NOTES ON THE λ-CALCULUS

Charles Wells[1]

February 25, 1999

email: charles@abstractmath.org
professional website: http://www.cwru.edu/artsci/math/wells/home.html
blog: http://www.gyregimble.blogspot.com/
abstract math website: http://www.abstractmath.org/MM//MMIntro.htm
*astounding math stories:*http://www.abstractmath.org/MM//MMAstoundingMath.htm

Thanks to Atish Bagchi, Roger Buelow and Duane Sears for finding errors and making suggestions for improvements to these notes.

Chapter 1

The syntax of the λ-calculus

1.1 Definition and notation

A λ-calculus consists of expressions made up of **constants**, **variables** and **λ-terms** defined by a context-free grammar in 1.1 below. These expressions are given a meaning determined by rewrite rules given in Section 2. What distinguishes different λ-calculi are the constants and the rewrite rules governing those constants.

1.1.1 Notation for context-free grammars

Heretofore, in a context-free grammar, we have used uppercase letters for the nonterminals. However, in a λ-calculus, the constants are normally written in uppercase letters. In the grammar below, nonterminals are named in square brackets. Besides allowing uppercase letters to be terminal, this makes it easy to remember what each nonterminal stands for. This notation is commonly used in specifying programming languages and is part of what is commonly called **Backus-Naur notation**.

1.1.2 A context-free grammar for the λ-calculus

$$\begin{aligned}&[\text{term}] \to [\text{constant}] \mid [\text{variable}] \mid \\ &\qquad ([\text{term}][\text{term}]) \mid (\lambda[\text{variable}].[\text{term}]) \\ &[\text{variable}] \to a \mid b \mid \ldots \mid x \mid y \mid z\end{aligned} \tag{1.1}$$

What [constant] stands for depends on the particular λ-calculus. They will normally be written either as sans-serif uppercase letters or as names in brackets. For example, S, K and [CONS] will be constants in particular λ-calculi discussed here.

A λ-calculus is called **pure** if it has no constants. The version of λ-calculus discussed here is untyped (the variables don't have types). An LL(1) grammar for the pure untyped λ-calculus is given in [4]. The *typed* λ-calculus is described in detail in [1] (which is the fundamental reference for λ-calculus) and in [3].

1.1.3 Example

All the following are correct λ-terms.

(xy)

$(\lambda x.(xx))$

$((\lambda x.(xx))(xy)u)$

$((\lambda x.(\lambda y.(yx)))(\lambda x.y)u)$

1.1.4 Definition

In an expression of the form (EF), E is said to be **applied** to F. The operation of applying E to F is called **application**.

1.1.5 Definition

An expression of the form $(\lambda x.E)$ is called a **λ-abstraction** and E is called the **body** of the abstraction.

1.1.6 Conventions

We adopt the following conventions and abbreviations. We indicate an arbitrary term by an italic uppercase letter: For example, E stands for some (possibly unspecified) term but S is a particular constant.

C.1 We normally omit outermost parentheses. For example, we write xy rather than (xy).

C.2 Application associates to the left. This allows us to avoid writing many parentheses. For example, xyz denotes $(xy)z$ (which itself denotes $((xy)z)$ by C.1). On the other hand, the parentheses in $x(yz)$ are compulsory. Similarly $wxyz$ denotes $((wx)y)z$.

C.3 The extent of the body of a λ-abstraction is as large as possible. This follows from the fact that the production for λ in the grammar (1.1) is [term] $\rightarrow (\lambda$[variable].[term]) rather than [term] $\rightarrow (\lambda$[variable].([term])) For example, $\lambda x.xy$ means $\lambda x.(xy)$, not $(\lambda x.x)y$. Note that $\lambda x.xyx$ and $(\lambda x.x)(yx)$ are two different terms; the first is short for $\lambda x.(xy)x$ and the second for $(\lambda x.x)(yx)$.

C.4 For a term E, $\lambda xy.E$ is an abbreviation for $\lambda x.(\lambda y.E)$, $\lambda xyz.E$ is an abbreviation for $\lambda x.(\lambda y.(\lambda z.E))$, and so on.

If E and F represent the same λ-term according to these conventions, we write $E \equiv F$. Thus $\lambda x.xyx \equiv \lambda x.(xy)x$ and $\lambda xy.(xx) \equiv \lambda x.(\lambda y.(xx))$. $E \equiv F$ does not mean merely that E and F *mean* the same thing, it means they are the *same term*. (Meaning is discussed in Chapter 2 below.)

1.1.7 Example

The following are the λ-terms in Example 1.1.3 rewritten according to our conventions.

xy

$\lambda x.xx$

$(\lambda x.xx)xyu$

$(\lambda xy.yx)(\lambda x.y)u$

1.1.8 Terms as functions

It is useful to think informally of a term of the form (EF) as meaning that E is a function evaluated at F. Thus $(xy)(uv)$ means the function xy applied to the term uv, whereas $(xy)uv$ means that xy is applied to u, and the result is applied to v. Thus in a λ-calculus, every term represents a function that can be applied to any other term.

This is not normally the way functions in, for example, calculus class behave. (We know what $\sin\pi$ is, but what is $\sin\sin$?) The formally defined meaning of λ-terms in Chapter 2 is (not surprisingly) rather different from anything you would see in a calculus class, or in most other mathematics classes for that matter.

1.2 Free and Bound

The meaning for λ-terms, given in Section 2 below, is defined in terms of substitution. In general, one is allowed to substitute a term for every occurrence of a free variable in another term. A variable x is free, essentially, if it is anywhere except in the term E in a term of the form $\lambda x.E$. Because of the complications that occur when applying the rewrite rules to be given later, we must give a formal definition of occurring free and occurring bound.

1.2.1 Definition of "free occurrence"

For any variable x,

F.1 The occurrence of x in the term x is free.

F.2 Each free occurrence of x in E and in F is a free occurrence of x in EF.

F.3 If x and y are different variables, any free occurrence of x in E is a free occurrence of x in $\lambda y.E$.

If there is a free occurrence of x in a term E, we say "x occurs free in E".

1.2.2 Definition of "bound occurrence"

For any variable x,

B.1 Any bound occurrence of x in E or in F is a bound occurrence of x in EF.

B.2 Any bound occurrence of x in E is a bound occurrence of x in $\lambda y.E$ for any variable y (including x).

B.3 The x immediately after the λ in $\lambda x.E$ is a bound occurrence of x in $\lambda x.E$, for any term E.

B.4 If x and y are the same variable, then any free occurrence of x in E is a bound occurrence of x in $\lambda y.E$.

If there is a bound occurrence of x in a term E, we say "x occurs bound in E". Note that x can occur both free and bound in a term.

1.2.3 Example

x and y occur free in xy and x occurs free in $\lambda y.yx$. The occurrences of y and u in $(\lambda x.xx)xyu$ are free. x occurs both free and bound in $(\lambda x.xx)xyu$ — the first three occurrences are bound and the last is free.

1.2.4 Fine points

B.3 is not always given in the literature, and in fact it would make no difference to later applications if it were omitted. By including it, we can say that every occurrence of any variable in any term is either free or bound.

1.3 Substitution

We use a specific notation for substitution. Informally, if E and F are λ-terms and x is a variable, then $F[x \leftarrow E]$ is the term obtained by substituting E for every free occurrence of x in F. There are technical complications with this simple idea that force us to define substitution more carefully.

1.3.1 Notation

The most common notation in the literature for $F[x \leftarrow E]$ is $F[E/x]$. Also used is $F[x := E]$.

1.3.2 Formal rules for substitution

The rules below, a modification of those in [2], give a formal recursive definition of substitution. They are useful in formal proofs concerning properties of terms. Normally, the informal definition given above is adequate for hand calculation.

In these rules, E, F and G are any terms and x and y are any variables.

S.1 $x[x \leftarrow E] = E$.

S.2 If c is a constant or a variable different from x, then $c[x \leftarrow E] = c$.

S.3 $(FG)[x \leftarrow E] = (F[x \leftarrow E])(G[x \leftarrow E])$.

S.4 $(\lambda x.F)[x \leftarrow E] = \lambda x.F$.

S.5 If y is different from x,

a) If x occurs free in F and y occurs free in E, then

$$(\lambda y.F)[x \leftarrow E] = (\lambda z.(F[y \leftarrow z]))[x \leftarrow E]$$

where z is a new variable that does not occur free in E or F.

b) Otherwise,

$$(\lambda y.F)[x \leftarrow E] = \lambda y.(F[x \leftarrow E])$$

1.3.3 Examples

The complicated part of substitution involves substituting in expressions involving λ. S.4 says in effect that you can't substitute for a bound variable. Thus $(\lambda y.xy)[y \leftarrow uv] = \lambda y.xy$.

S.5(a) is necessary to avoid "variable capture". Suppose the term you wanted to substitute for x in $\lambda y.xy$ had a free y in it. Then the y would become bound because of the y following the λ. In order to avoid this, we must change the bound variable to a new one:

$$(\lambda y.xy)[x \leftarrow uy] = \lambda z.uyz$$

S.5(b) applies when the statement "x occurs free in F and y occurs free in E" is false. By DeMorgan's Law, this happens in one of two ways. If x does not occur free in F then there is nothing to substitute for. For example,

$$(\lambda y.zy)[x \leftarrow uv] = \lambda y.zy$$

On the other hand, if y does not occur free in E, then there is no variable to be captured. For example,

$$(\lambda y.xy)[x \leftarrow uv] = \lambda y.(uv)y = \lambda y.uvy$$

1.4 Exercises

In Exercises 1 to 4, restore all the parentheses except the outer ones.

1.• $xy(uv)w$.

2.• $\lambda x.xx$.

3.• $(\lambda x.xx)y$.

4.• $\lambda xy.xy(uv)$.

Find all the free and bound occurrences of x in Exercises 5 to 9.

5.• $\lambda x.xx$.

6.• $(\lambda x.x)x$.

7.• $\lambda xy.xy(uv)$.

8.• $\lambda uv.xy(uv)$.

9.• $(\lambda x.xy)ux$.

In Exercises 10 to 14, write out the indicated expression.

10.• $xz(yz)[x \leftarrow uv]$.

11.• $xz(yz)[z \leftarrow uv]$.

12.• $\lambda x.xz(yz)[z \leftarrow xu]$

13.• $\lambda xyz.xz[z \leftarrow uv]$.

14.• $\lambda xyz.xz[y \leftarrow uv]$.

Chapter 2

The operational semantics of the λ-calculus

2.1 Meaning

The **semantics** of a language is its meaning. The semantics of a formal language such as the λ-calculus must be given by a construction of some sort that associates a meaning to each expression in the language. If the construction consists of a list of actions to take place, for example a list of machine instructions, the semantics is called the "operational semantics". If the construction consists of a mathematical object that in some sense expresses the meaning each expression should have to a *person*, the semantics is called the "denotational semantics".

The phrases "operational semantics" and "denotational semantics" are not normally given general definitions in the literature; rather, one defines the operational and/or denotational semantics for the system under discussion only.

In this chapter, we define an operational semantics for the λ-calculus in terms of rewrite rules. The meaning of an expression will turn out to be an expression in the same language, a kind of normal form for the given expression. This is as if one said that the meaning of a Boolean expression was its disjunctive normal form, or the meaning of the polynomial $x^2 - 3x^3 + 2x^2 - 5$ was its standard form $-3x^3 + 3x^2 - 5$.

2.2 α-conversion

2.2.1 Definition

If y is not free in F, rewriting $\lambda x.F$ as $\lambda y.(F[x \leftarrow y])$ is called **α-conversion** and is written $(\lambda x.F) \underset{\alpha}{\rightarrow} \lambda y.(F[x \leftarrow y])$ Two terms with the property that one is obtained from the other by α-conversion applied to subexpressions are said to be **α-congruent**. If E and F are α-congruent, one writes $E \underset{\alpha}{=} F$.

2.2.2 Example

$\lambda x.xuv \underset{\alpha}{=} \lambda y.yuv$ and $\lambda y.(\lambda x.xy) \underset{\alpha}{=} \lambda y.(\lambda u.uy)$. But note that $\lambda y.(\lambda x.xy)$ is not α-congruent to $\lambda y.(\lambda y.yy)$.

α-congruent expressions have the same "meaning", not only in the operational semantics but in any reasonable semantics.

2.3 β-reduction

2.3.1 Definition

Rewriting $(\lambda x.F)E$ as $F[x \leftarrow E]$ is called **β-reduction** or **β-conversion** and is written $(\lambda x.F)E \underset{\beta}{\rightarrow} (F[x \leftarrow E])$. The term $F[x \leftarrow E]$ is called a **β-contraction** of $(\lambda x.F)E$.

2.3.2 Example

$(\lambda x.xyz)(uv) \underset{\beta}{\rightarrow} uvyz$ and

$$(\lambda x.xyz)(\lambda v.vu) \underset{\beta}{\rightarrow} (\lambda v.vu)yz \underset{\beta}{\rightarrow} yuz$$

Note that, by contrast,

$$(\lambda x.x(yz))(\lambda v.vu) \underset{\beta}{\rightarrow} (\lambda v.vu)(yz) \underset{\beta}{\rightarrow} yzu$$

2.3.3 Example

$(\lambda x.xx)(\lambda x.xx) \underset{\beta}{\rightarrow} (\lambda x.xx)(\lambda x.xx)$. This is the fundamental example of an "infinite loop" in the λ-calculus.

β-reduction is the heart of the λ-calculus. It is the formalization of the familiar operation in mathematics of substituting a value for (every occurrence of) a variable to evaluate a function. For example the value of the function $F(x) = x^2 - 5x$ at 3 is obtained by substituting 3 for every occurrence of x in the body of the function F: so $F(3) = 3^2 - 5 \cdot 3 = -6$. In the informal λ notation used sometimes by logicians and computer scientists (and once in a while a mathematician!), $(\lambda x.x^2 - 5x)3 = 6$.

2.4 η-reduction

2.4.1 Definition

If x is not free in F, then $\lambda x.Fx$ may be rewritten as F. This is called **η-reduction** or **η-conversion**. Note that the term being converted is $\lambda x.Fx$. By our conventions, $\lambda x.Fx = \lambda x.(Fx) = \lambda x.((F)x)$. It is not $(\lambda x.F)x$. The latter term already β-reduces to F by Definition 2.3.1.

2.4.2 Example

$\lambda x.zyx \underset{\eta}{\rightarrow} zy$. Note that $\lambda x.zyx$ and zy have the same effect when applied to an arbitrary term:

$$(\lambda x.zyx)E = zyx[x \leftarrow E] = zyE$$

The presence of η reduction allows us to prove the Theorem 2.5.4 below, which says in effect that if two terms have the same applicative behavior (when you apply them to the same thing you get the same thing), then they are the same. But being "the same" has to be defined first!

2.5 Semantic Equality

It is a basic principle of the λ-calculus that any of the conversion rules, α, β or η, preserve the semantics of a λ-expression, whatever the semantics under consideration. Roughly speaking, if two expressions E and E' can be converted to each other by using the conversion rules forward or backward, we will write $E = E'$ and say that E and E' are **semantically equal** or **semantically congruent**. Note that this has a different meaning from the statement $E \equiv E'$, which means that E and E' are the same *syntactically*. The statement $E = E'$ means that E and E' have the same meaning.

The formal definition of semantic equality involves some subtleties we have not previously mentioned.

2.5.1 Definition

Semantic equality of two expressions E and E' is denoted $E = E'$ and is the least equivalence relation that satisfies the following laws:

SE.1 If F is any expression, then $E = E' \Rightarrow (EF = E'F$ and $FE = FE')$.

SE.2 (Rule ξ). $E = E' \Rightarrow \lambda x.E = \lambda x.E'$.

SE.3 If $E \underset{\alpha}{\rightarrow} E'$, then $E = E'$.

SE.4 If $E \underset{\beta}{\rightarrow} E'$, then $E = E'$.

SE.5 If $E \underset{\eta}{\to} E'$, then $E = E'$.

2.5.2 Example

Using S.5(b),

$$(\lambda yx.xy)(xz) = \lambda x.xy[y \leftarrow xz] = \lambda u.u(xz) \quad (2.1)$$

In the last equality, we could have used v instead of u, but in any case $\lambda u.u(xz) = \lambda v.v(xz)$ by SE.3.

2.5.3 Example

$$\begin{aligned}&(\lambda x.x(yz))(\lambda v.vu)(\lambda w.zzw)\\ &\quad = (\lambda v.vu)(yz)(\lambda w.zzw)\\ &\quad = yzu(\lambda w.zzw) = yzu(zz)\end{aligned} \quad (2.2)$$

The first two equalities are by β-reduction and the last by η-reduction.

2.5.4 Theorem (Extensionality)

Let x be any variable. In the pure λ-calculus, if x does not occur free in E or F, then $Ex = Fx$ implies that $E = F$.

Proof Suppose x is not free in E or F and $Ex = Fx$. Then by Rule ξ, $\lambda x.Ex = \lambda x.Fx$. But then by η-reduction, $E = F$.

More detail is given in [1], Theorem 2.1.29. (Note that references to sections or theorem numbers in [1] refer only to the *revised* edition.)

2.5.5 Terminology

In general, a theory of functions is said to be "extensional" if whenever $F(x) = G(x)$ for all x in their domains, then $F = G$. This is usually the case in mathematical treatments of functions. A logical treatment may differentiate functions which have different formulas or algorithms but the same applicative behavior. Such a point of view is said to be an "intensional" treatment (note the spelling).

2.6 δ-reduction

In a λ-calculus with constants, the constants will in general have special reduction rules. Such rules are collectively called δ rules.

2.6.1 Example

The three constants I, S and K with δ-rules given below are famous in the λ-calculus literature because it turns out that you can recover all of the pure λ-calculus by assuming only these constants and their rules (not assuming α, β or η reduction, but assuming extensionality). This is done in [1]. The converse is true, too (see Exercises 10, 11 and 12). The system that is based on I, S and K is called the **combinatory calculus**.

D.1 $\mathsf{I}E \underset{\delta}{\to} E$.

D.2 $\mathsf{K}EE' \underset{\delta}{\to} E$.

D.3 $\mathsf{S}EE'E'' \underset{\delta}{\to} EE''(E'E'')$.

2.6.2 Elimination of bound variables

S and K allow one to produce a calculus with the same expressiveness as the λ-calculus without the λ's or bound variables. More precisely, the facts in the following theorem allow one to replace all expressions by combinations of S, K, I and free variables. Actually, you can get rid of I – see Problem 14.

2.6.3 Theorem

For any expressions E and E', $\lambda x.EE' = \mathsf{S}(\lambda x.E)(\lambda x.E')$, and if x is not free in E, $\lambda x.E = KE$.

2.6.4 Example

$\lambda x.xx = \mathsf{S}(\lambda x.x)(\lambda x.x) = \mathsf{SII}$.

2.6.5 Example

$\lambda x.yx = \mathsf{S}(\lambda x.y)(\lambda x.x) = \mathsf{S}(\mathsf{K}y)\mathsf{I}$.

2.6.6 List operators

One can add list-making operations to the λ-calculus as follows. Add new constants [CONS], [HEAD] and [TAIL] and δ-reductions

L.1 $[\mathsf{HEAD}]([\mathsf{CONS}]xy) \underset{\delta}{\to} x$.

L.2 $[\mathsf{TAIL}]([\mathsf{CONS}]xy) \underset{\delta}{\to} y$.

In fact, however, these three operations can be defined in the pure λ-calculus. We set

LD.1 $[\mathsf{CONS}] = \lambda xyz.zxy$.

LD.2 $[\mathsf{HEAD}] = \lambda x.x(\lambda yz.y)$.

LD.3 $[\mathsf{TAIL}] = \lambda x.x(\lambda yz.z)$.

Then we can prove L.1 and L.2. We do the second and leave the first as an exercise:

$$\begin{aligned}
[\mathsf{TAIL}]([\mathsf{CONS}]pq) &= (\lambda x.x(\lambda yz.z))([\mathsf{CONS}]pq) \\
&= ([\mathsf{CONS}]pq)(\lambda yz.z) \\
&= (\lambda xyz.zxy)pq(\lambda yz.z) \\
&= (\lambda yz.zpy)q(\lambda yz.z) \\
&= (\lambda z.zpq)(\lambda yz.z) \\
&= (\lambda yz.z)pq \\
&= (\lambda z.z)q \\
&= q
\end{aligned}$$

2.7 Normal forms

2.7.1 Definition

A λ-expression is a **redex** if there is a subexpression to which β-reduction or η-reduction can be applied. The expression is in **normal form** if it is not a redex.

2.7.2 Example

Any expression without any occurrence of λ is in normal form.

2.7.3 Example

The expression $\lambda x.xy$ is in normal form.

2.7.4 Example

The expression $(\lambda x.xy)(\lambda u.uu)$ is a redex since it can be β-reduced to $(\lambda u.uu)y$, and then to yy; the latter is in normal form.

2.7.5

If a λ-expression can be reduced to a normal form, then we will take that normal form as its meaning in the operational semantics. In that case, we would hope that every expression could be reduced to normal form and that no expression could be reduced to two different normal forms. However, the first hope is not justified (Example 2.7.6 below). The second hope is justified if we take "different" to mean "semantically unequal". That follows from the Church-Rosser Theorem below.

2.7.6 Example

Let $\mathsf{D} = \lambda x.xx$. Then DD is not in normal form and cannot be reduced to normal form. Indeed, η-reduction does not apply and if you apply β-reduction to DD you get DD again.

2.7.7 Theorem (Church-Rosser)

If a λ-expression E can be converted to normal forms F and F', then $F \underset{\alpha}{\to} F'$.

We omit the (hard) proof.

2.7.8 Example

In Example 2.5.2, we pointed out that $(\lambda yx.xy)(xz)$ could be reduced either to $\lambda u.u(xz)$ or to $\lambda v.v(xz)$. However, those two expressions are semantically equal.

2.7.9 Example

Sometimes there are two subexpressions that can be reduced and you can choose which one to do first. For example,

$$(\lambda x.xy)((\lambda u.yu)a) = (\lambda x.xy)(ya) = yay$$

or

$$(\lambda x.xy)((\lambda u.yu)a) = ((\lambda u.yu)a)y = yay$$

2.7.10 Example

Sometimes one way of reducing leads to a normal form and the other way leads to an infinite loop. For example, $(\lambda x.y)(\mathsf{DD}) = y$ if you reduce the leftmost redex first, but you can reduce DD forever if you are perverse. However, in a sense that we will not make precise, you can always reduce any expression to normal form, if it can be reduced to normal form at all, by systematically applying β or η-reduction to the leftmost redex in the expression. (This is also called the Church-Rosser Theorem.)

Strategies for reduction have been studied extensively. See [1], Chapter 3.

2.8 Exercises

In these exercises, you may use Theorem 2.5.4.

In Problems 1 through 9, reduce the expression to normal form, if possible.

1.• $(\lambda y.yx)(\lambda y.yx)$.

2.• $(\lambda x.yx)(\lambda x.yx)$.

3.• $(\lambda x.yxy)z$.

4.• $(\lambda zx.xz)(\lambda x.x(yz))a$.

5.• $u(\lambda x.xy)v$.

6.• $(\lambda yx.xy)(x\lambda v.vv)b$.

7. $(\lambda xy.yxx)(\lambda xy.yxx)uv$.

8. $(\lambda x.xx)(\lambda y.yzy)ab$.

9. $(\lambda xy.yx)a(\lambda u.uu)$.

10.• Prove that $\mathsf{I} = \lambda x.x$.

11. Prove that $\mathsf{K} = \lambda xy.x$.

12. Prove that $\mathsf{S} = \lambda xyz.xz(yz)$.

13. Let F have δ-rule $\mathsf{F}EE' = E'$.
a) Show how to define F as a λ-abstraction.
b) Show how to define F in terms of S and K.

14. Prove that $\mathsf{SKK} = \mathsf{I}$.

15. Prove Theorem 2.6.3. (Hint: For the expression involving S, apply both sides to x and use extensionality).

16. Express $\lambda xy.yx$ in terms of S, K and I only.

17. Express $\lambda xy.x(\lambda u.y)$ in terms of S and K only.

18. Express $\lambda x.x\,(\lambda y.x)$ in terms of S, K and I.

19. Prove L.1.

Chapter 3

Representation of computable functions

3.1 Numerals in the λ-calculus

In order to define λ-terms representing the nonnegative integers, we first must define ordered pairs.

3.1.1 Definition

Let E and F be terms. Then $\langle E, F\rangle$ denotes the term $\lambda z.zEF$.

3.1.2 Definition

$[\mathsf{FIRST}] = \lambda xy.x = \mathsf{K}$ and $[\mathsf{SECOND}] = \lambda xy.y$.

3.1.3 Proposition

For any terms E and F,

$$\langle E, F\rangle[\mathsf{FIRST}] = E$$

and

$$\langle E, F\rangle[\mathsf{SECOND}] = F$$

3.1.4 Definition

Let '0' $= \mathsf{I}$ and for any nonnegative integer n, let '$n+1$' $= \langle[\mathsf{SECOND}], $'$n$'$\rangle$. A term of the form '$n$' for some nonnegative integer n is called a **numeral**.

Thus

'1' $= \langle[\mathsf{SECOND}], \mathsf{I}\rangle$
'2' $= \langle[\mathsf{SECOND}], \langle[\mathsf{SECOND}], \mathsf{I}\rangle\rangle$

and so on. This is not the only possible definition of numeral.

3.1.5 Representing primitive recursive functions

We can easily represent the basic primitive recursive functions on these numerals.

3.1.6 Definition

The successor function $[\mathsf{SUCC}]$ is given by

$$[\mathsf{SUCC}] = \lambda x.\langle [\mathsf{SECOND}], x\rangle$$

and $[\mathsf{PRED}]$ by

$$[\mathsf{PRED}] = \lambda x.x[\mathsf{SECOND}]$$

The constant zero function is $[\mathsf{ZERO}] = \lambda x.\mathsf{I}$.

Note that $[\mathsf{PRED}]\text{`0'} = [\mathsf{SECOND}]$, which is not a numeral. Thus $[\mathsf{PRED}]$ does not give the correct value on '0'. This can be repaired using the choice function in Definition 3.2.3 below.

3.1.7 Finite sequences

Finite sequences are obtained the same way as ordered pairs:

3.1.8 Definition

For any positive integer n and terms E_1, E_2, $\ldots, E_n$, define

$$\langle E_1, E_2, \ldots, E_n\rangle = \lambda z.zE_1E_2\cdots E_n$$

and define the projection functions by

$$\mathsf{P}_i^n = \lambda z.z(\lambda x_1x_2\cdots x_n.x_i)$$

where z is a variable not occurring free in $E_1, E_2, \ldots, E_n$.

3.1.9 Proposition

For any positive integer n and terms E_1, E_2, $\ldots, E_n$,

$$\mathsf{P}_i^n\langle E_1, E_2, \ldots, E_n\rangle = E_i$$

3.2 Constructions

3.2.1 Proposition

$E\circ F = \lambda x.(E(Fx)$. Then $(E\circ F)T = E(F(T))$ for any term T.

Note that this definition works properly for terms with normal forms. It does not give the correct definition of composition for terms that may not have a normal form when applied to some integers n. The proof that λ-definable partial functions are preserved under composition is difficult and not given here. (See [1], Section 8.4.)

3.2.2 Proposition

*Let $[\mathsf{TRUE}] = \lambda xy.x$ and $[\mathsf{FALSE}] = \lambda xy.y$. Then if E and F are any terms, then $[\mathsf{TRUE}]EF = E$ and $[\mathsf{FALSE}]EF = F$.*Note that $[\mathsf{TRUE}] = [\mathsf{FIRST}] = \mathsf{K}$ and $[\mathsf{FALSE}] = [\mathsf{SECOND}]$. This proposition allows us to define a branching operation.

3.2.3 Definition

If B, E and F are terms, then

$$[\mathsf{IF}\ B\ \mathsf{THEN}\ E\ \mathsf{ELSE}\ F] = BEF$$

Note that if B evaluates to $[\mathsf{TRUE}]$ or $[\mathsf{FALSE}]$, this will behave like an ordinary branching operator, but if B evaluates to anything else, the results are arbitrary.

3.3 Fixed point operators

3.3.1 Definition

Let $F : S \to S$ be a function. A **fixed point** for F is an element x of F for which $F(x) = x$.

3.3.2 The poset of partial functions

As an example of how fixed points can be used to define functions, let $\mathcal{P}$ denote the set of partial functions from $\mathbb{N}$ to $\mathbb{N}$. Let $\mathcal{H} : \mathcal{P} \to \mathcal{P}$ be the function for which for any function $F \in \mathcal{P}$,

$$\mathcal{H}(F)(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot F(n-1) & \text{if } n > 0 \end{cases}$$

3.3.3 Proposition

The factorial function is the unique fixed point of $\mathcal{H}$.

3.3.4 The fixed point operator

Remarkably, every λ-term has a fixed point, and moreover the fixed point can be calculated by a λ-term.

3.3.5 Theorem

Let $\mathsf{Y} = \lambda z.(\lambda x.z(xx))(\lambda x.z(xx))$. *Then for any term* F, $F(\mathsf{Y}F) = \mathsf{Y}F$.

Proof Let $\mathsf{W} = \lambda x.F(xx)$. Then

$$\mathsf{Y}F = \mathsf{WW} = \lambda x.F(xx)\mathsf{W} = F(\mathsf{WW}) = F(\mathsf{Y}F)$$

3.3.6 Example

Suppose we want an expression F with the property that $Fz = zF$ for any variable z. This is equivalent to finding F such that $F = \lambda x.xF$. since $(\lambda x.xF)z = zF$. Therefore F must be the fixed point of the expression $G = \lambda gx.xg$, since $GF = (\lambda gx.xg)F = \lambda x.xF$ (so if $GF = F$ then $F = \lambda x.xF$ by extensionality).

Therefore $F = \mathsf{Y}G = (\lambda u.G(uu))(\lambda u.G(uu))$. Now $G(uu) = (\lambda gx.xg)(uu) = \lambda x.x(uu)$. Let $H = \lambda ux.x(uu)$. Then $F = HH = (\lambda ux.x(uu))H = \lambda x.x(HH)$, so that $Fz = (HH)z = (\lambda x.x(HH))z = z(HH) = zF$ as required.

3.3.7 Primitive recursion

To define a function F so that it satisfies

$$F(k,n) = \begin{cases} G(n) & \text{if } k = 0 \\ H(F(k-1,n),k,n) & \text{otherwise} \end{cases}$$

we set

$$F = \mathsf{Y}\lambda fxy.$$
$$[\mathsf{IF}\ [\mathsf{ZERO}]x\ \mathsf{THEN}\ Gy\ \mathsf{ELSE}\ Hf([\mathsf{PRED}]xy)xy]$$

Thus if G and H can be given by λ-expressions, then so can F.

We can get functions of more than one variable by using a sequence instead of y.

3.3.8 Minimalization

Let P be a two-place predicate on the numerals, so that it is a λ-expression with the property that for any nonnegative integers m and n, P‘m’‘n’ is either [TRUE] or [FALSE].

Then to express the idea that $G(n)$ is the least m for which P‘m’‘n’ = [TRUE], we set

$$\hat{P} = \mathsf{Y}\lambda hxy.\Big[\mathsf{IF}\ Pxy\ \mathsf{THEN}\ x\ \mathsf{ELSE}\ \hat{P}([\mathsf{SUCC}]x)y\Big]$$

and $G = \hat{P}$‘0’. Then for any variable u,

$$Gu = \Big[\mathsf{IF}\ P\text{‘0’}u\ \mathsf{THEN}\ \text{‘0’}\ \mathsf{ELSE}\ \hat{P}\text{‘1’}u\Big]$$

and

$$\hat{P}\text{‘1’}u = \Big[\mathsf{IF}\ P\text{‘1’}u\ \mathsf{THEN}\ \text{‘1’}\ \mathsf{ELSE}\ \hat{P}\text{‘2’}u\Big]$$

and so on. Thus if a predicate is expressible in the λ-calculus, then so is the function defined from it by minimalization.

3.3.9 Theorem ((Kleene))

A function $F : \mathbb{N}^k \to \mathbb{N}$ *is computable if and only if there is a* λ *expression* E *for which for all nonnegative integers* $n_1, n_2, \ldots, n_k$,

$$E\,\text{‘}n_1\text{’}\text{‘}n_2\text{’}\ldots\text{‘}n_k\text{’} = \text{‘}F(n_1,\ldots,n_k)\text{’}$$

In this section, we have outlined how to represent any computable function by a λ expression. The complete proof and the proof of the converse is in [1], section 6.3.

3.4 Exercises

1. Prove Proposition 3.1.3.

2. Show that [PRED]‘0’ = [SECOND] (see section 3.1.5).

3. Repair the definition in section 3.1.5 so that the predecessor function gives ‘0’ when evaluated at ‘0’.

4. Construct a term F such that $Fxy = yxF$.

5. Construct a term P such that P‘m’‘n’ = ‘$m+n$’.

6. Prove Proposition 3.3.3.

7. A function $F : S \to S$ is **idempotent** if $F \circ F = F$. Show that a function is idempotent if and only if every element in the image of F is a fixed point of F. (The image of F is the set of values of F, in other words the set $\{y \mid \text{There is } x \in S \text{ such that } F(x) = y\}$.)

Answers to selected problems

Chapter 1

1. $((xy)(uv))w$.

2. $\lambda x.(xx)$.

3. $(\lambda x.(xx))y$.

4. $\lambda x.(\lambda y.(xy)(uv))$

5. All of them.

6. The first two.

7. Both of them.

8. None.

9. The first two.

10. $(uv)z(yz) = uvz(yz)$.

11. $x(uv)(y(uv))$. The parentheses are all necessary.

12. $\lambda w.w(xu)(y(xu))$.

13. $\lambda xyz.xz$. Note that rule S5 requires that x be different from y.

14. $\lambda xyz.xz$.

Chapter 2

1. xx.

2. yy.

3. yzy.

4. $a(\lambda x.x(yz))$.

5. $u(\lambda x.xy)v$ (it is already in normal form).

6. $(\lambda yx.xy)(x\lambda v.vv)b = (\lambda u.u(x\lambda v.vv))b = b(x\lambda v.vv)$.

10. For any term E, $(\lambda x.x)E = E = \mathsf{I}E$, so the result follows by Theorem 2.5.4.

Chapter 3

References

1. H. P. Barendregt, **The Lambda Calculus,** revised edition. Studies in Logic and the Foundations of Mathematics **103**, North-Holland, 1984.
2. J. R. Hindley and J. P. Seldin, **Introduction to Combinators and λ-Calculus.** London Mathematical Society Student Texts 1. Cambridge University Press, 1986.
3. J. Lambek and P. J. Scott, **Introduction to Higher-Order Categorical Logic.** Cambridge Studies in Advanced Mathematics 7. Cambridge University Press, 1986.
4. G. E. Revesz, **Lambda-calculus, Combinators, and Functional Programming.** Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1988.

Index