Algebra and Geometry of the λ-calculus

Masahiko Sato
Graduate School of Informatics, Kyoto Universiy
Keisuke Nakano
Research Institute of Electrical Communication, Tohoku University

1 Introduction

We study the syntactic structure of traditional λ-calculus, λ, as defined by Curry [2] from algebraic and geometric point of view.

To achieve this, we first define λ as a type system having only one type λ. Note that λ is a type-free system, but can be thought of a type system where every λ-expression is a member of type λ.

Then, we introduce a new type system λ^* which has the mechanism of binding a finite (possibly empty) sequence of variables. We identify such a sequence of variables with an element of the free monoid $\mathbb{Z}^*$. Then, we can naturally define a monoid action of $\mathbb{Z}^*$ on λ^*. We then define two natural bijective transformations between λ and λ^* which are inverses to each other.

Finally, we define another type system V. Geometrically, it is possible to think of V as a subspace of λ^* consisting of canonical representatives of α-equivalence classes of λ^*. We show that β-rule can be defined on V as a binary operation rather than as a substitution operation. We also show that V-calculus can be defined without using ξ-rule.

We claim that V-calculus provides an intrinsic representation of λ-calculus.

2 Preliminaries

We assume that the reader is familiar with the basic concepts of λ-calculus. See [1] and [4] for the details. Without loss of generality, we assume that the set of variables in the λ- and λ^*-calculus is $\mathbb{Z}$, which is the set of integers. The elements of $\mathbb{Z}$ are denoted by i, j, k, x, y, z; in particular, x, y, z are used to denote variables in the λ- and λ^*-calculus. The set of non-negative integers is denoted by $\mathbb{N}$, whose elements are denoted by m, n, p.

The set of sequences of integers is denoted by $\mathbb{Z}^*$, whose elements are denoted by u, v. The $\mathbb{Z}^*$ type is formally defined by

$$\frac{}{() : \mathbb{Z}^*} \qquad \text{and} \qquad \frac{x : \mathbb{Z} \quad u : \mathbb{Z}^*}{xu : \mathbb{Z}^*}$$

where the symbol $()$ denotes the empty sequence. We write a sequence of integers in the form of $(i_1\ i_2\ \ldots\ i_n)$ with $i_1, i_2, \ldots, i_n \in \mathbb{Z}$ for readability even though the above definition gives the form $i_1\ i_2\ \ldots\ i_n\ ()$. The *length* of a sequence u is denoted by $|u|$, e.g., $|(1\ 0\ -2)| = 3$. The set $\mathbb{Z}^*$ naturally forms a free monoid with the empty sequence $()$ as the identity element and the concatenation $(- \cdot -)$ of sequences as the binary operation defined by

$$(() \cdot u) := u \qquad \text{and} \qquad (xu \cdot v) := x(u \cdot v).$$

We define $\overline{n} := (0\ 1\ \ldots\ n-1)$ for $n \in \mathbb{N}$, e.g., $\overline{3} = (0\ 1\ 2)$.

3 λ- and λ^*-calculus

We start with the definition of tradional λ-calculus and then introduce the λ^*-calculus, which provides a natural link between the traditional λ-calculus and our V-calculus. The λ type is defined as follows:

$$\frac{x : \mathbb{Z}}{x : \lambda}\ \text{var} \qquad \frac{K : \lambda \quad L : \lambda}{(K\ L) : \lambda}\ \text{app} \qquad \frac{x : \mathbb{Z} \quad K : \lambda}{\lambda x.K : \lambda}\ \text{abs}$$

A term L is called a λ-*expression* when $L : \lambda$ is derivable. The set of free variables in a λ-expression K, denoted as $FV(K)$, is defined by

$$\begin{aligned} FV(x) &:= \{x\}, \\ FV((K\ L)) &:= FV(K) \cup FV(L), \quad \text{and} \\ FV(\lambda x.K) &:= FV(K) \setminus \{x\}. \end{aligned}$$

The computation rules of λ-calculus are

$$\begin{array}{ll}
\text{(e1)} & \Rightarrow K = K \\
\text{(e2)} & K = L \Rightarrow L = K \\
\text{(e3)} & K = K', K' = K'' \Rightarrow K = K'' \\
\text{(c)} & K = K', L = L' \Rightarrow (K\ L) = (K'\ L') \\
(\xi) & K = L \Rightarrow \lambda x.K = \lambda x.L \\
(\alpha) & K \equiv_\alpha L \Rightarrow K = L \\
(\beta) & K[x := K'] \equiv L \Rightarrow (\lambda x.K\ K') = L
\end{array}$$

where $\equiv_\alpha$ is used for the α-equivalence relation and $K[x := K']$ is the substitution of K' for x in K. The substitution is formally defined by

$$\begin{array}{rll}
x[x := K'] & \equiv_\alpha K' & \\
y[x := K'] & \equiv_\alpha y & \\
(K\ L)[x := K'] & \equiv_\alpha (K[x := K']\ L[x := K']) & \\
(\lambda x.K)[x := K'] & \equiv_\alpha \lambda x.K & \\
(\lambda y.K)[x := K'] & \equiv_\alpha \lambda y.K & \text{if } x \notin FV(K) \\
(\lambda y.K)[x := K'] & \equiv_\alpha \lambda y.K[x := K'] & \text{if } x \in FV(K) \text{ and } y \notin FV(K') \\
(\lambda y.K)[x := K'] & \equiv_\alpha \lambda z.K[y := z][x := K'] & \text{if } x \in FV(K) \text{ and } y \in FV(K')
\end{array}$$

where x and y are distinct variables and z is a fresh variable not occurring in K or K'.

The λ^*-calculus is introduced as a variant of the λ-calculus where abstractions are integrate with the other two constrution rules. The λ^* type is defined as follows:

$$\frac{u : \mathbb{Z}^* \quad x : \mathbb{Z}}{ux : \lambda^*}\ \text{var} \qquad\qquad \frac{u : \mathbb{Z}^* \quad M : \lambda^* \quad P : \lambda^*}{u(M\ P) : \lambda^*}\ \text{app}$$

A term M is called a λ^*-*expression* when $M : \lambda^*$ is derivable. The sequence u in a λ^*-expression of the form ux or $u(M\ P)$ is called the *context* of the λ^*-expression. The *length* of a λ^*-expression M, denoted as $|M|$, is defined by

$$|ux| := |u| \qquad\qquad \text{and} \qquad\qquad |u(M\ P)| := |u|.$$

The abstraction rule of the λ-calculus disappears in the λ^*-calculus because its syntax allows us to bind multiple (posibly zero) variables at once.

The free monoid $\mathbb{Z}^*$ acts on λ^*-expressions. This action will be used to define λ-to-λ^* conversion. We write $u \cdot M$ for the action of u on M and define it by

$$u \cdot (vx) := (u \cdot v)x \qquad \text{and} \qquad u \cdot (v(M\ P)) := (u \cdot v)(M\ P)$$

where the action on the λ^*-expression is shifted onto its context u.

Every λ-expression K can be converted into a λ^*-expression $M = K^*$. The conversion function $(-)^* : \lambda \to \lambda^*$ is defined by

$$\begin{aligned}(x)^* &:= ()x, \\ ((K\ L))^* &:= ()((K)^*\ (L)^*), \text{ and} \\ (\lambda x.K)^* &:= (x) \cdot (K)^*.\end{aligned}$$

Example 3.1. The S combinator is converted as

$$\begin{aligned}(\lambda x.\lambda y.\lambda z.((x\ z)\ (y\ z)))^* &= (x) \cdot (y) \cdot (z) \cdot (((x\ z)\ (y\ z)))^* \\ &= (x) \cdot (y) \cdot (z) \cdot ()(()(()x\ ()z)\ ()(()y\ ()z)) \\ &= (x) \cdot (y) \cdot (z)(()(()x\ ()z)\ ()(()y\ ()z)) \\ &= (x) \cdot (y\ z)(()(()x\ ()z)\ ()(()y\ ()z)) \\ &= (x\ y\ z)(()(()x\ ()z)\ ()(()y\ ()z)) \qquad \square\end{aligned}$$

Conversely, every λ^*-expression M can be converted into a λ-expression $K = M_*$. The conversion function $(-)_* : \lambda^* \to \lambda$ is defined by

$$\begin{aligned}(()y)_* &:= y \\ ((xu)y)_* &:= \lambda x.(uy)_* \\ (()(M\ N))_* &:= ((M)_*\ (N)_*) \\ ((xu)(M\ N))_* &:= \lambda x.(u(M\ N))_*\end{aligned}$$

It is easy to see that the conversion functions $(-)^*$ and $(-)_*$ are inverses to each other. Therefore, all rules of λ-calculus can be defined on λ^*-calculus in a similar way.

In the following, without loss of generality, we assume a *variable convetion* that every free variable in λ- and λ^*-expressions is a negative integer and every bound variable is a non-negative integer.

4 V-calculus

We introduce a new calculus V, which is as expressive as λ and λ^*-calculi but does not require the (α) and (ξ) rules. The idea of V is similar to the de Bruijn level notation (not de Bruijn index notation) [3], in which every bound variable is renamed by the number of abstractions at the outside of the abstraction binding it. The difference from the de Bruijn level notation is that no abstraction is explicitly declared in V.

Definition 4.1. The V type and a funcition $|-| : \mathrm{V} \to \mathbb{N}$ is defined in an inductive-recursion by

$$\frac{x : \mathbb{Z} \quad n : \mathbb{N} \quad x < n}{x^n : \mathrm{V}}\ \mathrm{var} \qquad\qquad \frac{M : \mathrm{V} \quad P : \mathrm{V} \quad |M| \geq n \quad |P| \geq n}{(M\ P)^n : \mathrm{V}}\ \mathrm{app}$$

with

$$|x^n| := n \qquad\qquad |(M\ P)^n| := n.$$

A term M is called a V-*expression* when $M : \mathrm{V}$ is derivable and $|M|$ is called the *level* of M. A V-expression $x^n : \mathrm{V}$ is said to be a *free variable* if $x < 0$; it is said to be a *bound variable* if $x \geq 0$. □

A V-expression of a level n corresponds to a λ^*-expression in which the sequence of bound variables is $\overline{n} = (0\ 1\ \ldots\ n-1)$.

Example 4.2. For comparison, we show representations of a common expression in different notations: λ, λ^*, de Bruijn level, V, and de Bruijn index.

$$\begin{array}{ll}
\lambda x.(\ \ x\ \ \lambda y.\lambda z.(\ \ x\ \ (\ \ y\ \ z\ )\ )\ ) & \lambda \\
(x)(\ ()x\ \ (y\ \ z)(\ ()x\ \ ()(\ ()y\ \ ()z\ )\ )\ ) & \lambda^* \\
(0)(\ ()0\ \ (1\ \ 2)(\ ()0\ \ ()(\ ()1\ \ ()2\ )\ )\ ) & \lambda^* \\
\lambda\ (\ \ 0\ \ \lambda\ \lambda\ (\ \ 0\ \ (\ \ 1\ \ 2\ )\ )\ ) & \text{de Bruijn level} \\
(\ \ 0^1\ \ (\ \ 0^3\ \ (\ \ 1^3\ \ 2^3)^3)^3)^1 & \mathrm{V} \\
\lambda\ (\ \ 0\ \ \lambda\ \lambda\ (\ \ 2\ \ (\ \ 1\ \ 0\ )\ )\ ) & \text{de Bruijn index}
\end{array}$$

The numbering convention of variables in V-expresions is the same as that of de Bruijn levels. Their difference is that V-expressions do not explicitly declare abstractions but each subexpression (which is either a variable or an application) has a superscript as its level to indicate how many abstractions are its outside. □

Every λ^*-expression can be converted into a V-expression using $\alpha : \lambda^* \to \mathrm{V}$ defined as follows:

$$\begin{aligned}
\alpha(()x) &:= x^0 \\
\alpha((uy)x) &:= \begin{cases} |u|^{|u|+1} & \text{if } x = y \\ z^{|u|+1} \quad \text{where } z^{|u|} = \alpha(ux) & \text{if } x \neq y \end{cases} \\
\alpha(u(M\ P)) &:= (\alpha(u \cdot M)\ \alpha(u \cdot P))^{|u|}
\end{aligned}$$

The 'where' clause in the case of $(uy)x$ with $x \neq y$ manages the fact that the result of $(u)x$ has the form of $z^{|u|}$. It can be shown that $M \equiv_\alpha N$ if and only if $\alpha(M) = \alpha(N)$ for λ^*-expressions M and N. This is why the V-calculus does not require the (α) rule.

Conversely, every V-expression can be converted into a λ^*-expression using $\iota : \mathrm{V} \to \lambda^*$ defined as follows:

$$\begin{aligned}
\iota(x^n) &:= \overline{n}x \\
\iota(M_1\ M_2)^n &:= \overline{n}(N_1\ N_2) \qquad \text{where } \overline{n} \cdot N_1 = \iota(M_1) \text{ and } \overline{n} \cdot N_2 = \iota(M_2)
\end{aligned}$$

The 'where' clause in the case of $(M_1\ M_2)^n$ manages the fact that for each $i = 1, 2$, the result of $\iota(M_i)$ has at least length n so that its context has a prefix $\overline{n}$. We have proven the following theorem in the Rocq theorem prover[1] to guarantee the relationship between α and ι.

Theorem 4.3. For every V-expression M, $\alpha(\iota(M)) = M$ holds.

A V-expression $(M\ P)^n$ is a β-redex when $|M| > n$, in which case the variable n is expected to be replaced by the V-expression P (with modification of its bound variables). We define $[-\ -]^n : \mathrm{V} \times \mathrm{V} \to \mathrm{V}$ so that $[M\ P]^n$ would be the result of β-reduction at $(M\ P)^n$. Since the first argument $M : \mathrm{V}$ should have a level greater than n, there exists $m : \mathbb{N}$ such that $|M| = m + n + 1$. The definition of $[-\ -]^n$ is given by

$$\begin{aligned}
[y^{m+n+1}\ P]^n &:= \begin{cases} y^{m+n} & \text{if } y < n \\ (P)_n^{[m]} & \text{if } y = n \\ (y-1)^{m+n} & \text{if } y > n \end{cases} \\
[(M\ N)^{m+n+1}\ P]^n &:= ([M\ P]^n\ [N\ P]^n)^{m+n}
\end{aligned}$$

[1]It is formerly known as the Coq proof assistant.

with an auxiliary function $(-)_n^{[m]} : \mathrm{V} \to \mathrm{V}$ is defined by

$$(z^p)_n^{[m]} := \begin{cases} z^{p+m} & \text{if } z < n \\ (z+m)^{p+m} & \text{if } z \geq n \end{cases}$$
$$((P\ Q)^p)_n^{[m]} := ((P)_n^{[m]}\ (Q)_n^{[m]})^{p+m}.$$

The function $[M\ P]^n$ decrements every bound variable in M that is greater than n by 1 and replaces the variable n in M by P with a 'modification'. The modification of P is done by the function $(P)_n^{[m]}$, which is used to shift every bound variable in P by m that is the relative level of the occurence of the variable n to be replaced.

The validity of the definition of $[-\ -]^n$ is guaranteed by the following theorem, which has been formally proven in Rocq.

Theorem 4.4. Let K be a λ-expression that satisfies a variable convention and that contains a β-redex of the form $(\lambda x.K_1\ K_2)$ inside n abstractions, and let K' be the result of β-reduction of K at the β-redex. Then the V-expression $\alpha(K^*)$ contains a β-redex of the form $(M_1\ M_2)^n$ and a V-expression obtained by replacing the β-redex by $[M_1\ M_2]^n$ is equal to $\alpha(K'^*)$.

The β-reduction in V-calculus requires just shifting of bound variables and replacing a variable by an V-expression, while the β-reduction in the original λ-calculus requires an involved substitution in which many bound variables should be renamed to avoid capturing as presented in the previous section. In our Rocq formalization, we have defined the β-reduction in the λ-calculus including the substitution operation as well as the β-reduction in the V-calculus with the $[-\ -]^n$ operation. We have shown in Rocq that the two definitions are equivalent in the sense of theorem 4.4.

The V-calculus has the following rules:

$$\begin{array}{ll} \text{(e1)} & M = M \\ \text{(e2)} & M = N \Rightarrow N = M \\ \text{(e3)} & M = N, N = P \Rightarrow M = P \\ \text{(c)} & M = M', N = N' \Rightarrow (M\ N)^n = (M'\ N')^n \\ (\beta) & |M| > n \Rightarrow (M\ P)^n = [M\ P]^n \end{array}$$

The (e1), (e2), and (e3) rules are the same as those of λ-calculus. The (c) rule is similar to that of λ-calculus but the level n is explicitly declared. The (β) rule is defined by a binary operation $[-\ -]^n$ as mentioned above. The α and (ξ) rules are not required in the V-calculus.

5 Conclusion

We have introduced a new calculus V which is as expressive as λ and λ^*-calculi but does not require the (α) and (ξ) rules. We have shown that V-calculus can be defined as a binary operation rather than as a substitution operation. Although V-expressions are similar to the de Bruijn level notation, they do not require the (ξ) rule because no abstraction is explicitly declared in V. Our V-calculus is formalized in Rocq, where all of the theorems given in this manuscript are proven.

References

[1] Hendrik Pieter Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985.

[2] H.B. Curry and R. Feys. *Combinatory Logic, Volume I*. North-Holland, 1958.

[3] N.G de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972.

[4] J. Roger Hindley and Jonathan P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press, USA, 2 edition, 2008.