1'''\boldsymbol{J})2 \rightarrow \boldsymbol{L}_1'''2 \rightarrow \boldsymbol{A}_5$

The notion developed above can be used to create an enumeration of formulas in λ-Calculus, as follows...

Theorem 4.3.3. *If $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_n, \boldsymbol{F}_1, \boldsymbol{F}_2, \ldots, \boldsymbol{F}_m$ contain no free variables, a formula $\boldsymbol{E}$ can be found which represents an enumeration of the least set of formulas which contains $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_n$ and is closed under each of the operations of the form $\boldsymbol{F}_\alpha\boldsymbol{X}\boldsymbol{Y}$, from the formulas $\boldsymbol{X}$ and $\boldsymbol{Y}$. This enumeration occurs when every formula in the set is convertible into one of the formulas in the infinite sequence $\boldsymbol{E}1, \boldsymbol{E}2, \ldots$* and *every formula in this infinite sequence is convertible into one of the formulas of the set.*

4.4 A Combinatory Equivalent of Conversion

Until now, any work that has been done with combinations first required us to convert the combination into a formula of λ-Calculus, work with it, and then convert it back into a combination. What we do now is develop a set of operations that act on combinations and return combinations, thus allowing us to work only with combinations. Many of these operations mimic similar operations in λ-Calculus.

To start with, we define the following...

Definition 4.4.1 (γ, β, and ω)**.**

$$\gamma \rightarrow J\tau(J\tau)(J\tau) \tag{4.4.1}$$
$$\beta \rightarrow \gamma(J\boldsymbol{I}\gamma)(J\boldsymbol{I}) \tag{4.4.2}$$
$$\omega \rightarrow \gamma(\gamma(\beta\gamma(\gamma(\beta J\tau)\tau))\tau) \tag{4.4.3}$$

Notice the following: $\tau \twoheadrightarrow \boldsymbol{T}$, $\gamma \twoheadrightarrow \boldsymbol{C}$, $\beta \twoheadrightarrow \boldsymbol{B}$ and $\omega \twoheadrightarrow \boldsymbol{W}$ (where $\boldsymbol{T}$, $\boldsymbol{C}$, $\boldsymbol{B}$, and $\boldsymbol{W}$ were defined previously).

Church then lists the 38 operations that may be performed on combinations, numbered in Roman numerals (note that all of these Roman numerals are preceded with a leading "0"). When using these operations on combinations it must always be cited what operation is being performed.

Notation 4.4.2 ($\vdash$)**.** When dealing with operations on combinations, we make use of the "$\vdash$", and interpret it as follows. For two combinations, α and β, $\alpha \vdash \beta$ means that α is changed into β by the operation cited.

The 38 operations (together) that Church lists can be proven equivalent to λ-conversion. The difference? The combination operations are much simpler than rules I, II, and III of §6, in the following ways:

1. The combination operations are one-valued. That is, given the combination operated on and the particular operation begin performed, the resulting combination is uniquely determined.
2. The combination operations do not do substitution at arbitrary places. Rather, the combination operations perform substitution only at specified places.

4.5 Gödel Numbers

Definition 4.5.1 (Gödel number of a combination)**.** The Gödel number of a combination is defined by induction as follows:

1. The Gödel number of I is 1
2. The Gödel number of J is 3
3. The Gödel number of the n^{th} variable in alphabetical order is $2n + 5$
4. If m and n are the Gödel numbers of $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively, then the Gödel number of $\boldsymbol{AB}$ is $(m + n)(m + n - 1) - 2n + 2$ *NOTE: this is always an even integer!*

Definition 4.5.2 (Gödel number of a formula)**.** The Gödel number of a formula is defined to be the Gödel number of the combination belonging to the formula. *Note: The Gödel number belonging to a combination is thus not necessarily the same as the Gödel number of the combination itself.*

The Gödel numbers of two combinations $\boldsymbol{A}$ and $\boldsymbol{B}$ are the same if and only if $\boldsymbol{A}$ and $\boldsymbol{B}$ are the same. The Gödel numbers belonging to two formulas $\boldsymbol{A}$ and $\boldsymbol{B}$ are the same if and only if $\boldsymbol{A} \twoheadrightarrow \boldsymbol{B}$ using only α-conversion.

Notation 4.5.3. In the following discussion, subscripts of formulas will be used extensively, and they are to be interpreted as follows. Let

$$\alpha_1 \rightarrow \boldsymbol{AB}\alpha \tag{4.5.1}$$
$$\alpha_2 \rightarrow \boldsymbol{CD}\alpha \tag{4.5.2}$$

where $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are arbitrary, and the form of α_1 and α_2 are arbitrary. All that is required is that an α appear in their expansion.
We may, then, iterate the use of subscripts, to obtains something such as

$$\alpha_{122} \rightarrow \boldsymbol{CD}(\boldsymbol{CD}(\boldsymbol{AB}\alpha)) \tag{4.5.3}$$

Note that the left-most subscript is the inner-most term, and the right- most subscript is the outer-most term.

In using Gödel numbers, we'll make use of a formula "form" defined such that if $\boldsymbol{N}$ represents the Gödel number belonging to formula $\boldsymbol{A}$, and $\boldsymbol{A}$ contains no free variables, then form $\boldsymbol{N} \twoheadrightarrow \boldsymbol{A}$. To derive this formula, note the following steps:

1. Notice that $par\boldsymbol{N} \twoheadrightarrow 2$ (meaning $\boldsymbol{N}$ is even) if $\boldsymbol{N}$ represents the Gödel number of combination having more than one term.
2. If $\boldsymbol{N}$ represents the Gödel number of combination $\boldsymbol{AB}$, then $\boldsymbol{Z}(\boldsymbol{HN})$ is convertible into the formula representing the Gödel number of $\boldsymbol{A}$, and $\boldsymbol{Z}'(\boldsymbol{HN})$ is convertible into the formula representing the Gödel number of $\boldsymbol{B}$. To simplify, we introduce the following:

$$\boldsymbol{N}_1 \rightarrow \boldsymbol{Z}(\boldsymbol{HN}) \tag{4.5.4}$$
$$\boldsymbol{N}_2 \rightarrow \boldsymbol{Z}'(\boldsymbol{HN}) \tag{4.5.5}$$

3. Using the method described in §14, we find a formula $\mathfrak{B}$ such that:

$$\begin{aligned} \mathfrak{B}1 &\rightarrow \lambda x.x12 && (4.5.6)\\ \mathfrak{B}2 &\rightarrow \boldsymbol{I} && (4.5.7)\\ \mathfrak{B}3 &\rightarrow \lambda x.x12\boldsymbol{J} && (4.5.8)\end{aligned}$$

4. And, using the same method, we find a formula $\mathfrak{U}$ such that:

$$\begin{aligned} \mathfrak{U}1 &\rightarrow \mathfrak{B} && (4.5.9)\\ \mathfrak{U}2 &\rightarrow \lambda xy.y(par\ x_1)x_1y(y(par\ x_2)x_2y) && (4.5.10)\end{aligned}$$

5. Finally, let "form" be defined as follows:

$$form \rightarrow \lambda n.\mathfrak{U}(par\ n)n\mathfrak{U} \tag{4.5.11}$$

6. Notice the following:

$$\begin{aligned} form\ 1 &\rightarrow \boldsymbol{I} && (4.5.12)\\ form\ 3 &\rightarrow \boldsymbol{J} && (4.5.13)\end{aligned}$$

   and if $\boldsymbol{N}$ represents and even positive integer...

$$form\ \boldsymbol{N} \rightarrow form\ \boldsymbol{N}_1(form\ \boldsymbol{N}_2) \tag{4.5.14}$$

With the formula "form" constructed above, then if $\boldsymbol{N}$ represents the Gödel number for a combination $\boldsymbol{A}'$ belonging to the formula $\boldsymbol{A}$, we have

$$form\ \boldsymbol{N} \twoheadrightarrow A' \twoheadrightarrow A \tag{4.5.15}$$

5 The Calculi of λ-$\boldsymbol{K}$-Conversion and λ-δ-Conversion

5.1 The Calculus of λ-$\boldsymbol{K}$-Conversion

The calculus of λ-$\boldsymbol{K}$-conversion is obtained by slightly altering the calculus of λ-conversion in the following manner. In §5, replace the definition of *well-formed formula* with the following:

Definition 5.1.1 (well-formed formula (λ-$\boldsymbol{K}$-conversion))**.** A well-formed formula is any formula where all variables occur either free or bound, according to the following rules:

1. A variable x is a well-formed formula and the occurrence of x in this formula (itself) is free.
2. If $\boldsymbol{M}$ and $\boldsymbol{N}$ are well-formed, then $(\boldsymbol{MN})$ is also well-formed. Any variables occurring free (or bound) in $\boldsymbol{M}$ or $\boldsymbol{N}$ are considered free (or bound) in $(\boldsymbol{MN})$.
3. If $\boldsymbol{M}$ is well-formed, then $(\lambda x\boldsymbol{M})$ is also well-formed and all occurrences of x in $(\lambda x\boldsymbol{M})$ are bound.

The *only* difference between this definition and the one presented in §2.2 is in rule III. The previous definition required x to occur free at least once in $\boldsymbol{M}$, and this definition makes no such requirement.

The rules of conversion in λ-$\boldsymbol{K}$-conversion are the same as they appear in §2.1, except that we must now take into consideration this new definition of *well-formed formula.*

One of the key differences between the calculi of λ-conversion and λ-$\boldsymbol{K}$-conversion is the possibility of defining the constancy function (this is not possible in λ-conversion since b, although a bounded variable, does not occur in the expression):

Definition 5.1.2 ($\boldsymbol{K}$ (the Constancy Function))**.**

$$\boldsymbol{K} \rightarrow \lambda a(\lambda ba) \tag{5.1.1}$$

Another key advantage in the calculi of λ-$\boldsymbol{K}$-conversion is the existence of the integer zero, defined as follows:

Definition 5.1.3 (0 (Zero))**.**

$$0 \rightarrow \lambda a(\lambda bb) \tag{5.1.2}$$

Many of the theorems in the calculi of λ-conversion still hold in λ-$\boldsymbol{K}$-conversion, however a few fail. One key theorem that still holds, however, is that which asserts the existence of the Kleene $\mathfrak{p}$ function.

Other key similarities are the definitions for the successor function, the addition and multiplication functions, and the predecessor function. The existence of the zero integer, however, can be used to greatly simplify the predecessor function and the Kleene $\mathfrak{p}$ function.

Definition 5.1.4 (Predecessor Function)**.** The following is an example of how we can slightly simplify previously defined functions by making use of 0.

$$\lambda a.2_2(a\lambda b\,[S(2_1b),\ 2_1b])\,[0,\ 0]) \tag{5.1.3}$$

Also, the notion of λ-$\boldsymbol{K}$-definability is analogous to that of λ-definability (as long as we restrict ourselves to the new definition of well-formed formula).

One serious drawback to the calculi of λ-$\boldsymbol{K}$-conversion is that theorem 2.3.9 fails. Without theorem 2.3.9, it is possible for a formula $\boldsymbol{FN}$ to have a normal form but $\boldsymbol{N}$ not. This is, as Church states, clearly unreasonable.

5.2 The Calculus of Restricted λ-$\boldsymbol{K}$-Conversion

In order to avoid the difficulty described above, we can add to Rules II and III in §2.2 the additional requirement that $\boldsymbol{N}$ shall be in normal form. Note that being in normal form is *effective*, but having a normal form is not. Basically, we require up-front that all of the formulas we work with be in normal form. The resulting calculus is called the *calculus of restricted λ-$\boldsymbol{K}$-conversion.*

The benefit of restricted λ-$\boldsymbol{K}$-conversion is that it more closely follows the development of λ-conversion, allowing many more of the theorems (specifically theorem 2.3.9) to remain true.

Many of the theorems developed for λ-conversion have nearly identical analogues in restricted λ-$\boldsymbol{K}$-conversion. Usually, the only difference is the additional premise that the formulas in question be in normal form.

5.3 Transfinite Ordinals

Previously, the positive integers were defined as functions of functions by applying one function to another a finite number of times. This method can be extended to represent the ordinal numbers of the second number class by allowing them to correspond in the same way to the transfinite powers of a function, provided that we first fix upon a limiting process relative to which the transfinite powers are taken.

For example, the ordinal ω could be represented as the function whose value for a function f (as an argument) is the function g such that gx is the limit of the sequence x, fx, $f(fx)$, $f(f(x))$, ... Accordingly, $\omega + 1$ would be $\lambda x.f(\omega fx)$, and so on.

Instead of fixing upon the limiting sequence described above, however, we could make the representation more general if we let the limiting process be an additional argument a. Using this method, we can construct the following formulas in the calculi of restricted λ-$\boldsymbol{K}$-conversion. The "o" subscript is used to distinguish these formulas from previous definitions.

$$\begin{aligned}
0_o &\rightarrow \lambda a(\lambda b(\lambda cc)) && (5.3.1)\\
1_o &\rightarrow \lambda abc.bc && (5.3.2)\\
2_o &\rightarrow \lambda abc.b(bc) && (5.3.3)\\
3_o &\rightarrow \lambda abc.b(b(bc)) && (5.3.4)\\
&\vdots\\
\boldsymbol{S}_o &\rightarrow \lambda dabc.b(dabc) && (5.3.5)\\
\boldsymbol{L}_o &\rightarrow \lambda rabc.a(\lambda d.rdabc) && (5.3.6)\\
\omega_o &\rightarrow \lambda abc.a(\lambda d.dabc) && (5.3.7)
\end{aligned}$$

We designate the following:

- 0_o represents the ordinal 0.
- If $\boldsymbol{N}$ represents the ordinal n, the principal normal form of $S_o\boldsymbol{N}$ represents the ordinal $n + 1$.

- If $\boldsymbol{R}$ represents the strictly increasing infinite sequence of ordinals n_0, n_1, n_2, …, in that $\boldsymbol{R}0_o$, $\boldsymbol{R}1_o$, $\boldsymbol{R}2_o$, … are convertible into formulas representing n_0, n_1, n_2, …, then the principle normal form of $\boldsymbol{L}_o\boldsymbol{R}$ represents the upper limit of this infinite sequence. *NOTE: For a quick example, simply let $\boldsymbol{R} = \boldsymbol{I}$. Then $\boldsymbol{R}$ represents all natural numbers!*

It should be noted that the formula representing a given ordinal of the second class of numbers is by no means unique. For example, ω may be represented not only by ω_o but also by the principle normal form of $\boldsymbol{L}_o\boldsymbol{S}_o$ (and many other formulas). Knowing this, formulas representing ordinals are not to be taken as denoting ordinals but rather as denoting things that are in many-one correspondences with ordinals.

Definition 5.3.1 (λ-$\boldsymbol{K}$-defined)**.** A function F of ordinal numbers is said to be λ-$\boldsymbol{K}$-defined by a formula $\boldsymbol{F}$ if both of the following requirements are met…

1. Whenever $Fm = n$ and $\boldsymbol{M}$ represents m, the formula $\boldsymbol{FM} \twoheadrightarrow \boldsymbol{N}$ where $\boldsymbol{N}$ represents n
2. Whenever an ordinal m is not in the range of F and $\boldsymbol{M}$ represents m, the formula $\boldsymbol{FM}$ has no normal form.

5.4 The Calculus of λ-δ-Conversion

Now, we develop the calculus of λ-δ-conversion. This is done by making the following modifications to the calculus of λ-conversion:

1. Add to the list of primitive symbols a symbol δ, which is neither an improper symbol nor a variable, but is classed with the variables as a *proper symbol.*
2. Add to the rule 1 in the definition of *well-formed formula* that the symbol δ is a well-formed formula.
3. Introduce the notion of δ-normal form as follows:

   Definition 5.4.1 (δ-normal form)**.** A formula is said to be δ-normal form if it contains no part of the form $(\lambda x\boldsymbol{P})\boldsymbol{Q}$ and contains no part of the form $\delta\boldsymbol{RS}$ with $\boldsymbol{R}$ and $\boldsymbol{S}$ containing no free variables.

   NOTE: Both of the conditions of being in δ-normal form and that $\boldsymbol{M}$ is not α-convertible into $\boldsymbol{N}$ are effective.
4. Add the following to the rules of conversion:

   IV To replace any part $\delta\boldsymbol{MN}$ of a formula by 1, provided that $\boldsymbol{M}$ and $\boldsymbol{N}$ are in δ-normal form and contain no free variables and $\boldsymbol{M}$ is not α-convertible into $\boldsymbol{N}$.

   V To replace any part 1 of a formula by $\delta\boldsymbol{MN}$, provided that $\boldsymbol{M}$ and $\boldsymbol{N}$ are in δ-normal form and contain no free variables and $\boldsymbol{M}$ is not α-convertible into $\boldsymbol{N}$.

   VI To replace any part $\delta\boldsymbol{MM}$ of of a formula by 2, provided that $\boldsymbol{M}$ is in δ-normal form and contains no free variables.

   VII To replace any part 2 of a formula by $\delta\boldsymbol{MM}$, provided that $\boldsymbol{M}$ is in δ-normal form and contains no free variables.

Now, some definition analogous to those for λ-conversion are made…

Definition 5.4.2 (λ-δ-conversion)**.** A λ-δ-conversion is any finite sequence of applications of rules I-VII.

Definition 5.4.3 (λ-δ-reduction)**.** A λ-δ-conversion is called a λ-δ-reduction if it contains no application of rule III, V, VII and exactly one application of the rules II, IV, or VI.

Definition 5.4.4 (Immediately Reducible)**.** $\boldsymbol{A}$ is said to be immediately reducible to $\boldsymbol{B}$ if there is a reduction of $\boldsymbol{A}$ into $\boldsymbol{B}$.

Definition 5.4.5 (Reducible)**.** $\boldsymbol{A}$ is said to be reducible to $\boldsymbol{B}$ if there is a conversion of $\boldsymbol{A}$ into $\boldsymbol{B}$ which consists of one or more successive reductions.

One advantage λ-δ-conversion has over λ-$\boldsymbol{K}$-conversion is that *all* of the theorems in the calculi of λ-conversion also hold in the calculi of λ-δ-conversion. This is because the calculi of λ-conversion is actually a subset of the calculi of λ-δ-conversion.

Definition 5.4.6 (λ-δ-conversion)**.** Any finite sequence of Rules I-VII.

Definition 5.4.7 (δ-normal form)**.** A formula will be called a δ-normal form of another if it is in δ-normal form and can be obtained from the other by a finite sequence of λ-δ-conversions.

The calculus of λ-δ-conversion does have one drawback: it requires an intensional interpretation of formulas. Take, for example, the formula $\lambda ab.\delta ab1ab$. This formula corresponds to the same function as 1 in extension, however they are not interchangeable (since $\delta 11 \twoheadrightarrow 2$ but $\delta 1(\lambda ab.\delta ab1ab) \twoheadrightarrow 1$.

Just as we defined the Constancy Function $\boldsymbol{K}$ in the calculus of λ-$\boldsymbol{K}$-conversion, we not introduce a constancy function for λ-δ-conversion.

Definition 5.4.8 (k - the Constancy Function)**.** Define κ to be as follows...

$$k \rightarrow \lambda ab.\delta bb\boldsymbol{I}a \tag{5.4.1}$$

Then, $k\boldsymbol{AB} \twoheadrightarrow \boldsymbol{A}$ if $\boldsymbol{B}$ has a δ-normal form and contains no free variables, and in that case only.

The entire theory of λ-definability of functions of positive integers carries over into the calculus of λ-δ-conversion since the calculus of λ-conversion is contained in that of λ-δ-conversion. All of the theorems of §4 also hold in the calculus of λ-δ-conversion.

The theory of combinations carries over into the calculus of λ-δ-conversion as long as we redefine a combination to mean any $[\boldsymbol{I}, \boldsymbol{J}, \delta]$-combination. In defining the combination belonging to a formula, we must also add the rule that the combination belonging to δ is $\boldsymbol{\delta}$.

To develop a combinatory equivalent of λ-δ-conversion, just as we did in §4.4, we must add the following four operations, where $\boldsymbol{F}$, $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ are combinations, and $\boldsymbol{A}$ and $\boldsymbol{B}$ belong to formulas in δ-normal form containing no free variables, and are not the same. $\boldsymbol{C}$ is the formula which represents the Gödel number $\boldsymbol{A}$:

Rules omitted. See page 66 of Church's text.

Using Gödel numbers in λ-δ-conversion carries over as well, as long as we add the rule that the Gödel number of δ is 5. Accordingly, we add the additional rule that $\mathfrak{B}5 \twoheadrightarrow \lambda x.x12\delta$, insuring that $form\ 5 \twoheadrightarrow \delta$.

It is now possible to define a formula *dncb* which enumerates the Gödel numbers of combinations which belong to formulas in δ-normal form and contain no free variables. *NOTE: the construction of dncb follows that of ncb.*

In λ-δ-conversion, it is also possible to define a formula *met* that acts as the inverse of the function *form*. That is, if $\boldsymbol{M}$ is a formula which contains no free variables and has a δ-normal form, then *met* $\boldsymbol{M}$ is convertible into the formula representing the Gödel number belonging to the δ-normal form of $\boldsymbol{M}$. We define *met* as follows:

Definition 5.4.9 (*met*)**.**

$$met \rightarrow \lambda x.dncb(\mathfrak{p}(\lambda n.\delta(form(dncb\ n))x)1) \tag{5.4.2}$$

5.5 A System of Symbolic Logic

Now, Church uses the calculus of λ-δ-conversion to construct a system of symbolic logic. We retain our use of 2 and 1 to represent truth and falsehood, respectively (knowing, of course, that these choices are arbitrary).

This system has only one axiom, the formula 2 (truth), and seven rules of inference that correspond to rules I-VII in λ-δ-conversion. All provable formulas, or "theorems", in this system are the formulas which can be derived from the formula 2 by sequences of applications of the rules of inference.

In this system, the familiar operations of negation, conjunction, and disjunction may be defined as follows:

Definition 5.5.1 (Negation, Conjunction, & Disjunction)**.** Defined respectively, they are:

$$[\neg\boldsymbol{A}] \rightarrow \pi(\lambda a.a\boldsymbol{I}(\delta 2\boldsymbol{A}))(\lambda a.a\boldsymbol{I}(\delta 1A)) \tag{5.5.1}$$

$$[\boldsymbol{A} \wedge \boldsymbol{B}] \rightarrow 4 \mathbin{\dot{-}} ([\neg\boldsymbol{A}] + [\neg\boldsymbol{B}]) \tag{5.5.2}$$

$$[\boldsymbol{A} \vee \boldsymbol{B}] \rightarrow \neg([\neg\boldsymbol{A}] \wedge [\neg\boldsymbol{B}]) \tag{5.5.3}$$

It follows from these definitions that $\boldsymbol{A} \vee \boldsymbol{B}$ cannot be a theorem unless either $\boldsymbol{A}$ or $\boldsymbol{B}$ is a theorem, and this situation cannot be altered by any suitable change in the definitions. Since this property is known to fail for classical systems of logic such as that of *Principia Mathematica*, it is clear that the present system therefore differs from the classical system in a direction which may be regarded as finitistic in character.

The propositional function *to be a positive integer* is represented in the system as a formula N, defined as follows:

Definition 5.5.2. N

$$N \rightarrow \lambda x.v(met\ x) \tag{5.5.4}$$

The general relation of equality or identity (in intension) is represented by δ. An existential quantifier Σ is defined as follows:

Definition 5.5.3 (ι and Σ)**.**

$$\iota \to \lambda f.form\,(Z'\,(H\,(dcnvt\;\alpha\,(\mathfrak{p}\,(\lambda n.\delta f\,(form\,(Z\,(H\,(dcnvt\;\alpha n)))))\,1)))) \tag{5.5.5}$$

Where α represents the Gödel number belonging to the formula 2. Here, ι acts as a *general selection operator.* Given a formula $\boldsymbol{F}$, if there is any formula $\boldsymbol{A}$ such that $\boldsymbol{FA} \twoheadrightarrow 2$, then $\iota\boldsymbol{F}$ is one such formula. If $\boldsymbol{F}$ has no such formula, then $\iota\boldsymbol{F}$ has no normal form.
NOTE: The equivalence of two propositional functions $\boldsymbol{F}$ and $\boldsymbol{G}$ does not necessarily imply the equivalence of $\iota\boldsymbol{F}$ and $\iota\boldsymbol{G}$. Now, Σ is defined using $\iota\ldots$:

$$\Sigma \to \lambda f.(\iota f) \tag{5.5.6}$$

Thus, $\Sigma\boldsymbol{F} \twoheadrightarrow 2$ if there is a formula $\boldsymbol{A}$ such that $\boldsymbol{FA} \twoheadrightarrow 2$, otherwise $\Sigma\boldsymbol{F}$ has no normal form.

NOTE: the ι operator can be compared with Hilbert's ϵ operator and Hilbert and Bernays η-operator.

Church notes that interpretation of ι and Σ depends on identifying formal provability in this system, which can be justified by the *Completeness Property* of λ-δ-conversion:

Theorem 5.5.4 (Completeness Property)**.** *A formula which is not provable, unless it is convertible into a principal normal form other than 2 (and hence disprovable), must have no normal form, and hence be meaningless.*

For the sake of convenience, we can adopt the following notation...

Notation 5.5.5. We may abbreviate *general selection* and *existential quantification*, respectively, as:

$$[\iota x\boldsymbol{M}] \quad\to\quad \iota(\lambda x\boldsymbol{M}) \tag{5.5.7}$$
$$[\exists x\boldsymbol{M}] \quad\to\quad \Sigma(\lambda x\boldsymbol{M}) \tag{5.5.8}$$

We might be tempted to introduce a *Universal Quantifier* (or simply an Existential Quantifier with Negation), but as Church notes, there are difficulties that arise. Gödel showed that any universal quantifier introduced by definition will have a certain character of incompleteness.

The *consistency* of this system of symbolic logic is actually a corollary of theorem 2.3.7.

One advantage of using this system of symbolic logic over set theory, is that certain paradoxes of set theory fail in this system. For some paradoxes, their failure to transfer into this system of symbolic logic is simply because the formula leading to the paradox has no normal form. For example, with Russell's paradox, the formula

$$(\lambda x.\neg(xx))(\lambda x.\neg(xx)) \tag{5.5.9}$$

has no normal form. In other, more complicated paradoxes, their failure to transfer over may depend on the incompleteness property of quantifiers, mentioned above.