TECHNICS AND INFORMATICS IN EDUCATION
6th International Conference, Faculty of Technical Sciences, Čačak, Serbia, 28–29th May 2016
TEHNIKA I INFORMATIKA U OBRAZOVANJU
6. međunarodna konferencija, Fakultet tehničkih nauka, Čačak, Srbija, 28–29. maj 2016.

UDK: 004.41/.42:519.68 Review paper

Computable functions and lambda calculus

Marko Stanković[1]
[1]Pedagogical Faculty in Vranje, University of Niš, Vranje, Serbia
e-mail markos@ucfak.ni.ac.rs

Abstract: *The lambda calculus is one of the more known formulizations of the effective procedure and is widely applied in functional programing. In relation to this, the main goal of this paper is to show a way of interpreting some computable functions via lambda terms. Thus, the paper gives a special insight into interpreting Boolean functions, Church's numerals and the most important arithmetic operations with them. A review of some combinators which have been proven to be useful when dealing with lambda terms is also given. Finally, an idea of proofs is presented and it shows that the class of lambda computable functions is equal to the class of primitive recursive functions.*

Keywords: *lambda calculus; lambda definability; Church numerals; computability*

1. INTRODUCTION

The λ-calculus is a collection of several formal systems, based on notation devised by Alonzo Church during the 1930's. It is made to describe the simplest ways in which operators and functions can be combined so as to get new operators.

In practice, λ-system ha an infinitely big grammar structure, depending on what it is used, Some have additional symbols of constants, but most was achieved through syntax limitations (for example, by restricting them to exactly specified types).

In the development of the algorithm theory there have been several formulations of an effective procedure. The most famous formulations, apart from λ-calculus, are certainly Turing's machines, Unlimited Register Machine (URM), primitive recursive functions, μ-recurrsive functions, Post's systems etc. Although they all seem completely different at first, all procedures determine one and the same class of functions.

1.1. Primitive recursive functions

We say that the function $f: \mathbb{N}^k \to \mathbb{N}$ is computable only and if only there is an effective procedure that, given any k–tuple $(x_1, \dots, x_k)$ of natural numbers, will produce $f(x_1, \dots, x_k)$ (Enderton, 2002:209).

Definition. Class of primitive recursive functions contains initial functions:

- Zero function $z(n) = 0$, for each $n \in \mathbb{N}$,
- Successor of a natural number function n, $s(n) = n + 1$ and
- Projection function $u_i^n(x_1, \dots, x_n) = x_i$, $1 \leq i \leq n$,

And all functions which are derived through final number of applications of the basic operations:

- *compositions*, i.e. if the functions $g: \mathbb{N}^m \to \mathbb{N}$ i $h_1, \dots, h_m: \mathbb{N}^k \to \mathbb{N}$ are already defined, the function $f: \mathbb{N}^k \to \mathbb{N}$ is also defined so that it stands for all $(x_1, \dots, x_k) \in \mathbb{N}^k$: $f(x_1, \dots, x_k) = g\big(h_1(x_1, \dots, x_k), \dots, h_m(x_1, \dots, x_k)\big)$ and
- *primitive recursions*, i.e. if the functions $g: \mathbb{N}^m \to \mathbb{N}$ and $h: \mathbb{N}^{m+2} \to \mathbb{N}$ are already defined the function $f: \mathbb{N}^{m+1} \to \mathbb{N}$ is also defined so that it stands for all $(x_1, \dots, x_m) \in \mathbb{N}^m$:

$$f(0, x_1, \dots, x_m) = g(x_1, \dots, x_m),$$
$$f(n+1, x_1, \dots, x_m) = h(f(n, x_1, \dots, x_m), n, x_1, \dots, x_m) \text{ for } n \in \mathbb{N}.$$

Limitation of this class is its closure to the operation of minimization which must be bounded. "By eliminating this limitation (through the introduction of the unbounded minimization) we get a class of *partially recursive functions*, which coincides with computable Turing functions" (Ognjanović, Krdžavac, 2004: 27).

2. CHURCH'S NUMERALS, DEFINABILITY

2.1. Natural numbers

λ-calculus is a theory of functions and algorithms, so the only possible way for interpreting numbers in λ-computations is regarding the same as if they were algorithms. However, a number is not an algorithm, but a datum. Although the situations may seem impossible, it is still possible to regard a number for an algorithm. For example, number 2 can be "built" by applying zero twice in the successor function. Appropriate λ-term which describes this situation is $\lambda fx.f(fx)$. It is important to stress that the algorithm is independent from the actualities of zero and successor function.

Therefore, natural number n (and zero), marked n, is interpreted as $\underline{n} \equiv \lambda fx.f^n x$, where f^n is defined in the following way: $f^0 x = x$; $f^{k+1} x = f(f^k x)$. This type of natural number coding was created by Alonzo Church and terms are named because of him as Church's terms. Church's coding is one of many possible ways (there are also Mogensen-Scott coding, which got its name by Torben Mogensen and Dana Scott). The technique in which data is seen as algorithms can be expanded on all (inductively defined) data.

2.2. Lambda definability

For a function which can be interpreted by λ-terms we say is λ-definable and we formally define it in the following way.

Definition (λ-definability). We say that the function that partially recursive function $f: \mathbb{N}^k \to \mathbb{N}$ is λ-definable if for some term F stands:

$$f(n_1, \dots, n_k) = m \Rightarrow F\underline{c}_{n_1}, \dots, \underline{c}_{n_k} =_\beta \underline{c}_m$$
$$f(n_1, \dots, n_k) = \uparrow \Rightarrow F\underline{c}_{n_1}, \dots, \underline{c}_{n_k} \text{has no normal form.}$$

We say that the term, λ-defines function f.

Also, it stands that

$$f(n_1, \dots, n_k) = m \Leftrightarrow F\underline{c}_{n_1}, \dots, \underline{c}_{n_k} =_\beta \underline{c}_m$$
$$f(n_1, \dots, n_k) = \uparrow \Leftrightarrow F\underline{c}_{n_1}, \dots, \underline{c}_{n_k} \text{has no normal form.}$$

and

$f(n_1, \dots, n_k) = \downarrow \Rightarrow F\underline{c}_{n_1}, \dots, \underline{c}_{n_k} =_\beta \underline{c}_{f(n_1, \dots, n_k)}$.

Therefore, in much simpler case when $f: \mathbb{N} \to \mathbb{N}$ for f we say it is λ-definable only and

only if there is a λ-term F so that $F(\underline{n}) =_\beta f(\underline{n})$.

2.3. Boolean values

We define interpretations of Boolean values as $true$, $false$ and term which interprets if.

$$true \equiv \lambda xy.x \qquad false \equiv \lambda xy.y \qquad if \equiv \lambda pxy.pxy$$

In other words, $true$ is a function of two arguments which returns its first argument, and false is a function which returns its second argument. Term $ifPQR$ should be read "if P then Q else R". Also, it stands $ifPQR \rhd_\beta PQR$. Then, if $P = true$ we get:

$$if\ trueQR \rhd_\beta trueQR = (\lambda xy.x)QR \rhd_\beta (\lambda y.Q)R \rhd_\beta Q.$$

Similarly we get $iffalseQR \rhd_\beta R$. Previous deliberation stand for arbitrary term P so that $P \rhd_\beta true$ or $P \rhd_\beta false$.

Based on Church – Rosser theorem (Hindley, Seldin, 2008:14), it follows that $true \neq false$ because $true$ and $false$ are different normal forms. Conjunction, disjunction and negation we interpret:

$$\wedge \equiv \lambda pq.if\ pq\ false \qquad \vee \equiv \lambda pq.if\ p\ true\ q \qquad \neg \equiv \lambda p.if\ p\ false\ true$$

Definitions can be checked by simple calculations. For example, $\wedge\ true\ true \rhd_\beta true$. Often instead of $true$ we use $\mathbf{1}$, and instead $false$ we use $\mathbf{0}$. "Notice that the Boolean $\mathbf{0}$ is the same term as the numeral $\underline{0}$, while Boolean $\mathbf{1}$ is different from the numeral $\underline{1}$" (Krivine, 1993:32).

3. ARITHEMTICS ON CHURCH'S NUMERALS

Let us define the interpretation of calculation functions with Church's numerals.

Function of *addition* we define as: $add \equiv \lambda mnfx.mf(nfx)$. Let us check the definition of adding in the following way:

$$\begin{aligned} add\ \underline{m}\underline{n} \equiv\ & (\lambda mnfx.mf(nfx))\underline{m}\underline{n} \rhd_\beta (\lambda nfx.\underline{m}f(nfx))\underline{n} \rhd_\beta \\ \rhd_\beta\ & (\lambda fx.\underline{m}f(\underline{n}fx)) \equiv \lambda fx.(\lambda gx.g^m x)f\big((\lambda hx.h^n x)fx\big) \rhd_\beta \\ \rhd_\beta\ & \lambda fx.(\lambda gx.g^m x)f\big((\lambda x.f^n x)x\big) \rhd_\beta \lambda fx.(\lambda gx.g^m x)f(f^n x) \rhd_\beta \\ \rhd_\beta\ & \lambda fx.(\lambda x.f^m x)(f^n x) \rhd_\beta \lambda fx.f^m(f^n x) \equiv \lambda fx.f^{m+n}x \equiv \underline{m+n} \end{aligned}$$

Function of *multiplication* we define as: $mult \equiv \lambda mnfx.m(nf)x$. Let us check the definition of multiplication in the following way:

$$\begin{aligned} mult\ \underline{m}\underline{n} \equiv\ & (\lambda mnfx.m(nf)x)\underline{m}\underline{n} \rhd_\beta (\lambda nfx.\underline{m}(nf)x)\underline{n} \rhd_\beta \lambda fx.\underline{m}f(\underline{n}f)x \\ \equiv\ & \lambda fx.(\lambda gx.g^m x)(\underline{n}f)x \rhd_\beta \lambda fx.(\underline{n}\,f)^m x \equiv \lambda fx.\big((\lambda hx.h^n x)\,f\big)^m x \\ \rhd_\beta\ & \lambda fx.(\lambda x.f^n x)^m x \rhd_\beta \lambda fx.(f^n)^m x \equiv \lambda fx.f^{m\cdot n}x \equiv \underline{m\cdot n} \end{aligned}$$

Exponentiation is defined by term $expt \equiv \lambda mnfx.nmfx$.

Function of *successor* of a natural number is defined with $succ \equiv \lambda nfx.f(nfx)$. Function $succ$ is defined in such a way so that reduction $succ\ \underline{n} \rhd_\beta \underline{n+1}$ stand. This is discussed in more detailed manner by Mazzola, Milmeiste and Weissmann, (2005:327).

Function which checks whether a numeral *equal to zero* is interpreted as $iszero \equiv \lambda n.n(\lambda x.\mathbf{0})\mathbf{1}$. Function is defined in such manner that reductions $iszero\ \underline{0} \rhd_\beta \mathbf{1}$ and

$iszero\ \underline{(n+1)} \rhd_\beta \mathbf{0}$.

Function *predecessor* of a natural number can be defined by a combination of some functions which were introduced before. Ordered pair (a, b) in λ-calculus can be interpreted by term $\lambda z.zab$. From the ordered pair we can isolate the first (second) element by using the $\mathbf{1}$ ($\mathbf{0}$). Function $\Phi \equiv \left(\lambda pz.z\left(succ(p\mathbf{1})\right)(p\mathbf{1})\right)$ is generated from the ordered pair $(n, n-1)$ (which is marked shortly with p), ordered pair $(n+1, n-1)$.

Sub-expression $p\mathbf{1}$ isolates the first element from the pair p. New pair is formed using this element, which in this case is enlarged by 1, while the second element is just copied in the new pair.

Predecessor of the number n is obtained by applying n times the function Φ on the ordered pair $(\lambda z.z\underline{0}\underline{0})$ and then it the second element of the ordered pair is isolated $pred \equiv (\lambda n.n\Phi(\lambda z.z\underline{0}\underline{0})\mathbf{0})$.

It should be noted that the value of the function $pred$ when it is applied to zero, is zero.

Function of *subtraction* is defined as $subtract\ \underline{m}\underline{n} \equiv \underline{n}\ pred\ \underline{m}$. The function will n times apply the function $pred$ on the interpretation of number m, which will give the desired result.

For example, we can now interpret the function of natural numbers f in λ-calculus. Let us presume that the function f is set with

$$f(n,m) = \begin{cases} 2+4m, & n=2, n=0, \\ n+5m, & \text{otherwise}, \end{cases}$$

then the appropriate λ-term[1] is:

$$\lambda nm.if\left(\vee\left(iszero\left(pred(pred\ n)\right)\right)(iszero\ n)\right)\left(add\ \underline{2}(mult\ m\ \underline{4})\right)\left(add\ n(mult\ \underline{5}\ m)\right).$$

Function for testing *equality* and *inequality*. Function which tests whether number x is bigger or equal to y is defined as $G \equiv \left(\lambda xy.iszero(x\ pred\ y)\right)$. This function is applies x times the function of predecessor on y and if the result is zero, then it follows that $x \geq y$. If $x \geq y$ and $y \geq x$, then $x = y$. This brings us to the definition of the function E which tests the equality of two numbers: $E \equiv \left(\lambda xy.\wedge\left(iszero(x\ pred\ y)\right)\left(iszero(y\ pred\ x)\right)\right)$. Interpretation of the functions $x > y$, $x < y$ or $x \leq y$ can be similarly defined.

4. COMPUTABLE FUNCTIONS IN LAMBDA CALCULUS

4.1. Combinators

Combinator is λ-term with no free variables. Intuitively, combinators can be understood as "completely determined operations", because they have no free variables.

Definition: For the term Q we say it is a fix point of term M if it stands $MQ =_\beta Q$.

Table 1. *Combinators*

$K \equiv \lambda x.(\lambda y.x)$	Combinator which forms a constant function

[1]Due to simplification $\vee$ is not substituted by appropriate term. Similar stands for function E later on in the text.

$B \equiv \lambda x.\left(\lambda y.\left(\lambda z.x(yz)\right)\right)$	Combinator which combines two functions
$S \equiv \lambda x.\left(\lambda y.\left(\lambda z.(xz)(yz)\right)\right)$	Operator of the stronger compositions
$C \equiv \lambda x.\left(\lambda y.(\lambda z.xzy)\right)$	Combinator which switches places with arguments
$Y \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$	Curry's combinator
$\Theta \equiv \left(\lambda x.\left(\lambda f.\left(f(xxf)\right)\right)\right)\left(\lambda x.\left(\lambda f.\left(f(xxf)\right)\right)\right)$	Turing's combinator

Curry's combinaotr Y has the property reduce YX and $X(YX)$ to the same term. Turing's combinator has the property that for each λ-term X and ΘX are reduced to $X(\Theta X)$. For more details see Hindley and Seldin, (2008:34).

Theorem (fixed point theorem): Every λ-term has at least one fixed point.

Proof: It should be proven that for every term F there is term X so that $FX = X$. Let $W \equiv \lambda x.F(xx)$ and $X \equiv WW$. Then $X \equiv WW \equiv (\lambda x.F(xx))W = F(WW) = FX$.

4.2. Computability and definability

Theorem: Function f is computable if and only if it is λ-definable.

Proof: Let us show that initial functions, functions obtained by composition and recursion of initial functions, are λ-definable function. Initial function we can interpret in the following way:

- function $z(n) = 0$, is interpreted by term $\underline{0}$,
- function $s(x) = x + 1$ is interpreted by function $succ$,
- function of projection $u_i^n(x_1, \dots, x_n) = x_i,\ 1 \le i \le n$ is interpreted with $u_i^n \equiv \lambda x_1, \dots, x_n.x_i$.

Let f be function with k variables and let $g_1, \dots, g_k$ be λ–definable function with n variables. Then $h(x_1, \dots, x_n) = f\big(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n)\big)$ function obtained by composition of the function f and $g_1, \dots, g_k$. If $F,\ G_1, \dots, G_k$ are terms which interpret functions $f, g_1, \dots, g_k$, then function h is λ-definable and term which represents it is

$$H \equiv \lambda x_1, \dots, x_n.F\big(G_1(x_1, \dots, x_n), \dots, G_k(x_1, \dots, x_n)\big).$$

Now we will present the idea on how recursion can be interpreted in λ-calculus, and details of the proof can be seen in Hindley and Seldin, (2008:50). Let h be λ-definable function and let H be her interpretation in λ-calculus. Term F which interprets function f is defined in the following manner: $F \equiv \lambda u x_1, \dots x_n.(\boldsymbol{R}(Hx_1, \dots x_n)(\lambda uv.Huvx_1, \dots x_n)u)$, where $\boldsymbol{R}$ stands for recursive combinatory. This combinatory has the property that for all terms X, Y and k it stands $\boldsymbol{R}XY\ \underline{0} \rhd_\beta X$ and $\boldsymbol{R}XY\ \underline{(k+1)} \rhd_\beta Y\underline{k}(\boldsymbol{R}XY\ \underline{k})$.

Let G be the interpretation of the function g. Than the operation of minimization $\min_m(g(x_1, \dots x_n, m) = 0)$ is interpreted by term

$$\left(Y\left(\lambda f.\lambda m.if\left(iszero\big(G(x_1, \dots x_n, \underline{m})\big)\right)\underline{m}\big(f(succ\ \underline{m})\big)\right)\right)\underline{0}.$$

This proves that the class of λ-definable functions contains all the initial functions and that it is closed for composition, recursion and minimization. Therefore, this class contains all computable functions. Let us present the idea of the reverse claim.

Let us suppose that function f is represented by λ-term X. If f is n-ary function, than let us suppose that her argument is $(x_1, \dots, x_n)$ and let us write down the term $X\underline{x_1} \dots \underline{x_n}$. Than reductions are done until we reach a numeral and we return this as an answer; by the way, the function is not defined. Based on Church's thesis, it follows that the function is computable.

5. CONCLUSION

Although Alonzo Church had designed λ-calculation as a basis for constructive logic, this formalization has become one of the models for computating functions with tremendous application in informatics. Because of his extreme expressiveness, lambda terms can express complex computer data such as numbers, Boolean values, binary trees and similar, while for computation of function β-reduction is used. Recursive functions can be interpreted based in the Fixed-point theorem and Y combinator. The connection between λ-calculus and programming is best shown in correspondence between λ-calculus and program language ALGOL 60 (Landin, 1965), as well as the fact that λ-computaion forms the basis of the program language LISP. Michaelson (2011) writes about the basics of functional programing and lambda computation.

REFERENCES

[1] Enderton, H. (2002). *A Mathematical Introduction to Logic* (Second ed). USA: Elsevier.

[2] Hindley, R., Seldin, J. (2008). *Lambda-Calculus and Combinators, an Introduction*. New York: Cambridge University Press.

[3] Krivine, J. (1993). *Lambda-calculus, types and models*, (Translated from french by René Cori). Paris, Ellis Horwood.

[4] Landin, P. (1965). *Correspondance between ALGOL 60 and Church's Lambda-notation: Part I*. Communications of the ACM CACM, 8(2), 89-101. doi:10.1145/363744.363749

[5] Mazzola, G., Milmeiste, G., Weissmann, J. (2005). *ComprehensiveMathematics for Computer Scientists 2*. Berlin: Springer.

[6] Michaelson, G. (2011). *AN INTRODUCTION TO FUNCTIONAL PROGRAMMING THROUGH LAMBDA CALCULUS*. Dover Publications.

[7] Ognjanović, Z., Krdžavac, N. (2004). *Uvod u teorijsko računarstvo*. Beograd – Kragujevac.