Journal of Functional Programming 1 (2): 229–234, April 1991

THEORETICAL PEARLS

Self-interpretation in lambda calculus

HENK BARENDREGT
Faculty of Mathematics and Computer Science, Catholic University of Nijmegen, The Netherlands

This editorial emphasizes results from the theory behind functional programming (including lambda calculus, type theory and term-rewriting systems) that are particularly beautiful, and which have short and elegant proofs. Readers are encouraged to send comments or contributions to:

Henk Barendregt
Faculty of Mathematics and Computer Science,
Catholic University, Nijmegen,
Toernooiveld 1, 6525 ED Nijmegen,
The Netherlands
E-mail: henk@cs.kun.nl

Programming languages which are capable of interpreting themselves have been fascinating computer scientists. Indeed, if this is possible then a 'strange loop' (in the sense of Hofstadter, 1979) is involved. Nevertheless, the phenomenon is a direct consequence of the existence of universal languages. Indeed, if all computable functions can be captured by a language, then so can the particular job of interpreting the code of a program of that language. Self-interpretation will be shown here to be possible in lambda calculus.

The set of λ-*terms*, notation Λ, is defined by the following abstract syntax

$$\Lambda = V|\Lambda\Lambda|\lambda V.\Lambda$$

where

$$V = v|V'$$

is the set $\{v, v', v'', v''', \ldots\}$ of *variables*. Arbitrary variables are usually denoted by x, y, z, ... and λ-terms by M, N, L, ... A *redex* is a λ-term of the form

$$(\lambda x.M)N$$

and has as *contractum*

$$M[x := N],$$

that is, the result of substituting N for (the free occurrences of) x in M. Stylistically, it can be said that λ-terms represent functional programs including their input. A *reduction machine* executes such terms by trying to reduce them to normal form; that is, redexes are continuously replaced by their contracta until hopefully no more redexes are present. If such a normal form can be reached, then this is the output of the functional program; otherwise, the program diverges.

From the point of view of a reduction machine, a λ-term M can be considered as an executable. It 'itches' at many places: all redexes want to be reduced.

https://doi.org/10.1017/S0956796800020062 Published online by Cambridge University Press

1 Definition

(i) Each λ-term M has a unique natural number $\#M$ as code. One way of coding is

$$\#v^{(i)} = \langle 0, i \rangle,$$
$$\#(MN) = \langle 1, \langle \#M, \#N \rangle\rangle,$$
$$\#(\lambda x.M) = \langle 2, \langle \#x, \#M \rangle\rangle,$$

where $\langle -,- \rangle$ is some effective coding of pairs of numbers as a single number, for example,

$$\langle n, m \rangle = \tfrac{1}{2}(n+m)(n+m+1)+m.$$

(ii) Let $\ulcorner 0 \urcorner$, $\ulcorner 1 \urcorner$, $\ulcorner 2 \urcorner$, ... be some set of numerals (λ-terms representing the natural numbers). We take the *Church numerals* $\ulcorner n \urcorner \equiv \lambda fx.f^n(x)$.

Write $\ulcorner M \urcorner \equiv \ulcorner \#M \urcorner$, the *internal λ-code* of M. Now $\ulcorner M \urcorner$ does not itch: being a Church numeral it is in normal form.

Write FV(M) for the set of free variables of M. A λ-term M is *closed* if $FV(M) = \varnothing$. the set of closed λ-terms is denoted by Λ^0.

2 Definition

(i) An *interpreter* (or *evaluator*) is an (external) function $E: \Lambda \to \Lambda$ such that

$$E(\ulcorner M \urcorner) \equiv M.$$

(ii) A *self-interpreter* is a λ-term E such that for $M \in \Lambda^0$ one has

$$\mathsf{E}\ulcorner M \urcorner =_\beta M. \tag{1}$$

Here $=_\beta$ (or simply $=$) denotes convertibility between elements of Λ.

3 Remarks

(i) Equation (1) cannot hold for open terms containing free variables. Indeed, E has at most a finite number of free variables and $\ulcorner M \urcorner$ being a numeral has none, but on the right-hand side M may have arbitrarily many free variables.

(ii) Define the *quote* to be the function $Q: \Lambda \to \Lambda$ such that

$$Q(M) \equiv \ulcorner M \urcorner.$$

A *self-quote* is a λ-term Q such that (say for closed terms M)

$$\mathsf{Q}M =_\beta \ulcorner M \urcorner.$$

Such a self-quote does not exist, however. Indeed, the existence of Q implies

$$\ulcorner \mathsf{II} \urcorner =_\beta \mathsf{Q}(\mathsf{II}) =_\beta \mathsf{QI} =_\beta \ulcorner \mathsf{I} \urcorner.$$

Since numerals are in β-normal form it follows that $\ulcorner \mathsf{II} \urcorner \equiv \ulcorner \mathsf{I} \urcorner$, so $\#(\mathsf{II}) = \#\mathsf{I}$. However, II and I are different terms, and so have different codes, which is a contradiction.

Kleene already in (1936) showed that there is a self-interpreter E for the lambda calculus. One would think that E is defined by recursion on the structure of its argument. There is, however, a difficulty: closed terms are not built up inductively,

https://doi.org/10.1017/S0956796800020062 Published online by Cambridge University Press

but formed as a subset of the wider class of open terms. Kleene avoided this problem by building up closed terms from combinators S, K and I (actually, he worked with the λI-calculus and used combinators like S, B, C and I). The construction was as follows

$$\ulcorner M \urcorner \xrightarrow{\text{CL}} \ulcorner M_{CL} \urcorner \xrightarrow{\mathsf{E}_{CL}} M_{CL} =_\beta M$$

where CL is a compiler from λ-terms to combinatory terms, and E_{CL} is an interpreter for combinatory terms. The translation CL gives, for example,

$$(\lambda z . zz)_{CL} \equiv \mathsf{SII}\,(=_\beta \lambda z . \mathsf{I}z(\mathsf{I}z) =_\beta \lambda z . zz).$$

P. de Bruin (my former student) gave an essentially simpler construction of a self-interpreter for the λ-calculus. He used an idea from denotational semantics. In the following construction, F plays the role of an environment in the sense that it determines the values of the free variables.

4 Theorem (Kleene, 1936)

There exists a self-interpreter E for the lambda calculus.

Proof (de Bruin)

By the representability of computable functions there is a term E_0 such that

$$\begin{aligned} \mathsf{E}_0 \ulcorner x \urcorner F &=_\beta F \ulcorner x \urcorner, \\ \mathsf{E}_0 \ulcorner MN \urcorner F &=_\beta F(\mathsf{E}_0 \ulcorner M \urcorner F)(\mathsf{E}_0 \ulcorner N \urcorner F), \\ \mathsf{E}_0 \ulcorner \lambda x . M \urcorner F &=_\beta \lambda x . (\mathsf{E}_0 \ulcorner M \urcorner F_{[\ulcorner x \urcorner \mapsto x]}), \end{aligned}$$

where $F_{[\ulcorner x \urcorner \mapsto x]} = F'x$, with

$$\begin{aligned} F'x \ulcorner x \urcorner &=_\beta x, \\ F'x \ulcorner y \urcorner &=_\beta F \ulcorner y \urcorner, \quad \text{if } y \not\equiv x. \end{aligned}$$

By induction on the structure of $M \in \Lambda$ it can be shown that

$$\mathsf{E}_0 \ulcorner M \urcorner F =_\beta M[x_1 := F \ulcorner x_1 \urcorner, \ldots, x_n := F \ulcorner x_n \urcorner] \tag{2}$$

(simultaneous substitution), where $\{x_1, \ldots, x_n\} = FV(M)$. Now we can take

$$\mathsf{E} \equiv \lambda m . \mathsf{E}_0 m \mathsf{I}.$$

Indeed, for closed M it follows by equation (2) that

$$\mathsf{E} \ulcorner M \urcorner =_\beta \mathsf{E}_0 \ulcorner M \urcorner \mathsf{I} =_\beta M. \qquad \square$$

Using the self-interpreter E it can be shown that certain λ-terms exist without giving details. We first introduce some λ-terms inspired by the language LISP.

5 Definition

$$\begin{aligned} \mathsf{cons} &\equiv \lambda xyz . zxy; \\ \mathsf{nil} &\equiv \lambda xyz . y; \\ \mathsf{null} &\equiv \lambda x . x(\lambda abcd . d). \end{aligned}$$

https://doi.org/10.1017/S0956796800020062 Published online by Cambridge University Press

6 Proposition

(i) $\mathsf{null\ nil} = \lambda xy.x \equiv \mathsf{true}$;

(ii) $\mathsf{null\ (cons\ a\ b)} = \lambda xy.y \equiv \mathsf{false}$.

(iii) Moreover, there exist terms car and cdr such that

$$\mathsf{car\ (cons\ a\ b)} = \mathsf{a};$$
$$\mathsf{cdr\ (cons\ a\ b)} = \mathsf{b}.$$

Proof

(i), (ii). Easy. (iii) Take $\mathsf{car} \equiv \lambda x.x(\lambda ab.a)$ and $\mathsf{cdr} \equiv \lambda x.x(\lambda ab.b)$. □

7 Notation

Write

$$\mathsf{a}:\mathsf{b} \equiv \mathsf{cons\ a\ b};$$
$$\langle\,\rangle \equiv \mathsf{nil};$$
$$\langle x_1, \ldots, x_{n+1}\rangle \equiv x_1 : \langle x_2, \ldots, x_{n+1}\rangle.$$

For example, $\langle a, b\rangle \equiv a:b:\mathsf{nil} \equiv (\mathsf{cons\ a\ (cons\ b\ nil)})$.

The following problem was raised by Dr Wim Vree of the University of Amsterdam.

8 Problem

Does there exist a λ-term F such that for all $n \in \mathbb{N}$ one has

$$F\ulcorner n\urcorner = \lambda x_1 \ldots x_n.\langle x_1, \ldots, x_n\rangle? \qquad (3)$$

Solution

Write $M_n \equiv \lambda x_1 \ldots x_n.\langle x_1, \ldots, x_n\rangle$. Clearly, $\# M_n$ is computable from n, say $\# M_n = g(n)$ with g recursive. Let g be λ-defined by G, say. Then

$$G\ulcorner n\urcorner = \ulcorner g(n)\urcorner = \ulcorner M_n\urcorner.$$

Then $F = \lambda n.\mathsf{E}(Gn)$ satisfies equation (3)

$$F\ulcorner n\urcorner = \mathsf{E}(G\ulcorner n\urcorner) = \mathsf{E}\ulcorner M_n\urcorner = M_n. \qquad \square$$

At first, Vree thought the answer to Problem 8 was negative. After seeing the positive answer, he came up with a more constructive solution.

Constructive solution

One can find a λ-term rev such that for all n

$$\mathsf{rev}\,\langle x_1, \ldots, x_n\rangle = \langle x_n, \ldots, x_1\rangle.$$

(For example, $\mathsf{rev} = \lambda L_1.\mathsf{rev}' L_1 \langle\,\rangle$,

with $\mathsf{rev}'\,(a:b)\,L_2 = \mathsf{rev}'\,b\,(a:L_2)$

$\mathsf{rev}'\ \mathsf{nil}\ L_2 = L_2$.

So take $\mathsf{rev}' \equiv \mathsf{Y}(\lambda r L_1 L_2.\mathsf{if}[\mathsf{null}\ L_1]\,\mathsf{then}\,[L_2]\,\mathsf{else}\,[r\,(\mathsf{cdr}\ L_1)\,((\mathsf{car}\ L_1):L_2)])$, where Y is the fixed point combinator and if X then Y else Z is simply XYZ.)

https://doi.org/10.1017/S0956796800020062 Published online by Cambridge University Press

Construct a λ-term V such that

$$V\ulcorner n+1\urcorner = \lambda Lx.(V\ulcorner n\urcorner(x:L)),$$
$$V\ulcorner 0\urcorner = \mathsf{rev}.$$

(For example, $V = Y(\lambda vn.\mathsf{if}[\mathsf{zero?}\,n]\,\mathsf{then}\ \mathsf{rev}\ \mathsf{else}\ [\lambda Lx.v(\mathsf{pred}\,n)(x:L)])$, where pred represents the predecessor function.)

Then $F = \lambda n.V n\,\mathsf{nil}$ satisfies equation (3). Indeed

$$\begin{aligned} F\ulcorner n\urcorner x_1 \ldots x_n &= V\ulcorner n\urcorner\,\mathsf{nil}\,x_1 \ldots x_n \\ &= (\lambda Lx.V\ulcorner n-1\urcorner(x:L))\,\mathsf{nil}\,x_1 \ldots x_n \\ &= V\ulcorner n-1\urcorner(x_1:\mathsf{nil})\,x_2 \ldots x_n \\ &= (\lambda Lx.V\ulcorner n-2\urcorner(x:L))\,(x_1:\mathsf{nil})\,x_2 \ldots x_n \\ &= V\ulcorner n-2\urcorner(x_2:x_1:\mathsf{nil}))\,x_3 \ldots x_n \\ &\ldots \\ &= V\ulcorner 0\urcorner(x_n:\ldots:x_1:\mathsf{nil}) \\ &= \mathsf{rev}\,\langle x_n, \ldots, x_1\rangle \\ &= \langle x_1, \ldots, x_n\rangle. \end{aligned}$$

□

A concrete λ-term satisfying equation (3) is the following

$$\begin{aligned} F \equiv \lambda n.&(\lambda ab.b(aab))(\lambda ab.b(aab)) \\ &(\lambda vn.[n(\lambda xyz.y)(\lambda xy.x)] \\ &\quad[\lambda L.(\lambda ab.b(aab))(\lambda ab.b(aab)) \\ &\qquad(\lambda rL_1 L_2.[L_1(\lambda abcd.d)][L_2][r(L_1(\lambda ab.b))(\lambda z.z(L_1(\lambda ab.a))L_2)]) \\ &\qquad L(\lambda xyz.y)] \\ &\quad[\lambda Lx.v(\lambda yz.n(\lambda pq.q(py))(\lambda w.z)(\lambda t.t))(\lambda z.zxL)]) \\ &n(\lambda xyz.y). \end{aligned}$$

9 *Exercises*

(i) Show that there is no λ-term G such that for all $n \in \mathbb{N}$ one has

$$Gx_1 \ldots x_n = \langle x_1, \ldots, x_n\rangle. \tag{4}$$

(ii) Construct a λ-term H such that for all $n \in \mathbb{N}$ one has

$$H\ulcorner n\urcorner x_1 \ldots x_n = \lambda z.zx_1 \ldots x_n. \tag{5}$$

References

Hofstadter, D. R. 1979. *Gödel, Escher, Bach: an Eternal Golden Braid*, Basic Books.
Kleene, S. C. 1936. λ-definability and recursiveness. *Duke Math. J.* 2, 340–353.