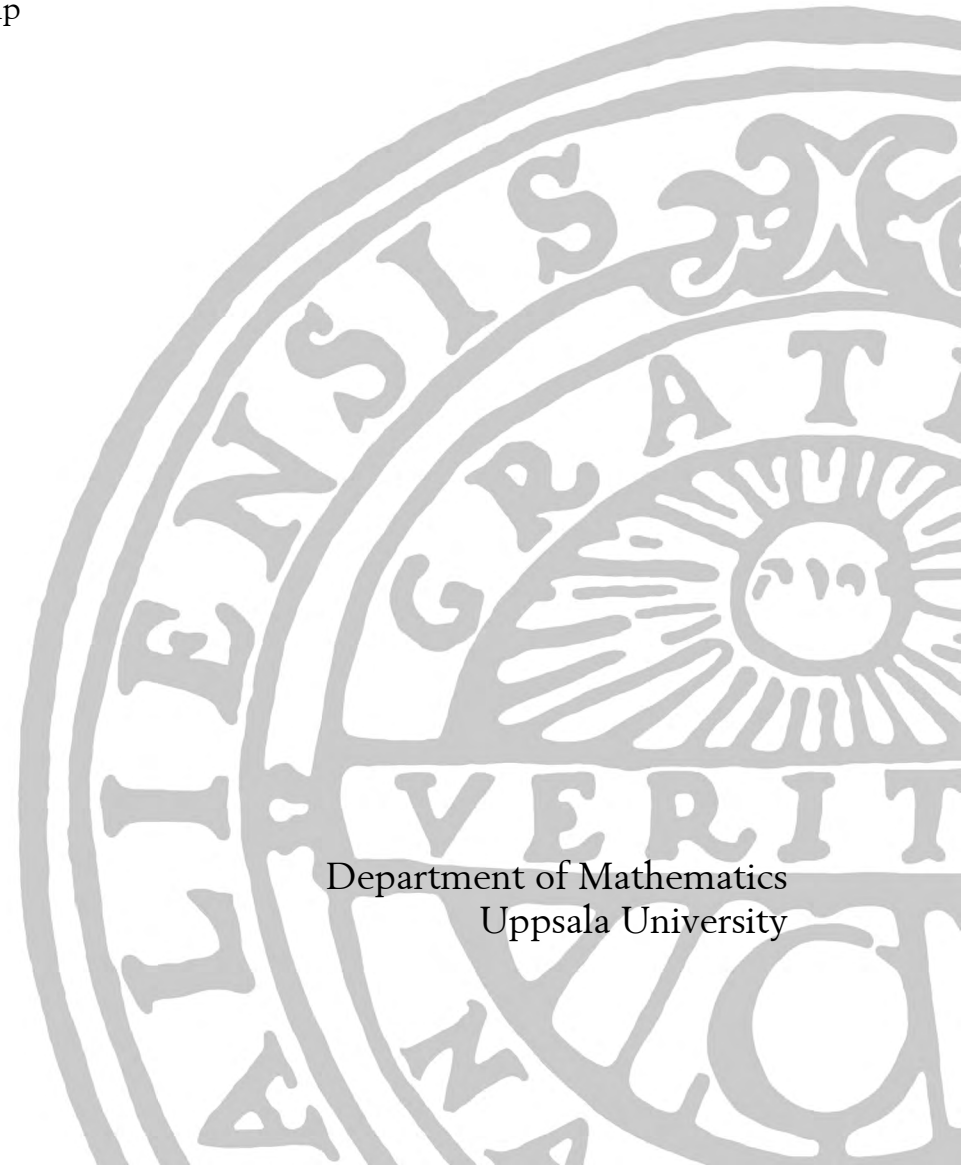
UPPSALA
UNIVERSITET

U.U.D.M. Project Report 2017:34

λ-Calculus and Decidability

Erik Larsson

Examensarbete i matematik, 15 hp
Handledare: Vera Koponen
Examinator: Jörgen Östensson
Augusti 2017



Department of Mathematics
Uppsala University

λ-Calculus and Decidability

Erik Larsson

8th August 2017

1 Introduction

Commonly, mathematics uses "sets" as a basis. However, λ-calculus was developed to explore mathematics using "operators" as instead. [1] The resulting λ-calculus is a mathematical model of functional programming, where λ-terms represents programs in any given functional programming language. [2] In addition to introducing the reader to λ-calculus, the concept of recursive functions is defined, and it is shown that every recursive function, whether it is partial recursive or total recursive, can be defined by a λ-term. The main result is the Scott-Curry theorem: that for any two λ-terms, there does not exist a recursive function which can determine if for every possible input, their outputs are the same. Since λ-terms represents computer programs, and the Church-Turing thesis claims that every computable function is recursive, if the Church-Turing thesis is true, then the main result can be translated into: "For any two functional programs, there exists no computer program which can determine if they, for every input, output the same values." Finally, section 7 discusses the Church-Turing thesis and functional programs in relation to the previous material.

Contents

2 Basic concepts

Before discussing decidability in λ-calculus, it is necessary to introduce the λ-terms, which is done here. Some notation and terminology pertaining the λ-terms is introduced. Furthermore, partial and total functions, as well as of algorithms are defined. This collection of definitions, while some may seem trivial and obvious to the reader, are included in order to provide both a foundation to build other concepts upon and to avoid any possible ambiguity which may rise from poorly defined terms. The definitions pertaining λ-terms, are from [1].

2.1 λ-calculus

Before defining the building blocks of λ-calculus, there is an argument for defining the language which the λ-calculus is defined in, its vocabulary. Henceforth, it will be assumed that the vocabulary is pure, and consists of infinitely many variable symbols, unless otherwise stated. Definition 2.1 is referred to for a clear and precise definition of the above terms.

Definition 2.1. A **vocabulary** is a set of infinitely many variable symbols, and finitely or infinitely many constant symbols. If a vocabulary consists of no constant symbols, it is called **pure**. Otherwise, it is called **applied**.

The previously mentioned building blocks of λ-calculus are called "λ-terms", defined in 2.2. As mentioned in the introduction, and as hinted at by the use of the words "application" and "abstraction", there is a parallel between λ-calculus and the usual notion of (programmable) functions. For example, consider the function $f = x+y$ and consider the λ-term $F = \lambda x.x+y$. If λ is regarded as a meaningless symbol and the symbols between λ and the dot as the parameters of the function, and the final expression is the output of the function, then one can identify the function f with the abstraction F.

Definition 2.2. An **atom** is any variable or constant symbol.
The set of all **λ-terms** is defined recursively:

1. All atoms are λ-terms.
2. If M and N are λ-terms, then so is (MN).
3. If M is a λ-term, and x is a variable, then $(\lambda x.M)$ is a λ-term.

(MN) is an **application**, and $\lambda x.M$ is an **abstraction**.

Some important notation is introduced in the following remark.

Remark.

1. Let $=$ denote identity between λ-terms. Note that $\lambda x.x \neq \lambda y.y$, even though the functions which they can be seen as parallel to ($f(x) = x$ and $f(y) = y$) are seen as the same.
2. Let $ABCD$ denote $((AB)C)D$
3. Let $MN = PQ$ denote $M = P$ and $N = Q$
4. Let $\lambda xy.N$ denote $\lambda x.(\lambda y.N)$

Moreover, definition 2.3 gives a meaning to an occurrence of a λ-term within another, which will be useful when defining substitution later.

Definition 2.3. For λ-terms M, N, P, Q, define **an occurrence of** P as follows:

1. P occurs in P
2. If P occurs in M, or N, then P occurs in (MN)
3. If P occurs in Q, or $P = x$, then P occurs in $\lambda x.Q$

2.2 Functions

For clarity, a definition of "partial function" and of "total function" is included, see definition 2.4. Since a k-tuple of natural numbers can be encoded as a single natural number, using the k first prime numbers, it is unnecessary to consider co-domains of higher dimensions than 1.

Definition 2.4. A **partial function** is a function $f : \mathcal{U} \subseteq \mathbb{N}^k \to \mathbb{N}$. A **total function** is a function $g : \mathbb{N}^k \to \mathbb{N}$.

Remark. $0 \in \mathbb{N}$

Remark. In this report, "function" will be synonymous to "total function", unless otherwise specified.

In later parts of this report, when discussing recursive functions and the Church-Turing thesis, a definition of "algorithm" is needed, see definition 2.5. While an algorithm is here defined as a sequence of instructions for a human, an algorithm can informally be thought of as a computer program in any programming language. The definition was chosen since it assumes no previous knowledge of programming, and there would be no additional gain to defining an algorithm in terms of computer programs.

Definition 2.5. An **algorithm** is a sequence of instructions which, given an arbitrary (but finite) amount of time as well as an infinite amount of pen and paper, can be carried out by a human.

3 α-conversion and β-equality

The terms of λ-calculus have already been defined, but the one equivalence relation which we so far may use $(=)$ is not entirely satisfactory. Consider the functions f and g, defined as $f(x) = x$ and $g(y) = y$. Then, f and g can be regarded as the same function. However, let $F = \lambda x.x$ and $G = \lambda y.y$ be λ-terms. Then, $F \neq G$. The operation α-conversion addresses this change of variable, and is introduced here.

In addition to α-conversion, β-reduction and β-equality are also introduced in this section. The operation β-conversion converts a term of the form $(\lambda x.X)Y$ to a term of the form X, but with all occurrences of x in X replaced by Y. In terms of functions, this would, if $f(x) = 2x+1$ and $g(x) = x + 1$, convert $f(g(x))$ to $f(x) = 2(x + 1) + 1$. The β-equality takes α-conversion and adds β-reduction and inverse β-reduction, in order to create an equivalence relation which equates all functions which for all possible inputs produce the same outputs.

The definitions of this section are based on definitions in [1].

3.1 α-conversion

Before defining α-conversion, there is a need to define bound variables, free variables, and substitution. Their definitions (3.1, 3.2) are straightforward and intuitive, with the possible exception of items 5 and 6 of 3.2. The two cases when the substitution is applied to $\lambda y.P$ are needed to ensure that the resulting λ-term will not depend on the variable y.

Definition 3.1. An occurrence of a variable x in a λ-term P is **bound** if there exists a λ-term M so that x occurs in M and $\lambda x.M$ occurs in P. If P has no bound occurrence of the variable y, then y is said to be **free** in P. A **closed term** is a λ-term with no free variables.

Definition 3.2. Let N, P, Q be λ-terms, and let x, y, z be variables, where z is the first bound variable of N and P. Then define a **substitution** of x with N, written $[N/x]$, by

1. $[N/x]x = N$
2. $[N/x]a = a$ for all atoms $a \neq x$
3. $[N/x](PQ) = ([N/x]P([N/x]Q)$
4. $[N/x](\lambda x.P) = \lambda x.P$
5. $[N/x](\lambda y.P) = \lambda y.[N/x]P$, if $y \neq x$ and if y is bound in N or x is bound in P
6. $[N/x](\lambda y.P) = \lambda z.[N/x][z/y]P$ if $y \neq x$ and y is free in N and x is free in P

Remark. Any substitution has an inverse substitution which returns the original λ-term. To show this, let N, P, A, B, Z be λ-terms so that Z does not occur in P, and let x, y, z be variables so that $x \neq y$ and z is the first bound variable of N, P and A. Consider the following cases:

$P = x \Rightarrow [x/N][N/x]P = [x/N]N = x$

P atomic and $P \neq x \Rightarrow [x/Z][N/x]P = [x/Z]P = P$

$P = (AB) \Rightarrow [N/x]P = ([N/x]A[N/x]B)$ which can be reversed by the other cases

$P = \lambda x.A \Rightarrow [x/Z][N/x](\lambda x.A) = [x/Z](\lambda x.A) = \lambda x.A$

$P = \lambda y.A$ and y is bound in N or x is a bound variable in A
$\Rightarrow [N/x](\lambda y.A) = \lambda y.[N/x]A$ which by the other cases is reversible

$P = \lambda y.A$ and y is a free variable in N and x is a free variable in A
$\Rightarrow [x/N][N/x](\lambda y.A) = [x/N](\lambda z.[N/x][z/y]P)$ which by the other cases is reversible

Now, α-conversion can be defined in definition 3.3.

Definition 3.3. Let the λ-term P contain an occurrence of $\lambda x.M$, and let y be bound in M. Replacing $\lambda x.M$ with $\lambda y.[y/x]M$ is a **change of bound variable** in P. Moreover, P is **congruent to** Q, or P **α-converts** to Q if Q can be obtained from P by a finite series of changes of bound variables. This is written $P \equiv_\alpha Q$ or $P \equiv Q$.

Remark. α-conversion is an equivalence relation. Reflexivity and transitivity are trivial properties, and symmetry follows from that substitution is reversible.

If two λ-terms differ by at most a finite sequence of changes of bound variables, then they act on other terms in the same way. However, the reverse statement is not true - if two terms act on other terms in the same way, they are not necessarily α-congruent to each other. The operation $=_\beta$, introduced in the next section, fixes this issue of α-conversion.

3.2 β-equality

Before defining β-equality, it is convenient to define β-reduction, since the latter is an extension of the former.

Definition 3.4. Let M, N be λ-terms, and let x be a variable symbol. Then, $(\lambda x.M)N$ is a **β-redex**, and $[N/x]M$ is its **contractum**. Replacing a β-redex by its contractum is called a **contraction** or a **β-contraction**. A λ-term P **β-reduces** to a term Q if Q can be obtained from P by a finite sequence of β-contractions and changes of bound variables. This is written $P \triangleright_\beta Q$.

The operator $\triangleright_\beta$ is reflexive and transitive. However, it is not symmetric, since a contraction cannot be reversed by another contraction. Before addressing this asymmetry, β-normal forms are defined.

A particular subset of the λ-terms are those which does not contain any β-redexes. Since they cannot be β-reduced, they are in some sense simpler than but acts in the same way as other λ-terms which reduce to them.

Definition 3.5. A λ-term A is in **normal form** or **β-normal form** if no β-redexes occur in A. The **normal form of** A is the term A' such that $A \triangleright_\beta A'$ and A' is in normal form, if such a term exists, and undefined otherwise.

The following theorem which shows that $\triangleright_\beta$ is confluent is due to Church-Rosser, and will not be proven here. A proof can be found in [1], theorem A.2.11.

Theorem 3.6. *Let A, B, C be λ-terms. If $A \triangleright_\beta B$ and $A \triangleright_\beta C$, then there exists a λ-term D such that $B \triangleright_\beta D$ and $C \triangleright_\beta D$.*

Corollary 3.7. *If A is a λ-term, and A' is the normal form of A, then A' is unique.*

The corollary shows that it is sensible to define *the* normal form of a λ-term, as was done in definition 3.5. A proof of the corollary is as follows.

Proof. Assume that A is a λ-term, and for a contradiction, assume that B, C are its normal forms. So $A \triangleright_\beta B$ and $A \triangleright_\beta C$. Then, by 3.6, there exists a λ-term D such that $B \triangleright_\beta D$ and $C \triangleright_\beta D$. Thus both B and C must contain β-redexes, so they are not in normal form, which is a contradiction. Thus, A has at most one normal form.

It is straightforward to change β-reduction into an equivalence relation, by forcing symmetry. This leads to β-equality, see definition 3.8. Since β-equality equates λ-terms which represents functions which are evaluated the same, it is natural to consider β-equality when working with λ-terms.

Definition 3.8. Two λ-term A, B are said to be **$=_\beta$-equal** if B can be obtained from A by a finite series of β-reductions and reversed β-reductions. This is written $A =_\beta B$.

Remark. For any two λ-terms A, B, $A =_\beta B \Rightarrow A \equiv B$. However, the converse is false.

4 Recursive Functions

The set of all recursive functions, sometimes referred to as the set of all computable functions, is a set of functions which, if the Church-Turing thesis is true, can be calculated by algorithms. The Church-Turing thesis is addressed in section 7, and will not be assumed to be true or false. It will be clear if a conclusion relies on the thesis.

Here, the set of the recursive functions is built, by defining the basic functions, primitive recursion, and unbounded search. Also, the set of the primitive recursive functions is defined, since they are a part of the recursive functions. The recursive functions are later used to define decidability, and are also important in the Scott-Curry undecidability theorem. The definitions here are based on definitions in [3].

4.1 Basic functions

A particular subset of the functions are the so-called basic functions. They are basic in the sense that they are building blocks used to construct the set of primitive recursive functions. The basic functions are the successor, zero, and projection functions.

Definition 4.1. The **basic functions** are the following:

1. $s(x) = x + 1$
2. $z(x) = 0$
3. $p_i^k(x_1, ..., x_k) = x_i$

4.2 Primitive recursion

Since the set of all recursive functions is an extension of the primitive recursive functions, it is sensible to first define primitive recursion and primitive recursive functions.

Definition 4.2. A set A of functions is **closed under primitive recursion** if for any $k > 1, k \in \mathbb{N}$, and all $f(x_1, ..., x_{k-1}) \in A$, $g(x_1, ..., x_{k+1}) \in A$, there is an $h \in A$ defined as follows:

$$\begin{aligned} h(0, x_2, ..., x_k) &= f(x_2, ..., x_k) \\ h(x_1 + 1, x_2, ..., x_k) &= g(x_1, ..., x_k, h(x_1, ..., x_k)) \end{aligned}$$

Remark. In definition 4.2, primitive recursion is conducted over the first argument of h. However, by composition with projector functions, primitive recursion may be conducted over any argument of h.

Definition 4.3. The **set of all primitive recursive functions** is the smallest set S of functions such that

1. $x, z, p_i^k \in S$ for all $i, k \in \mathbb{N}$
2. S is closed under primitive recursion

Remark. Addition, multiplication, and polynomials are examples of primitive recursive functions. [3]

4.3 Unbounded search

The final piece needed before defining the set of all recursive functions is unbounded search. By thinking of a function $f(x_1, ..., x_k, y)$ as satisfied for all y where $f(x_1, ..., x_k, y) = 0$, unbounded search of f finds the least such y, if it exists.

Definition 4.4. A set A of functions is **closed under unbounded search** if for any $f(x_1, ..., x_{k+1}) \in A$, there exists $us_y f \in A$ defined as follows:

$$\begin{aligned} us_y f(x_1, ..., x_k, y) &= z \text{ if } z \text{ is the least natural number such that} \\ &\quad f(x_1, ..., x_k, z) = 0 \\ &\quad \text{and is otherwise undefined} \end{aligned}$$

Remark. In definition 4.4, search is conducted over the last argument of f. However, by composition with projector functions, search can be conducted over any argument of f.

Remark. According to [1], definition 4.4 is equivalent to the following:

A set A of functions is closed under unbounded search if for all primitive recursive $f(x) \in A$ and primitive recursive $g(x_1, ..., x_{k+1}) \in A$, there exists $h(x_1, ..., x_k) \in A$ such that

$$\begin{aligned} h(x_1, ..., x_k) &= f(\mu(g(x_1, ..., x_k, n) = 0)) \\ &\quad \text{where there exists an } n \text{ such that } g(x_1, ..., x_k, n) = 0 \\ &\quad \text{and } \mu(g(x_1, ..., x_k, n)) \text{ is the least such } n \end{aligned}$$

Using unbounded search and primitive recursion, the set of all recursive functions is defined.

Definition 4.5. The **set of all recursive functions** is the smallest set S of functions such that

1. $x, z, p_i^k \in S$
2. S is closed under primitive recursion
3. S is closed under unbounded search

Remark. Every primitive recursive function is a recursive function. However, not every recursive function is primitive recursive. An example of this is the Ackermann function A, defined below. [3]

$$\begin{aligned} A(0, x) &= x + 1 \\ A(n, 0) &= A(n - 1, 1) \\ A(n, x) &= A(n - 1, A(n, x - 1)) \end{aligned}$$

Remark. A function which can be computed by an algorithm is recursive if and only if the Church-Turing thesis is true. However, a recursive function can always be computed by an algorithm, since the basic functions, primitive recursion, unbounded search, and composition can be computed by algorithms.

5 Decidability

As indicated by the name of the main result, the Scott-Curry undecidability theorem for λ-calculus, there is a need to define decidability before presenting the theorem. As such, this section will define decidability, as well as give a fixed point lemma and a definability theorem. These statements, which relates recursive functions with the λ-terms, are given with proofs. The fixed point lemma is used to prove the definability theorem, which in turn is used to prove the Scott-Curry undecidability theorem for λ-calculus, which is given later in this section. Finally, a corollary to the Scott-Curry theorem and a perspective from functional programming is given. The definitions and results here are based on [1].

5.1 Decidable sets

In order to discuss the decidability of λ-calculus, it is necessary to define decidability for sets of natural numbers, see definition 5.1. This definition will be used to define decidability in the context λ-calculus.

Definition 5.1. Let $A, B \subseteq \mathbb{N}$. Then they are **recursively separable** if there exists a recursive function φ such that $\varphi(n) = 1 \ \forall n \in A$ and $\varphi(n) = 0 \ \forall n \in B$. A is said to be **recursive** or **decidable** if it and its complement are recursively separable.

Remark. The actual outputs of the function φ above are irrelevant, since as long as they are different depending on if the input is in A or B it is possible to apply translation, scaling, and/or modulo operations to the output to obtain 0 or 1.

Remark. Informally, if the Church-Turing thesis is true, two sets $A, B \subseteq \mathbb{N}$ are decidable if and only if there exists an algorithm which is able to in finite time determine, for any natural number X, if $X \in A$ or if $X \in B$.

In order to do apply the above to λ-terms, we need a parallel between the natural numbers and the λ-numbers (terms). This parallel is a Gödel numbering, the existence of which is stated in assumption 5.2 of [1].

Definition 5.2. A **Gödel numbering** of the λ-terms is an injective map with the set of λ terms as domain and $\mathbb{N}$ as codomain.

Remark. There are several possible Gödel numberings of the λ-terms. The results of this report will hold for any Gödel numbering such that there exists recursive functions τ, υ so that

$$\begin{aligned} \tau(gd(X), gd(Y)) &= gd(XY) \\ \upsilon(n) &= gd(\lambda xy.x^n y) \end{aligned}$$

The term $\lambda xy.x^n y$ is the Church numeral of n, written $\bar{n}$ and defined in definition 5.4.

Using the Gödel numbering above, we define decidability for λ-calculus as in definition 5.3.

Definition 5.3. Two sets A, B of λ-terms are said to be recursively separable if the two sets of their corresponding Gödel numbers are recursively separable. A is said to be **recursive** or **decidable** if it and its complement are recursively separable.

5.2 Definability by λ-terms

Previously, when defining α-conversion and β-equality, a parallel between the λ-terms and the functions on natural numbers was suggested, but not formalized. The Church numerals, see definition 5.4, defines a set of λ-terms which correspond to $\mathbb{N}$. Using these, the suggested parallel will be formalized.

Definition 5.4. Let $n \in \mathbb{N}$. Then, the **Church numeral representing** n is $\bar{n} = \lambda xy.x^n y$.

Remark. x^n is used as shorthand notation for $xx...x$, where x occur n times. Let $x^0 y = y$ by definition.

Remark. In addition to providing an equivalence to the natural numbers in λ-calculus, the Church numerals have the following useful property: let $n \in \mathbb{N}$, and let X, Y be λ-terms. Then

$$\bar{n}XF =_\beta X^n F$$

The proof is an exercise in using the definitions of $=_\beta$ and the Church numerals:

$$\begin{aligned} \bar{n}XF &= (\lambda xy.x^n y)XF \\ &=_\beta [X/x](\lambda y.x^n y)F \\ &=_\beta [F/y](X^n y) \\ &=_\beta X^n F \end{aligned}$$

Definition 5.5 uses the Church numerals to formalize the parallel between the natural numbers and the λ-terms, by considering the range of the function and the λ-terms.

Definition 5.5. Let f be any (total or partial) function, and F a λ-term. Then F **defines** f if, for all $x_1, ...x_k$, $Fx_1...x_k =_\beta \overline{f(x_1, ..., x_k)}$ if $f(x_1, ..., x_k)$ exists, and $Fx_1...x_k$ has no normal form if $f(x_1, ..., x_k)$ does not exist.

5.3 Combinators

A particular subset of the λ-terms are the combinators, see definition 5.6.

Definition 5.6. A **combinator** is a closed λ-term containing no constants.
Moreover, the following combinators are defined:

$$\begin{array}{lcl} \boldsymbol{I} & \equiv & \lambda x.x \\ \boldsymbol{K} & \equiv & \lambda xy.x \\ \boldsymbol{D} & \equiv & \lambda xyz.z(\boldsymbol{K}y)x \end{array}$$

Remark. The combinator $\boldsymbol{K}$ represents the constant function returning x. Also, the combinator $\boldsymbol{D}$ can be seen as an ordered pair combinator, since if $z = \bar{0}$ then $\boldsymbol{D} = x$, and otherwise $\boldsymbol{D} = y$.

The following fixed-point lemma shows the existence of solutions of some equations of λ-terms. Since equation 2 is proven by deriving a solution to the equation, the second part of the proof can be used to construct solutions of other equations of the same form.

Lemma 5.7. *There is a combinator $\boldsymbol{Y}$ such that*

$$\boldsymbol{Y}x =_\beta x(\boldsymbol{Y})x \tag{1}$$

Moreover, for every λ-term Z, the equation

$$Xy_1...y_n =_\beta [X/x]Z \tag{2}$$

has a solution for X.

Proof. For 1, let $\boldsymbol{Y} \equiv \boldsymbol{U}\boldsymbol{U}$, where $\boldsymbol{U} \equiv \lambda ux.x(uux)$. Then,

$$\begin{array}{lcl} \boldsymbol{Y}x & \equiv & (\lambda u.(\lambda x.x(uux)))\boldsymbol{U}x \\ & =_\beta & [\boldsymbol{U}/u](\lambda x.x(uux))x \\ & \equiv & (\lambda x.x(\boldsymbol{U}\boldsymbol{U}x))x \\ & =_\beta & x(\boldsymbol{U}\boldsymbol{U}x) \\ & \equiv & x(\boldsymbol{Y}x) \end{array}$$

For 2, let $X \equiv \boldsymbol{Y}(\lambda xy_1...y_n.Z)$. Then,

$$\begin{array}{lcl} Xy_1...y_n & \equiv & (\boldsymbol{Y}(\lambda xy_1...y_n.Z))y_1...y_n \\ & =_\beta & (\lambda xy_1...y_n.Z)\boldsymbol{Y}(\lambda xy_1...y_n.Z)y_1...y_n \\ & \equiv & (\lambda xy_1...y_n.Z)Xy_1...y_n \\ & =_\beta & [X/x]Z \end{array}$$

Definition 5.8. A **fixed point combinator** is a combinator $\boldsymbol{Y}$ such that equation 1 in 5.7 is satisfied.

Remark. The $\boldsymbol{Y}$ which was chosen in the proof was found by Alan Turing in 1937. There are other fixed-point combinators, for example $\boldsymbol{Y}_{CR} \equiv \lambda x.\boldsymbol{V}\boldsymbol{V}$, where $\boldsymbol{V} \equiv \lambda y.x(yy)$. This $\boldsymbol{Y}_{CR}$ is attributed to Curry-Rosenblom [1].

5.4 Definability theorem

Theorem 5.9. *Let f be a recursive function. Then, f is defined by a combinator F.*

The proof of the theorem constructs the combinator F, using induction over the length of f[1].

Proof. Assume that f is recursive. Then, f is the result of compositions, primitive recursion, and unbounded search applied to the basic functions. Thus, the combinator F is constructed inductively. Each basic function is defined by the following λ-terms:

$$\begin{aligned}\bar{s} &= \lambda uxy.x(uxy)\\ \bar{z} &= \lambda xy.y\\ \bar{p_i^k} &= \lambda x_1...x_k.x_i\end{aligned}$$

Remark. The zero function is the Church numeral for 0, and for any Church numeral $\bar{n}$, $\bar{s}\bar{n}$ gives $\overline{n+1}$:

$$\begin{aligned}\bar{s}\bar{n} &= (\lambda uxy.x(uxy))(\lambda xy.x^n y) =_\beta\\ &=_\beta [\lambda xy.x^n y/u](\lambda xy.x(uxy)) =_\beta\\ &=_\beta \lambda xy.x((\lambda xy.x^n y)xy) =_\beta\\ &=_\beta \lambda xy.xx^n y = \lambda xy.x^{n+1}y = \overline{n+1}\end{aligned}$$

It is trivial that the projection terms $\bar{p_i^k}$ returns argument i of the k arguments.

Moreover, let $\varphi_1, ...\varphi_n$ be recursive functions, and define the composition $\varphi_1(\varphi_2, ..., \varphi_n)$ with

$$\boldsymbol{C} = \lambda x_1...x_k.(\overline{\varphi_1}(\overline{\varphi_2}x_1...x_k, ..., \overline{\varphi_n}x_1...x_k))$$

For any recursive functions φ_1, φ_2, define primitive recursion with

$$\boldsymbol{P} = \lambda u x_1...x_k.(\boldsymbol{R}(\bar{\varphi_1}x_1...x_k)(\lambda uv.\bar{\varphi_2}uvx_1...x_k)u)$$

Where

$$\begin{aligned}\boldsymbol{R} &= \lambda xyu.u(\boldsymbol{Q})(\boldsymbol{D}\bar{0}x)\bar{1} \text{ and}\\ \boldsymbol{Q} &= \lambda yv.\boldsymbol{D}(\bar{s}(v\bar{0}))(y(v\bar{0})(v\bar{1}))\end{aligned}$$

Recall that $\boldsymbol{D}$ is defined in definition 5.6.

In order to motivate the choice of $\boldsymbol{P}$, assume that

$$\begin{cases}\boldsymbol{R}XY\bar{0} =_\beta X\\ \boldsymbol{R}XY\overline{k+1} =_\beta Y\bar{k}\boldsymbol{R}XY\bar{k}\end{cases} \tag{3}$$

Then,

$$\begin{cases}\boldsymbol{P}\bar{0}x_1...x_k &=_\beta \varphi_1 x_1...x_k\\ \boldsymbol{P}\overline{k+1}x_1...x_k &=_\beta (\lambda uv.\bar{\varphi_2}uvx_1...x_k)\bar{k}(\boldsymbol{R}(\bar{\varphi_1}x_1...x_k)(\lambda uv.\bar{\varphi_2}uvx_1...x_k)\bar{k}\\ &=_\beta \bar{\varphi_2}\bar{k}(\boldsymbol{P}\bar{k}x_1...x_k\end{cases} \tag{4}$$

The final $=_\beta$ follows from the definition of $\boldsymbol{P}$. Also, equation 4 together with the definition of primitive recursion (definition 4.2) shows that $\boldsymbol{P}$ defines primitive recursion.

In order to show equation 3, one has to show that for any λ-terms X, Y, and positive integer n:

[1]Not length as in the number of symbols in f, but rather length as in the number of operations (composition, primitive recursion, and unbounded search) that has been applied to the basic functions to obtain f.

$$\begin{array}{rcl} \boldsymbol{Q}Y(\boldsymbol{D}\bar{n}X) & =_\beta & \boldsymbol{D}(\bar{s}(\boldsymbol{D}\bar{n}X\bar{0}))(Y(\boldsymbol{D}\bar{n}X\bar{0})(\boldsymbol{D}\bar{n}X\bar{1}) \\ & =_\beta & \boldsymbol{D}(\bar{s}\bar{n})(Y\bar{n}X) \\ & =_\beta & \boldsymbol{D}(\overline{n+1})(Y\bar{n}X) \end{array}$$

By induction[2] it follows that

$$\boldsymbol{Q}Y^n(\boldsymbol{D}\bar{0}X) =_\beta \boldsymbol{D}(\bar{n}X_n) \text{ for some term } X_n \tag{5}$$

Then, from equation 5 and the definition of $\boldsymbol{D}$:

$$\boldsymbol{R}XY\bar{n} \equiv (\lambda xyu.u(\boldsymbol{Q}y)(\boldsymbol{D}\bar{0}x)\bar{1})XY\bar{n}$$

Now,

$$\begin{array}{rcl} \boldsymbol{R}XY\bar{0} & \equiv & (\lambda xyu.u(\boldsymbol{Q}y)(\boldsymbol{D}\bar{0}x)\bar{1})XY\bar{0} \\ & =_\beta & \bar{0}(\boldsymbol{Q}Y)(\boldsymbol{D}\bar{0}X)\bar{1} \\ & =_\beta & (\boldsymbol{Q}Y)^0(\boldsymbol{D}\bar{0}X)\bar{1} \\ & =_\beta & \boldsymbol{D}\bar{0}X\bar{1} \\ & =_\beta & X \end{array}$$

And

$$\begin{array}{rcl} \boldsymbol{R}XY\overline{n+1} & \equiv & (\lambda xyu.u(\boldsymbol{Q}y)(\boldsymbol{D}\bar{0}x)\bar{1})XY\overline{n+1} \\ & =_\beta & \overline{n+1}(\boldsymbol{Q}Y)(\boldsymbol{D}\bar{0}X)\bar{1} \\ & =_\beta & (\boldsymbol{Q}Y)^{n+1}(\boldsymbol{D}\bar{0}X)\bar{1} \\ & =_\beta & (\boldsymbol{Q}Y)((\boldsymbol{Q}Y)^n(\boldsymbol{D}\bar{0}X))\bar{1} \\ & =_\beta & (\boldsymbol{Q}Y)\boldsymbol{D}(\bar{n}X_n)\bar{1} \\ & =_\beta & \boldsymbol{D}(\overline{n+1})(Y\bar{n}X_n)\bar{1} \\ & =_\beta & Y\bar{n}X_n \\ & =_\beta & Y\bar{n}(\boldsymbol{R}XY\bar{n}) \end{array}$$

Thus satisfying equation 3.

For unbounded search, let $h(x_1,...,x_k) = f(\mu(g(x_1,...,x_k,n)$ for all $x_1,...,x_n \in \mathbb{N}$, and let f, g be defined by combinators $\boldsymbol{F}, \boldsymbol{G}$. Then, any λ-term $\boldsymbol{H}$ satisfying equation 6 will define h.

$$\begin{cases} \boldsymbol{H} \equiv \lambda x_1...x_k.\boldsymbol{F}(\boldsymbol{A}x_1...x_k\bar{0}) \\ \boldsymbol{A}x_1...x_ky = (\text{If } \boldsymbol{G}x_1...x_ky = \bar{0} \text{ then } y, \text{else } \boldsymbol{A}x_1...x_k(\bar{s}y)) \end{cases} \tag{6}$$

By lemma 5.7 and the definition of $\boldsymbol{D}$, $\boldsymbol{A} \equiv \boldsymbol{Y}(\lambda ux_1...x_ky.\boldsymbol{D}y(ux_1...x_k(\bar{s}y))(Gx_1...x_ky))$ for any fixed-point combinator $\boldsymbol{Y}$, since:

$$\begin{array}{rcl} \boldsymbol{A}x_1...x_ky & \equiv & (\boldsymbol{Y}(\lambda ux_1...x_ky.\boldsymbol{D}y(ux_1...x_k(\bar{s}y))(Gx_1...x_ky)))x_1...x_ky \\ & =_\beta & (\lambda ux_1...x_ky.\boldsymbol{D}y(ux_1...x_k(\bar{s}y))(Gx_1...x_ky))\boldsymbol{Y} \\ & & (\lambda ux_1...x_ky.\boldsymbol{D}y(ux_1...x_k(\bar{s}y))(Gx_1...x_ky))x_1...x_ky \\ & \equiv & (\lambda ux_1...x_ky.\boldsymbol{D}y(ux_1...x_k(\bar{s}y))(Gx_1...x_ky))\boldsymbol{A}x_1...x_ky \\ & =_\beta & \boldsymbol{D}y(\boldsymbol{A}x_1...x_k(\bar{s}y))(Gx_1...x_ky) \\ & =_\beta & (\text{If } \boldsymbol{G}x_1...x_ky = \bar{0} \text{ then } y, \text{else } \boldsymbol{A}x_1...x_k(\bar{s}y)) \end{array} \tag{7}$$

Remark. Since every primitive recursive function is recursive, the theorem applies to primitive recursive functions as well.

[2]The base case is trivial, and the induction step can be seen by applying the above.

5.5 The Scott-Curry undecidability theorem for λ-calculus

Theorem 5.10. *Let A, B be non-empty disjoint sets of λ-terms. If A, B are closed under β-equality, then they are not recursively separable.*

Remark. Recall that if two sets A, B are not recursively separable, then they are not decidable, and therefore the name of the theorem.

Proof. Let A, B be non-empty disjoint sets of terms which are closed under β-equality. Assume that f separates A, B. Then, by theorem 5.9 there exists a combinator F so that F defines f. In other words, for all X,

$$\begin{array}{rcl} X \in A & \rightarrow & F\ulcorner X\urcorner =_\beta \bar{1} \\ X \in B & \rightarrow & F\ulcorner X\urcorner =_\beta \bar{0} \end{array}$$

Recall the definitions of τ and υ, see remark 5.1, as well as the definition of the combinator $\boldsymbol{D}$, see definition 5.6. Then, by theorem 5.9 there exists T defining τ, and N defining υ. Moreover, consider any $P \in A$, $Q \in B$. Define $J \equiv H\ulcorner H\urcorner$, where $H \equiv \lambda y.DPQ(F(Ty(Ny)))$. Then,

$$\begin{array}{rcl} J & =_\beta & DPQ(F(T\ulcorner H\urcorner(N\ulcorner H\urcorner))) \\ & =_\beta & DPQ(F(T\ulcorner H\urcorner\ulcorner\ulcorner H\urcorner\urcorner)) \\ & =_\beta & DPQ(F\ulcorner(H\ulcorner H\urcorner)\urcorner) \\ & \equiv & DPQ(F\ulcorner J\urcorner) \end{array}$$

Thus, due to the ordered pair property of $\boldsymbol{D}$,

$$\begin{array}{l} F\ulcorner J\urcorner =_\beta \bar{1} \Rightarrow J =_\beta DPQ(F\ulcorner J\urcorner) =_\beta Q \\ F\ulcorner J\urcorner =_\beta \bar{0} \Rightarrow J =_\beta DPQ(F\ulcorner J\urcorner) =_\beta P \end{array}$$

So, if $f(gd(J)) = 1$, then

$$\begin{array}{rrcl} & F\ulcorner J\urcorner & =_\beta & \bar{1} \\ \Rightarrow & J & =_\beta & Q \\ \Rightarrow & J & \in & B \\ \Rightarrow & f(gd(J)) & = & 0 \end{array}$$

Similary, if $f(gd(J)) = 0$, then

$$\begin{array}{rrcl} & F\ulcorner J\urcorner & =_\beta & \bar{0} \\ \Rightarrow & J & =_\beta & P \\ \Rightarrow & J & \in & A \\ \Rightarrow & f(gd(J)) & = & 1 \end{array}$$

Since $f(gd(J))$ is neither 1 nor 0, but cannot assume any other vaues, f cannot exist. Thus, A and B are not recursively separable, which proves the theorem.

Corollary 5.11. *Let X be any λ-term. Then, the set of all λ-terms which are β-equal to X is not recursively separable from the set of all λ-terms not β-equal to X.*

The proof of the corollary is simple - take A in theorem 5.10 as the set of all λ-terms equal to X, and let B be the complement of A.

6 Defining partial recursive functions

So far, functions have been assumed to be total. There is no natural extension of the set of primitive recursive functions which allows functions defined partially, but unbounded search can be adapted in such a way. This partial unbounded search can then be used instead of the unbounded search defined in section 4.3. It will be shown that the definability theorem holds for these partial functions as well, although the proof uses leftmost- and quasi-leftmost reductions, which will be defined here as well. Theorem 6.8 and its preceding lemmas are based on [6], and the definition of partial recursive functions as well as theorem 6.9 are based on [1].

Remark. To facilitate distinguishing between the previously defined recursive functions and the partial recursive functions defined here, the recursive functions defined in 4.5 will henceforth be called **total recursive functions**.

6.1 Partial Recursive Functions

The following definitions are similar to the corresponding definitions when defining ordinary, total recursive functions. However, they allow functions which are undefined for certain inputs.

Definition 6.1. The set A is closed under **partial unbounded search** if for all primitive recursive $f(x) \in A$ and primitive recursive $g(x_1, ..., x_{k+1}) \in A$, there exists $h(x_1, ..., x_k) \in A$ such that

$$\begin{aligned} h(x_1, ..., x_k) &= f(\mu(g(x_1, ..., x_k, n) = 0)) \\ &\quad \text{where } \mu(g(x_1, ..., x_k, n) = 0) \text{ is the least } n \text{ such that } g(x_1, ..., x_k, n) = 0, \\ &\quad \text{and undefined if no such } n \text{ exists.} \end{aligned}$$

Remark. Any set which is closed under unbounded search is closed under partial unbounded search, but the converse statement is not true.

The set of all partial recursive functions are constructed similarly to the set of all total recursive functions.

Definition 6.2. The **set of all partial recursive functions** is the smallest set S of functions such that

1. $x, z, p_i^k \in S$
2. S is closed under composition
3. S is closed under primitive recursion
4. S is closed under partial unbounded search

6.2 Reductions

The reductions uses $\triangleright\beta$, which was defined in 3.4.

Definition 6.3. A sequence of β-contractions, possibly with interweaving α-conversions, is a **reduction**.

Intuitively, a leftmost reduction evaluates a composition of functions by always evaluating the leftmost function first.

Definition 6.4. An occurrence φ of a β-redex in a λ-term A is called **maximal** if and only if it does not occur in another redex which occurs in A. If φ is the leftmost of the maximal β-redexes of A, then φ is **leftmost maximal**. If reduction of A is such that in every β-contraction of the reduction, the leftmost maximal β-redex is contracted, and the reduction continues infinitely or until there are no more β-redexes to contract, then the reduction is the **leftmost reduction** of A. A **left reduction** is a leftmost reduction which does not need to be infinite or terminated.

Remark. For any term A, the leftmost reduction is unique, since for any given λ-term, its leftmost maximal redex is unique.

The following lemma provides a method with which it is possible to find the normal form of any λ-term. However, if no normal form exists, then the leftmost reduction is of infinite length. As a consequence of this, if a λ-term has an infinite leftmost reduction, then the term has no normal form.

Lemma 6.5. *Let A be a λ-term. Then, leftmost reduction of A ends in the normal form of A, if it exists.*

Proof. Let A be a λ-term, with normal form B. Then, by the Church-Rosser theorem, there is a sequence of contractions starting at A which results in B. Assume, for a contradiction, that this sequence of contractions is not a leftmost reduction of A. Then, for some $i \in \mathbb{N}$, the leftmost redex was not contracted in contraction i. Denote this redex R. If R is never contracted in the sequence, then B cannot be in normal form. However, if R is contracted in contraction j, where $j \geq i$, then changing the order of contractions such that contraction j occur before contraction i will result in a leftmost reduction which ends in B. Thus, the sequence of contractions must be a leftmost reduction or can be rearranged into a leftmost reduction if the sequence ends in a λ-term in normal form.

Definition 6.6. A **quasi-leftmost reduction** of a term A is reduction which continues infinitely or until there are no more β-redexes to reduce, such that for each $i \in \mathbb{N}$, if the term after contraction i is not the final term then there exists $j \geq i$ such that the leftmost maximal β-redex is β-reduced in the jth reduction of the sequence.

The following lemma can be used to rearrange the order of left reductions and not-left reductions, and will be used to prove theorem 6.8.

Lemma 6.7. *Let A, B, C be λ-terms. Assume that there exists a non-left reduction of A which results in B, and a non-empty left reduction of B which results in C. Then, there exists a λ-term B' such that there is a non-empty left reduction of A which results in B', and a non-left reduction of B' which ends in C.*

Remark. The proof of lemma 6.7 is not given here, but can be found in [6], lemma 13.2.5.

Theorem 6.8. *If a λ-term A has a quasi-leftmost reduction which is infinite, then A has no normal form.*

Proof. Let A be a λ-term which has an infinite quasi-leftmost reduction. Let A_i be A after the ith step of the reduction. Assume for a contradiction that there does not exist infinitely many i where A_i is the result of a left reduction. Let j be the largest integer such that A_j was obtained from a left reduction. Then A_{j+1} is obtained from a non-left reduction, and there is no $k \geq j+1$ such that A_k is obtained from a left reduction, which contradicts that the reduction is an infinite quasi-leftmost reduction. Thus, there exists infinitely many i where A_i is the result of a left reduction. Then, by lemma 6.7, the infinite quasi-leftmost reduction can be rearranged into an infinite sequence of reductions where the first infinitely many reductions are left reductions. Thus, there exists an infinite leftmost reduction of A. It then follows by lemma 6.5 that A has no normal form.

6.3 Definability of Partial Recursive Functions

The following theorem on the definability of partial recursive functions is stronger than theorem 5.9, but the proof requires a bit more work in addition to what was done when proving 5.9.

Theorem 6.9. *Let f be a partial recursive function. Then, f is defined by a combinator F.*

Proof. The proof for the basic functions, and primitive recursion is identical to the corresponding parts of the proof of theorem 5.9, and will therefore be omitted here. The proof shows that partial partial unbounded search is definable:

Let f, g be primitive recursive, and for all $x_1, ..., x_k \in \mathbb{N}$, let

$$h(x_1, ..., x_k) = f(\mu(g(x_1, ..., x_k, n) = 0))$$

Moreover, let $H \equiv \lambda x_1 ... x_k . \boldsymbol{F}(\boldsymbol{A} x_1 ... x_k \bar{0})$ where $\boldsymbol{A} \equiv \boldsymbol{Y}(\lambda u x_1 ... x_k y . \boldsymbol{D} y (u x_1 ... x_k (\bar{s} y))(G x_1 ... x_k y))$ for any fixed-point combinator $\boldsymbol{Y}$.

Also, for all $x_1, ..., x_k \in \mathbb{N}$,

$$H\bar{x_1}...\bar{x_k} =_\beta \overline{h(x_1, ..., x_k)}$$

Let

$$\begin{array}{rcl} T & \equiv & \lambda x . \boldsymbol{D} \bar{0} (\lambda uv . u (x (\bar{s} v)) u (\bar{s} v)) \\ P & \equiv & \lambda xy . T x (xy) (T x) y \end{array}$$

For any λ-terms X, Y, and u, v which are not free variables in XY, the definitions of P and T give that

$$\begin{array}{rl} PXY & =_\beta TX(XY)(TX)Y \\ & =_\beta \boldsymbol{D}\bar{0}(\lambda uv . u(X(\bar{s}v))u(\bar{s}v))(XY)(TX)Y \end{array}$$

So, if $XY = \bar{0}$, it follows by property of $\boldsymbol{D}$ and definition of $\bar{0}$ that

$$PXY =_\beta \bar{0}(TX)(Y) =_\beta (\lambda xy . y)(TX)(Y) =_\beta Y$$

Also, if $XY = \overline{x+1}$ for some $x \in \mathbb{N}$, the following holds by property of $\boldsymbol{D}$

$$\begin{array}{rl} PXY & =_\beta (\lambda uv . u(X(\bar{s}v))u(\bar{s}v))(TX)Y \\ & =_\beta (TX)(X(\bar{s}Y))(TX)(\bar{s}Y) \\ & =_\beta PX(\bar{s}Y) \end{array}$$

Thus,

$$\begin{array}{ll} PXY =_\beta Y & \text{if } XY =_\beta \bar{0} \\ PXY =_\beta X(\bar{s}Y) & \text{if } XY =_\beta \overline{x+1} \text{ for some } x \in \mathbb{N} \end{array}$$

Now, let

$$\bar{h} \equiv \lambda x_1 ... x_k . P(\bar{g} x_1 ... x_k)\bar{0}\boldsymbol{I}(H x_1 ... x_k)$$

Now, it remains to show that for this choice of $\bar{h}$, $\bar{h}x_1...x_k =_\beta \overline{h(x_1, ..., x_k)}$ has to be true whenever there exists a y so that $g(x_1, ..., x_k, y) = 0$, and $\bar{h}$ has no normal form when no such y exists. Now, assume that $x_1, ..., x_k$ are such that $g(x_1, ..., x_k, y) = 0$ for some y, and let y_0 be the lowest such y.

Since PXY reduces to the lowest Y so that $XY = \bar{0}$, it follows that

$$\bar{h}\bar{x_1}...\bar{x_k} =_\beta \bar{y_0}\boldsymbol{I}(H\bar{x_1}...\bar{x_k})$$

Then, since $\bar{y_0}$ is a Church numeral,

$$\begin{aligned}\bar{y_0}\boldsymbol{I}(H\bar{x_1}...\bar{x_k}) &=_\beta \boldsymbol{I}^{y_0}(H\bar{x_1}...\bar{x_k})\\ &=_\beta H\bar{x_1}...\bar{x_k}\\ &=_\beta \overline{h(x_1,...,x_k)}\end{aligned}$$

Assume that $x_1, ..., x_k$ are such that there exists no y for which $g(x_1, ..., x_k, y) = 0$. Since g is primitive recursive and therefore total, it follows that for every $y \in \mathbb{N}$, there exists a $z_y \in \mathbb{N}$ such that

$$g(x_1, ..., x_k, y) = z_y + 1$$

Let $A \equiv \bar{g}\bar{x_1}...\bar{x_k}$, and let $B \equiv Hx_1...x_k$. Thus, $A\bar{y} \triangleright_\beta \overline{z_y + 1}$, and

$$\begin{aligned}\bar{h}x_1...x_k &\triangleright_\beta PA\bar{0}\boldsymbol{I}B\\ &\triangleright_\beta TA(A\bar{0})(TA)\bar{0}\boldsymbol{I}B\end{aligned}$$

Take $y = 0$, and the following is obtained:

$$\begin{aligned}TA(A\bar{0})(TA)\bar{0}\boldsymbol{I}B &\triangleright_\beta TA(\overline{z_0 + 1})(TA)\bar{0}\boldsymbol{I}B\\ &\triangleright_\beta \lambda x.\boldsymbol{D}\bar{0}(\lambda uv.u(x(\bar{s}v))u(\bar{s}v))A(A\bar{0})(TA)\bar{0}\boldsymbol{I}B\\ &\triangleright_\beta \boldsymbol{D}\bar{0}(\lambda uv.u(A(\bar{s}v))u(\bar{s}v))(A\bar{0})(TA)\bar{0}\boldsymbol{I}B\end{aligned}$$

Since $A\bar{0} \triangleright_\beta \overline{z_0 + 1}$ and $z_0 + 1 \neq 0$,

$$\begin{aligned}\boldsymbol{D}\bar{0}(\lambda uv.u(A(\bar{s}v))u(\bar{s}v))(A\bar{0})(TA)\bar{0}\boldsymbol{I}B &\triangleright_\beta (\lambda uv.u(A(\bar{s}v))u(\bar{s}v))(TA)\bar{0}\boldsymbol{I}B\\ &\triangleright_\beta TA(A(\bar{s}\bar{0}))(TA)(\bar{s}\bar{0})\boldsymbol{I}B\\ &\triangleright_\beta TA(A\bar{1})(TA)\bar{1}\boldsymbol{I}B\end{aligned}$$

It is thus clear that $TA(A\bar{i})(TA)\bar{i}\boldsymbol{I}B \triangleright_\beta TA(A\overline{i+1})(TA)\overline{i+1}\boldsymbol{I}B$ for all $i \in \mathbb{N}$.

Also, since the contraction

$$\lambda x.\boldsymbol{D}\bar{0}(\lambda uv.u(x(\bar{s}v))u(\bar{s}v))A(A\bar{0})(TA)\bar{0}\boldsymbol{I}B \triangleright_\beta \boldsymbol{D}\bar{0}(\lambda uv.u(A(\bar{s}v))u(\bar{s}v))(A\bar{0})(TA)\bar{0}\boldsymbol{I}B$$

is leftmost maximal, the infinite series of contractions is an infinite quasi-leftmost reduction. Thus, by theorem 6.8, $\bar{h}x_1...x_k$ has no normal form, which was to be shown.

7 Church-Turing thesis

The Church-Turing thesis claims that the set of all functions which can be defined by algorithms is exactly the set of all recursive functions. Since a computer program can be defined by its code, which in turn is an algorithm, it is possible to consider computer programs instead of algorithms in the statement of the thesis. However, algorithm will be used here since computer programs, and their instruction set, change over time.

However, while not proven, the thesis has not been falsified either. A proof of the thesis would have to account for every possible instruction that an algorithm may contain, and every possible language to express the algorithm in, which is clearly not feasible. However, there is some evidence indicating that the thesis likely is true. The fact that the thesis has not been falsified, despite being mentioned as early as 1936, is one such piece of evidence.[5] Moreover, the systems of λ-calculus, Kleene's μ-recursion, Turing machines, and Herbrand-Gdel's equational definability have been shown to be equivalent in strength, despite being very different systems. [4] Finally, it is notable that the results presented here does not rely on the Church-Turing thesis, while the presented interpretation of the results does.

7.1 Functional programming

The similarity between λ-terms and functions has already been discussed, but one can also view λ-calculus as a model for functional programming, where the terms are programs. Then, the relation $\equiv$ holds for programs which are syntactically the same, except for in the naming of their private variables. Similarly, $=_\beta$ relates programs A, B which produces identical outputs for the same inputs, with no regard for the inner syntax of the programs. Then, two sets A, B of functional programs are recursively separable if there exists a recursive function φ such that φ produces one output for programs in A, and another output for programs in B. If the Church-Turing thesis is true, then "recursive function" may be replaced with "algorithm". A combinator is a program with no private variables, and which uses no global variables. The definability theorem, theorem 5.9, shows that if the Church-Turing thesis is true, then every algorithm can be represented by a combinator. By the Scott-Curry theorem, there is no algorithm which can determine for any set of programs A if any other program gives the same output as at least one program in A does, for every possible input. The corollary 5.11 is an application of this, where A contains only one program. The definability theorem for partial recursive functions, theorem 6.9, states that the above holds for algorithms which may not be defined for every possible input.

References

[1] Hindley, J. Seldin, J. *Lambda-Calculus and Combinators, and Introduction.* Cambridge University Press 2008.

[2] Machado, R. *An Introduction to Lambda Calculus and Functional Programming.* Theoretical Computer Science (WEIT) 2013.

[3] Hedman, Shawn *A First Course in Logic.* Oxford University Press 2004.

[4] Butterfield, A. Ngondi, G. E. *A Dictionary of Computer Science.* Oxford University Press 2016.

[5] Cooper, S.B. van Leeuwen, J. *Alan Turing: His Work and Impact.* Elsevier Science 2013.

[6] Barendregt, H.P. *The Lambda Calculus, its Syntax and Semantics.* North-Holland Publishing Company 1984.