External and Internal Syntax of the λ-calculus

Masahiko Sato[1]

1 Graduate School of Informatics, Kyoto University

Abstract. It is well known that defining the substitution operation on λ-terms appropriately and establish basic properties like the substitution lemma is a subtle task if we wish to do it formally. The main obstacle here comes from the fact that unsolicited capture of free variables may occur during the substitution if one defines the operation naively.
We argue that although there are several approaches to cope with this problem, they are all unsatisfactory since each of them defines the λ-terms in terms of a single fixed syntax. We propose a new way of defining λ-terms which uses an external syntax to be used mainly by humans and an internal syntax which is used to implement λ-terms on computers.
In this setting, we will show that we can define λ-terms and the substitution operation naturally and can establish basic properties of terms easily.

1 Introduction

There is a growing interest in the study of syntactic structure of expressions equipped with the variable binding mechanism. The importance of this study can be justified for various reasons, including those of educational, scientific and engineering reasons. This study is educationally important since in logic and computer science, we cannot avoid teaching the technique of substitution of higher order linguistic objects correctly and rigorously. Scientific importance is obvious as can be seen from the historical facts that correctly defining the substitution operation was difficult and sometimes resulted in erroneous definitions. Engineering importance comes from recent developments of proof assistance and symbolic computation systems which are increasingly used to assist and verify metamathematical results rather than ordinary mathematical results. We cite here only Aydemir et al. [1] which contains an extensive list of literature on this topic.

We share all those reasons above with other researchers in this field as our motivation to study this subject, but we are especially interested in this subject because of the following ontological question.

> What are *syntactic objects* as objects of mathematical structures with variable binding mechanism?

This is a *semantical* question and cannot be answered by simply manipulating symbols syntactically. To answer this question, we have to study syntax semantically. Our contribution in this paper is precisely the result of such a study.

We have already contributed in this study in [23–28] by investigating the mathematical structure of symbolic expressions. We think that Frege [8], McCarthy [16, 17], Martin-Löf [19, Chapter 3] and Gabbay-Pitts [9, 10] contributed very much in the semantical study of syntax. Our work which we report here is influenced by these works and in particular by the works of Frege and Gabbay-Pitts.

Frege not only formulated syntax of a logical language with binders for the first time, he also formulated it by using two disjoint sets of variables, one for *global variables* using Latin letters and the other for *local variables* using German letters [15, page 25]. Later, Gentzen [11], for instance, followed this approach, but traditionally both logic and the λ-calculus have been formulated using only one sort of variables including Gödel [12] and Church [5] perhaps because of the influence of Whitehead and Russell [33]. McCarthy contributed to semantical understanding of syntactic objects by introducing Lisp symbolic expressions [16] and by introducing the concept *abstract syntax* in [17]. He introduced the term 'abstract syntax' by providing functions to analyze and synthesize *syntactic objects* hiding details of concrete representations of these syntactic objects. This approach works well for languages without variable binding mechanism, but it was difficult to provide abstract syntax (in McCarthy's sense[1]) for languages with binders until Gabbay and Pitts [9, 10] invented nominal technique which implemented abstraction using Fraenkel-Mostowski set theory. They utilized the *equivariance property* which holds in FM-set theory over an abstract set of atoms to deal with α-equivalence and abstraction mechanism on languages with binders having explicit variable names (rather than languages with nameless variables based, for instance, on de Bruijn indices).

Our approach is similar to Gabbay-Pitts' in the sense that the equivariance property holds for our languages, but, unlike their case, we work in standard mathematics and develop our theory by introducing a new notion of *B-algebra* ('B' is for 'binding') which is an algebra equipped with the mechanism of variable binding. For a set $\mathbb{X}$ of atoms, we can introduce the set $\mathbb{S}[\mathbb{X}]$ of *symbolic expressions* over $\mathbb{X}$ as a free B-algebra freely generated from $\mathbb{X}$.

A standard method of defining λ-terms (with explicit names for bound variables) goes as follows. First the set Λ of λ-terms is inductively defined as the smallest set satisfying the set equation $\Lambda = \mathbb{X} + \Lambda \times \Lambda + \mathbb{X} \times \Lambda$ where $\mathbb{X}$ is a given set of variables. Unfortunately it is not possible to define substitution operation on this data structure in a meaningful way due to the possibility of variable capture. To get out of this situation, the α-equivalence relation $=_\alpha$ is defined, and various notions and properties of λ-terms are established by *identifying* α-equivalent terms. However, as pointed out by McKinna-Pollack [18], Pitts [20], Urban [31], Vestargaard [32] etc., that we have to work modulo α-equivalence creates many technical difficulties when we reason about properties of λ-terms by structural induction on λ-terms.

[1] The term 'abstract syntax' used in 'HOAS (Higher Order Abstract Syntax)' has different sense. For this reason, structural induction/recursion works for syntactic objects described by abstract syntax in McCarthy's sense but not in HOAS.

We wish to solve this problem by proposing a new way of defining λ-terms which uses an external syntax to be used mainly by humans and an internal syntax which is used to implement λ-terms on computers. Our motivation for introducing two kinds of syntax is as follows.

Firstly, we wish to have a syntax which inductively creates the set $\mathbb{L}$ of λ-terms isomorphic to $\Lambda/{=_\alpha}$, since by doing so we can constructively grasp each λ-term through the process of creating the term inductively. Note that in case of λ-terms as elements of $\Lambda/{=_\alpha}$, we cannot grasp each term as above, since although each element of Λ is inductively created, each element of $\Lambda/{=_\alpha}$ is obtained abstractly by *identifying* α-equivalent elements of Λ. We will call the syntax which defines $\mathbb{L}$ *the internal syntax* since it can be easily implemented on a computer.

Secondly, in addition to the internal syntax, we will also introduce *the external syntax* which is intended to be used by humans. The external syntax is the same as the standard syntax of λ-calculus given for example in Barendregt [2] and we use Λ as the set of λ-terms but work modulo $=_\alpha$. We can never avoid having an external syntax, since we need it to read and write λ-terms. So, the question is the choice of an external syntax which is comfortable for humans to use as a medium to talk about abstract but real λ-terms as syntactic objects. We think that for this purpose we are right in choosing the standard syntax as *the* external syntax *provided that* we can work in it comfortably and smoothly. Our approach achieves this by defining a natural *semantic function* $[\![-]\!]$ which maps each λ-term M in the external syntax to a λ-term $[\![M]\!]$ in the internal syntax in such a way that $[\![M]\!] = [\![N]\!]$ iff $M =_\alpha N$.

This paper is organized as follows. In Section 2 we introduce the system $\mathbb{S}$ of symbolic expressions with binding structure. We also introduce a new notion of B-algebra and characterize the set of symbolic expressions as a free B-algebra. We also define substitutions as endomorphisms on $\mathbb{S}$ and point out that permutations (i.e., bijective substitutions) are automorphisms and that the group of permutations naturally acts on $\mathbb{S}$ and endows the equivariance property on $\mathbb{S}$.

In Section 3, we introduce the internal syntax for λ-calculus, and define the set $\mathbb{L}$ of λ-terms as a subset of the free B-algebra $\mathbb{S}$ generated by the set $\mathbb{X}$ of global variables. The internal syntax has two sorts of variables, global and local variables. These two sorts of variables have explicit *names* and hence, in the case of local variables, these names can be used to directly refer to the corresponding binders. In contrast with this, if we use de Bruijn indices [6], local variables become *nameless* and we need the complex mechanism of lifting so that these nameless variables can correctly refer to the corresponding binders. Substitution on $\mathbb{L}$ is defined as B-algebra endomorphism. So, there is no need of renaming of variables while computing substitutions. In this paper, we take up the untyped λ-calculus as a canonical example of linguistic structure with the mechanism of variable binding. The system is canonical as it is well-known since Church [4] that λ-calculus can be used as an implementation language of other languages with binders.

In Section 4, we introduce the external syntax by the standard method using only one sort of variables which are used both as global variables (aka free variables) and local variables (aka bound variables). The set Λ of λ-terms in this syntax is also a subset of the same base set $\mathbb{S}$ we used to define the internal syntax. The main difference of the external syntax from the internal syntax is that in the former syntax only one sort of variables is used while two sorts of variables are used in the latter syntax. This difference comes from our construction of $\Lambda \subset \mathbb{S}$ without using the binding mechanism of the B-algebra $\mathbb{S}$. The external syntax and the internal syntax are naturally related by the semantic surjective function $[\![-]\!] : \Lambda \to \mathbb{L}$ which is homomorphic with respect to the application constructor and collapses α-equality to the equality on $\mathbb{L}$.

Section 5 concludes the paper by comparing our results with Gabbay-Pitts' approach and with that of Aydemir et al. [1], and finally by remarking that the data structure of our internal syntax is isomorphic to those of the representations proposed by Quine [22], Bourbaki [3], Sato and Hagiya [23] and Sato [24, 26].

This paper is a slightly revised and corrected version of Sato [29] which was presented at SCSS 2008.

2 Symbolic expressions

In this section we define the set of *symbolic expressions* as a free algebra equipped with a binary operation and a binding operation, and generated by a denumerably infinite set $\mathbb{X}$ of *atoms*. In the construction, we will also use the set $\mathbb{N}$ of natural numbers (which includes 0) as binders.

We will write '$M : \mathbb{S}$', '$X : \mathbb{X}$' and '$x : \mathbb{N}$' for the judgments 'M is a symbolic expression', 'X is an atom' and 'x is a natural number' respectively, and define the set $\mathbb{S}$ of symbolic expressions over $\mathbb{X}$ by the following rules. Since $\mathbb{S}$ is defined depending on $\mathbb{X}$, we will write '$\mathbb{S}[\mathbb{X}]$' for $\mathbb{S}$ when we wish to emphasize the dependency. Atoms will also be called *global variables* and natural numbers will also be called *local variables*. We will use letters 'X', 'Y', 'Z' for global variables, 'x', 'y', 'z' for local variables, and 'M', 'N', 'P', 'Q' for symbolic expressions and later elements of B-algebras.

$$\frac{X : \mathbb{X}}{X : \mathbb{S}} \qquad \frac{x : \mathbb{N}}{x : \mathbb{S}} \qquad \frac{M : \mathbb{S} \quad N : \mathbb{S}}{(M\ N) : \mathbb{S}} \qquad \frac{x : \mathbb{N} \quad M : \mathbb{S}}{[x]\,M : \mathbb{S}}$$

The expression '$(M\ N)$' is said to be *the pair of M and N*. The expression '$[x]\,M$' is said to be *the abstraction by x of M*, 'x' is said to be *the binder of this expression* and 'M' is said to be *the scope of the binder 'x'*. The above definition of symbolic expressions reflects our idea that local variables may get bound by a binder but global variables should never get bound.

We will use the domain of symbolic expressions as our universe of discourse in the rest of the paper. It is possible to directly capture the structural inductive structure of the universe, but here, we introduce the *birthday function* $|-| : \mathbb{S} \to \mathbb{N}$ which we think foundationally more basic, as follows.

1. $|X| \triangleq 1$.
2. $|x| \triangleq 1$.
3. $|(M\ N)| \triangleq \max(|M|, |N|) + 1$.
4. $|[x]\,M| \triangleq |M| + 1$.

The birthday function is defined by reflecting our ontological view of mathematical objects according to which each mathematical object must be constructed by applying a *constructor function* to already constructed objects. By assigning the birthday of a symbolic expression as above, we can see that all the four rules we used in our formation rules of symbolic expressions do enjoy this property. The construction, therefore, proceeds as follows. We observe that among the four rules of symbolic expressions, the first two are unary constructors and the last two are binary constructors. We assume that we have no symbolic expressions on day 0 but global variables and local variables are already constructed so that we have them all on day 0. So, on day 1, only the first two constructors are applicable. Hence, on day 1, all the global and local variables are recognized as symbolic expressions. On day 2, all the four rules are applicable, but only the last two rules produce new symbolic expressions, and they are: $(M\ N)$ where M, N are both variables, or $[x]\,M$ where x is a local variable and M is a variable, be it global or local. The construction of symbolic expressions continues in this way day by day, and every symbolic expression shall be born on its birthday.

This construction suggests the following induction principle which can be used to establish general properties about symbolic expressions:

$$\frac{(\forall N : \mathbb{S}.\, |N| < |M| \Rightarrow \Phi(N)) \Rightarrow \Phi(M)}{\Phi(M)}\ .$$

By using this rule, we can see the validity of the following structural induction rule.

If we can derive the following four judgments

1. $\forall X : \mathbb{X}.\, \Phi(X)$,
2. $\forall x : \mathbb{N}.\, \Phi(x)$,
3. $\forall M, N : \mathbb{S}.\, \Phi(M) \wedge \Phi(N) \Rightarrow \Phi((M\ N))$,
4. $\forall x : \mathbb{N}.\, \forall M : \mathbb{S}.\, \Phi(M) \Rightarrow \Phi([x]\,M)$,

then we may conclude the judgment: $\forall M : \mathbb{S}.\, \Phi(M)$.

With each symbolic expression M we assign a set $\mathrm{LV}(M)$ called *the set of free local variables in* M and a set $\mathrm{GV}(M)$ called *the set of global variables in* M as follows.

1. $\mathrm{LV}(X) \triangleq \{\}$.
2. $\mathrm{LV}(x) \triangleq \{x\}$.
3. $\mathrm{LV}((M\ N)) \triangleq \mathrm{LV}(M) \cup \mathrm{LV}(N)$.
4. $\mathrm{LV}([x]\,M) \triangleq \mathrm{LV}(M) - \{x\}$.

Note: In general, the binder x of an expression $\texttt{[}x\texttt{]}\,M$ may contain x in M again as a binder. In fact, $\texttt{[}x\texttt{]}\,\texttt{[}x\texttt{]}\,x$ is an example of such a case, and in this case we consider that the right-most occurrence of 'x' is bound by the inner binder and not by the left-most binder. We will say that x *occurs free* in M if $x \in \mathrm{LV}(M)$.

1. $\mathrm{GV}(X) \triangleq \{X\}$.
2. $\mathrm{GV}(x) \triangleq \{\}$.
3. $\mathrm{GV}(\texttt{(}M\ N\texttt{)}) \triangleq \mathrm{GV}(M) \cup \mathrm{GV}(N)$.
4. $\mathrm{GV}(\texttt{[}x\texttt{]}\,M) \triangleq \mathrm{GV}(M)$.

Note: The above definition reflects our idea that global variables are never to be bound. We will say that X *occurs* in M if $X \in \mathrm{GV}(M)$.

It is possible to characterize the set $\mathbb{S}$ algebraically by introducing the notion of B-algebra ('B' is for 'binding'). A *B-algebra* is a triple

$$\langle A, \texttt{()} : A \times A \to A, \texttt{[]} : \mathbb{N} \times A \to A \rangle$$

where A is a set which contains $\mathbb{N}$ as its subset. A *magma* (also called a groupoid) is an algebraic structure equipped with a single binary operation, and the notion of B-algebra introduced here is derived from this notion of magma. A B-algebra is a magma equipped with an additional binding operation.

Note: The notion of B-algebra is different from the notion of binding algebra introduced in Firore et al. [7, Section 2]. While our B-algebra has an explicit binding operation $\texttt{[}x\texttt{]}\,M$ which can bind any $x \in \mathbb{N}$ in any $M \in A$, a binding algebra does not have such an explicit algebraic operation of abstraction. Instead, a binding algebra presupposes the existence of the objects obtained by variable binding and operate on these objects.

A *B-algebra homomorphism* is a function h from a B-algebra A to a B-algebra B such that $h(x) = x$, $h(\texttt{(}M\ N\texttt{)}) = \texttt{(}h(M)\ h(N)\texttt{)}$ and $h(\texttt{[}x\texttt{]}\,M) = \texttt{[}x\texttt{]}\,h(M)$ hold for all $M, N \in A$ and $x \in \mathbb{N}$. It is then easy to see that

$$\langle \mathbb{S}[\mathbb{X}], \texttt{()} : \mathbb{S} \times \mathbb{S} \to \mathbb{S}, \texttt{[]} : \mathbb{N} \times \mathbb{S} \to \mathbb{S} \rangle$$

is a *free* B-algebra with the free generating set $\mathbb{X}$. In fact, let B be an B-algebra and consider any $\rho : \mathbb{X} \to B$. Then this ρ can be uniquely extended to a B-algebra homomorphism $[\rho] : \mathbb{S}[\mathbb{X}] \to B$ as follows:

1. $[\rho]X \triangleq \rho(X)$.
2. $[\rho]x \triangleq x$.
3. $[\rho]\texttt{(}M\ N\texttt{)} \triangleq \texttt{(}[\rho]M\ [\rho]N\texttt{)}$.
4. $[\rho]\texttt{[}x\texttt{]}\,M \triangleq \texttt{[}x\texttt{]}\,[\rho]M$.

Here, we are interested in the case where B is $\mathbb{S}$ and $\rho : \mathbb{X} \to \mathbb{S}$ is a finite map. We will call such a map *a finite simultaneous substitution*, or simply *a substitution*. If ρ sends X_i to P_i ($1 \leq i \leq n$, and X_i are distinct) and fixes the rest, $[\rho] : \mathbb{S} \to \mathbb{S}$ is an endomorphism and we will write '$[P_i/X_i]M$' for $[\rho]M$ and call it '*the result of (simultaneously) substituting P_i for X_i in M*'. The substitution operation satisfies the following equations.

1. $[P_i/X_i]X = \begin{cases} P_i & \text{if } X = X_i \text{ for some } i, \\ X & \text{if } X \neq X_i \text{ for all } i. \end{cases}$
2. $[P_i/X_i]x = x$.
3. $[P_i/X_i](M\ N) = ([P_i/X_i]M\ [P_i/X_i]N)$.
4. $[P_i/X_i][x]M = [x][P_i/X_i]M$.

It should be noted that, since substitution is an endomorphism, the substitution operation commutes with the operations of B-algebra smoothly. We note that if ρ and σ are substitutions, then their composition $\rho \circ \sigma$ is also a substitution satisfying the identity $[\rho \circ \sigma]M = [\rho][\sigma]M$. This is a useful property of substitutions as first-class objects.

An endoporhism $[\rho]$ becomes an automorphism if and only if ρ is a permutation, that is, the image of ρ is $\mathbb{X}$ and $\rho : \mathbb{X} \to \mathbb{X}$ is a bijecton. We write '$G_{\mathbb{X}}$' for the group of finite permutations on $\mathbb{X}$. The group $G_{\mathbb{X}}$ naturally acts on the B-algebra $\mathbb{S}[\mathbb{X}]$ by defining the group action of $\pi \in G_{\mathbb{X}}$ on M as $[\pi]M$. In particular, we have $[/]M = M$ and $[\pi \circ \sigma]M = [\pi][\sigma]M$. When $\pi = X, Y/Y, X$ is a transposition which transposes X and Y, we will write '$X /\!\!/ Y$' for π. A transposition is its own inverse since we have $[X /\!\!/ Y] \circ [X /\!\!/ Y] = [X, Y/Y, X] \circ [X, Y/Y, X] = [X, Y/X, Y] = [/]$. For each $\pi \in G_{\mathbb{X}}$ the group action $[\pi](-)$ determines a B-algebra automorphism on $\mathbb{S}[\mathbb{X}]$.

We can apply the general notion of *equivariance* to the group $G_{\mathbb{X}}$. Suppose that $G_{\mathbb{X}}$ acts on two sets U, V and consider a map $f : U \to V$. The map f is said to be an *equivariant map* if f commutes with all $\pi \in G$ and $u \in U$, namely, $f([\pi]u) = [\pi]f(u)$. An equivariant map for an n-ary function can be defined similarly. For example, let $P : U \times V \to \mathbb{B}$ be a binary relation whose values are taken in the set $\mathbb{B} = \{\mathtt{t}, \mathtt{f}\}$ of truth values and define the action of $G_{\mathbb{X}}$ on $\mathbb{B}$ to be a trivial one which fixes the two truth values. Then, that P is an equivariant map means that $P([\pi]u, [\pi]v) = P(u, v)$ holds for all $u \in U, v \in V$ and $\pi \in G_{\mathbb{X}}$. This means that an equivariant relation preserves the validity of the relation under permutations, and for this reason, we may call an equivariant relation *an equivariance*. Thus, the action of $G_{\mathbb{X}}$ provides a useful tool for establishing properties about symbolic expressions since all the statements we make about symbolic expressions enjoy *the equivariance property*. Importance of the notion of equivariance in the abstract treatment of syntax seems to be first emphasized by Gabbay and Pitts [9, 20]. We will apply the notion of equivariance in Section 3 and in Section 4, Theorem 3.

We need to define another form of substitution operation on $\mathbb{S}$ which substitutes a symbolic expression P for *free* occurrences of a local variable y in M. We will write '$[P/y]M$' for the result of the operation and define it as follows.

1. $[P/y]X \triangleq X$.
2. $[P/y]x \triangleq \begin{cases} P & \text{if } x = y, \\ x & \text{if } x \neq y. \end{cases}$
3. $[P/y](M\ N) \triangleq ([P/y]M\ [P/y]N)$.
4. $[P/y]\,[\![x]\!]\,M \triangleq \begin{cases} [\![x]\!]\,M & \text{if } x = y, \\ [\![x]\!]\,[P/y]M & \text{if } x \neq y. \end{cases}$

Note that $[P/y]$ is a function from $\mathbb{S}$ to $\mathbb{S}$ but, unless P is y, it is not a B-algebra homomorphism since it neither preserves y nor commutes with the abstraction operation $[\![y]\!](-)$.

The intended meaning of the fourth clause of the above definition is as follows. According to our definition of the substitution $[P/y]$ we have $[P/y]\,[\![x]\!]\,M = [\![x]\!]\,M$ if $x = y$. This is natural since $\mathrm{LV}([\![x]\!]\,M)$ does not contain y in this case. If $x \neq y$, then the definition is again natural since it is defined so that the substitution will commute with the abstraction operation. However, it should be noted that, in this case, if P contains free occurrences of x then these occurrences of x will be bound after the substitution. This is an unsolicited situation and known ways to avoid this is either to rename x in $[\![x]\!]\,M$ or to rename x in P. The first way is so called α-renaming and the second is called lifting. In Section 3, we will introduce a third way in which we only consider a subset of $\mathbb{S}$ which is rich enough to define λ-terms and at the same time does not create this unsolicited situation. The third way solves the problem by not creating the problem. The height function we define below plays an important role in achieving this.

We can readily show, by induction on the construction of M, the following useful lemmas. We note in passing that, although we can prove it inductively, the Permutation Lemma below follows as an instance of equivariance which says that the substitution function commutes with the action of permutations.

Lemma 1 (GV Lemma). $\mathrm{GV}([P/X]M) \subseteq (\mathrm{GV}(M) - \{X\}) \cup \mathrm{GV}(P)$.

Lemma 2 (Permutation Lemma). *If π is a finite permutation on $\mathbb{X}$, then we have $[\pi][P/Y]M = [[\pi]P/[\pi]Y][\pi]M$.*

Lemma 3 (Substitution Lemma). *If $X \neq Y$ and $X \notin \mathrm{GV}(Q)$, then we have $[Q/Y][P/X]M = [[Q/Y]P/X][Q/Y]M$.*

We conclude this section by defining the *height function* $\mathrm{H} : \mathbb{X} \times \mathbb{S} \to \mathbb{N}$ which will play a crucial role in our development of the internal syntax.

1. $\mathrm{H}_X(Y) \triangleq \begin{cases} 1 & \text{if } X = Y, \\ 0 & \text{if } X \neq Y. \end{cases}$
2. $\mathrm{H}_X(x) \triangleq 0$.
3. $\mathrm{H}_X((M\ N)) \triangleq \max(\mathrm{H}_X(M), \mathrm{H}_X(N))$.
4. $\mathrm{H}_X([\![x]\!]\,M) \triangleq \begin{cases} 0 & \text{if } \mathrm{H}_X(M) = 0, \\ \mathrm{H}_X(M) & \text{if } x = 0 \text{ or } \mathrm{H}_X(M) > x, \\ x + 1 & \text{otherwise.} \end{cases}$

We will call $\mathrm{H}_X(M)$ *the height of* X *in* M. We note that $\mathrm{H}_X(M)$ is 0 if and only if X is not used in the construction of M, that is, $X \notin \mathrm{GV}(M)$. If $\mathrm{H}_X(M) = n+1$, then it means that (if we write M in tree form) either $n = 0$ and X occurs as a leaf at least once in the tree and all the paths from the root to X do not go through a non-zero binder, or n is the largest binder among all the binders we encounter if we go down the tree from the root to all the occurrences of X.

We can easily prove the following lemmas.

Lemma 4 (Height Preservation Lemma). *If* $X \neq Y$ *and* $X \notin \mathrm{GV}(Q)$*, then* $\mathrm{H}_X([Q/Y]M) = \mathrm{H}_X(M)$*.*

Lemma 5 (Height Lemma). *If* $x = \mathrm{H}_X(M)$ *and* $x \notin \mathrm{LV}(M)$*, then* $[X/x][x/X]M = M$*.*

Proof. We prove the following stronger proposition by induction on the construction of M.

If $x \geq \mathrm{H}_X(M)$ *and* $x \notin \mathrm{LV}(M)$*, then* $[X/x][x/X]M = M$*.*

1. $M = Y$.
   (a) $Y = X$. In this case we have, $[X/x][x/X]M = [X/x][x/X]X = X = M$.
   (b) $Y \neq X$. In this case we have, $[X/x][x/X]M = [X/x][x/X]Y = Y = M$.
2. $M = y$. In this case, we have $x \neq y$ since $x \notin \mathrm{FV}(y)$. Hence, $[X/x][x/X]M = [X/x][x/X]y = [X/x]y = y = M$.
3. $M = \texttt{(}P\ Q\texttt{)}$. In this case, we have $x \geq \mathrm{H}_X(M) = \max\{\mathrm{H}_X(P), \mathrm{H}_X(Q)\}$. Hence, by induction hypotheses for P and Q, we have $[X/x][x/X]\texttt{(}P\ Q\texttt{)} = \texttt{(}[X/x][x/X]P\ [X/x][x/X]Q\texttt{)} = \texttt{(}P\ Q\texttt{)}$.
4. $M = \texttt{[}y\texttt{]}P$. We have two cases here.
   (a) $x = y$. In this case we have $[X/x][x/X]\texttt{[}y\texttt{]}P = [X/x]\texttt{[}y\texttt{]}[x/X]P = \texttt{[}y\texttt{]}[x/X]P$. We further divide this case into the following two cases.
      i. $\mathrm{H}_X(P) = 0$. In this case we have $\texttt{[}y\texttt{]}[x/X]P = \texttt{[}y\texttt{]}P$.
      ii. $\mathrm{H}_X(P) > 0$. We consider the following two cases.
         A. $y = 0$. In this case we have $\mathrm{H}_X(\texttt{[}y\texttt{]}P) = \mathrm{H}_X(P)$. But, we also have $\mathrm{H}_X(\texttt{[}y\texttt{]}P) \leq x = y = 0$. Hence $\mathrm{H}_X(P) = 0$, which is a contradiction.
         B. $y \neq 0$. Since $\mathrm{H}_X(P) > 0$ and $y \neq 0$, we have $\mathrm{H}_X(\texttt{[}y\texttt{]}P) > y$. But, on the other hand, we have $y = x \geq \mathrm{H}_X(\texttt{[}y\texttt{]}P)$, which is a contradiction.
   (b) $x \neq y$. In this case we have $[X/x][x/X]\texttt{[}y\texttt{]}P = \texttt{[}y\texttt{]}[X/x][x/X]P$. Since $x \neq y$ and $x \notin \mathrm{LV}(\texttt{[}y\texttt{]}P) = \mathrm{LV}(P) \setminus \{y\}$, we have $x \notin \mathrm{LV}(P)$. So, we have $\texttt{[}y\texttt{]}[X/x][x/X]P = \texttt{[}y\texttt{]}P$ by induction hypothesis.

□

Lemma 6 (Freshness Lemma). *If* P *is a symbolic expression such that* $X \notin \mathrm{GV}(P)$*,* N *is a symbolic expression and* x *is a local variable, then* $[N/X][X/x]P = [N/x]P$*.*

3 The internal syntax

In this section, we define the internal syntax for the λ-calculus. The internal syntax is more basic than the external syntax we introduce in Section 4. It is so for the following two reasons. Firstly, each λ-term defined by the internal syntax directly corresponds to a λ-term as an abstract mathematical object. Namely, the equality relation on the λ-terms defined by the internal syntax is the syntactical identity relation, while the equality on the external λ-terms must be defined modulo α-equivalence. Secondly, we can later define the equality relation on external λ-terms by giving an interpretation of them in terms of internal terms. For these reasons, we will find internal λ-terms easier to implement on a computer than external terms.

As the domain for representing the λ-terms of the internal syntax, we use the free B-algebra $\mathbb{S}[\mathbb{X} \cup \{\mathtt{app}, \mathtt{lam}\}]$, where $\mathbb{X}$ is a denumerably infinite set containing neither `app` nor `lam` and disjoint from the set $\mathbb{N}$. We will write '$\mathbb{L}$' for the set of λ-terms in this syntax. Although $\mathbb{L}$ is not a subalgebra of $\mathbb{S}$, it enjoys the nice property of being closed under the substitution operation. Namely, for any $X \in \mathbb{X}$ and $M, N \in \mathbb{L}$, we will have $[N/X]M \in \mathbb{L}$ (Theorem 1).

We define the set $\mathbb{L}$ inductively by the following rules. The judgment '$M : \mathbb{L}$' means that M is a λ-term. We will write '`(app` M N`)`' as an abbreviation of '`(app (`M N`))`'.

$$\frac{X : \mathbb{X}}{X : \mathbb{L}} \qquad \frac{M : \mathbb{L} \quad N : \mathbb{L}}{\mathtt{(app}\ M\ N\mathtt{)} : \mathbb{L}} \qquad \frac{X : \mathbb{X} \quad M : \mathbb{L}}{\mathtt{(lam\ [}x\mathtt{]}[x/X]M\mathtt{)} : \mathbb{L}}\ (*)$$

Note: In the third rule $(*)$, the height of X in M must be x. We see that in case $x = 0$, then the conclusion of the rule becomes `(lam [0]`M`)`.

A λ-term is called an *application* if it is defined by the second rule above, and an *abstract* if defined by the third rule. Each abstract $M =$ `(lam [`x`]`P`)` defines a function $f_M : \mathbb{S} \to \mathbb{S}$ by putting $f_M(N) \stackrel{\triangle}{=} [N/x]P$ for all $N \in \mathbb{S}$. We will write '$M(N)$' for $f_M(N)$ and call it the *instantiation of the abstract* M *by* N.

We explain the notion of equivariance for the set $\mathbb{S} = \mathbb{S}[\mathbb{X} \cup \{\mathtt{app}, \mathtt{lam}\}]$. Here, the equivariance property is the property which reflects the intrinsic internal symmetry of the set $\mathbb{S}$ with respect to the group action $[\pi](-) : G_{\mathbb{X}} \times \mathbb{S} \to \mathbb{S}$ which sends any $M \in \mathbb{S}$ to $[\pi]M \in \mathbb{S}$ where π is any finite permutation on $\mathbb{X}$. Let $\Phi(M)$ be a statement about $M \in \mathbb{S}$. Then the statement has the *equivariance property* if, for any $M \in \mathbb{S}$ and $\pi \in G_{\mathbb{X}}$, $\Phi(M)$ holds if and only $\Phi([\pi]M)$ holds. (See also [10].)

We can see, albeit informally, that all the statements we prove in this paper have the equivariance property as follows. Suppose that we have a derivation D of $\Phi(M)$. We can formalize this derivation in a formal language whose syntax is based on $\mathbb{S}' = \mathbb{S}[\mathbb{X} \cup \{\mathtt{app}, \mathtt{lam}\} \cup \mathbb{C}]$ where $\mathbb{C}$ is a set of constants, such as logical symbols, necessary to formalize our derivation. Then we have $D \in \mathbb{S}'$ and $\Phi(M) \in \mathbb{S}'$. Here, the functionality of the group action is $[\pi](-) : G_{\mathbb{X}} \times \mathbb{S}' \to \mathbb{S}'$ and we have $[\pi]\Phi(M) = \Phi([\pi]M)$. Now, since D proves $\Phi(M)$, we have $[\pi]D$ proves $[\pi]\Phi(M) = \Phi([\pi]M)$ since all the axioms and inference rules of our formalized

system are closed under the group action on $\mathbb{S}'$. For example, the result of group action by $\pi \in G_{\mathbb{X}}$ on the three rules defining the set $\mathbb{L}$ is:

$$\frac{[\pi]X : \mathbb{X}}{[\pi]X : \mathbb{L}} \qquad \frac{[\pi]M : \mathbb{L} \quad [\pi]N : \mathbb{L}}{\texttt{(app } [\pi]M \ [\pi]N\texttt{)} : \mathbb{L}} \qquad \frac{[\pi]X : \mathbb{X} \quad [\pi]M : \mathbb{L}}{\texttt{(lam [}x\texttt{]}[x/[\pi]X][\pi]M\texttt{)} : \mathbb{L}} \ (*)$$

They are all instances of the same rules including the side condition $(*)$ since we have $\mathrm{H}_{[\pi]X}([\pi]M) = \mathrm{H}_X(M)$.

The essential reason for the validity of the equivariance property is the *indistinguishability* of elements in $\mathbb{X}$. Namely, all we know about $\mathbb{X}$ is that it is disjoint from $\mathbb{N}$ and does not contain `app` or `lam`, and hence we are not able to state in our language a property which holds for a particular element of $\mathbb{X}$ but does not hold for some other elements in $\mathbb{X}$. In contrast with this, consider the transposition τ which transposes `app` and `lam`. Then τ induces an automorphism $[\tau]$ on $\mathbb{S}'$, but this automorphism sends a true statement '$\texttt{(app } X \ X\texttt{)} : \mathbb{L}$' to a false statement '$\texttt{(lam } X \ X\texttt{)} : \mathbb{L}$' for any $X \in \mathbb{X}$.

Given any $M \in \mathbb{S}$, we can decide whether $M \in \mathbb{L}$ or not by induction on $|M|$. For example, if M is of the form $\texttt{(lam [}x\texttt{]}N\texttt{)}$, then,

$$\begin{aligned} M \in \mathbb{L} &\iff N = [x/X]P \text{ for } \textit{some } X \text{ and } P \in \mathbb{L} \\ &\iff [X/x]N \in \mathbb{L} \text{ for } \textit{some } X \notin \mathrm{GV}(N) \\ &\iff [X/x]N \in \mathbb{L} \text{ for } \textit{any } X \notin \mathrm{GV}(N). \end{aligned}$$

The last equivalence is an instance of some/any property (Pitts [20]) whose proof we omit here. So, to decide if $M \in \mathbb{L}$, we have only to take an $X \notin \mathrm{GV}(N)$ and decide if $[X/x]N \in \mathbb{L}$. We can decide this, since $|[X/x]N| = |N| < |M|$. The decision for other cases can be made similarly.

We have the following theorems which guarantee that λ-terms are closed under substitution and instantiation.

Theorem 1 (Substitution). *If P, Q are λ-terms and Y is a global variable, then $[Q/Y]P$ is a λ-term.*

Proof. We argue by induction on the birthday of P and prove by the case of the last rule applied to obtain the derivation of $P : \mathbb{L}$. The only nontrivial case is when P is of the form $\texttt{(lam [}x\texttt{]}[x/X]M\texttt{)}$ and it is derived by the rule:

$$\frac{X : \mathbb{X} \quad M : \mathbb{L}}{\texttt{(lam [}x\texttt{]}[x/X]M\texttt{)} : \mathbb{L}} \ .$$

where $x = \mathrm{H}_X(M)$.

In this case, by induction hypothesis, we have $[Q/Y]M : \mathbb{L}$ and since

$$[Q/Y]\texttt{(lam [}x\texttt{]}[x/X]M\texttt{)} = \texttt{(lam [}x\texttt{]}[Q/Y][x/X]M\texttt{)},$$

our goal is to prove:

$$\texttt{(lam [}x\texttt{]}[Q/Y][x/X]M\texttt{)} : \mathbb{L}.$$

Here, we have the following three possible cases.

Case 1: $X = Y$. In this case, we have

$$\texttt{(lam [}x\texttt{]}[Q/Y][x/X]M\texttt{)} = \texttt{(lam [}x\texttt{]}[Q/X][x/X]M\texttt{)} = \texttt{(lam [}x\texttt{]}[x/X]M\texttt{)}.$$

So, our goal becomes $\texttt{(lam [}x\texttt{]}[x/X]M\texttt{)} : \mathbb{L}$, which we already know to hold.

Case 2: $X \neq Y$ and $X \notin \mathrm{GV}(Q)$. In this case, by the Substitution Lemma 3, we have

$$\texttt{(lam [}x\texttt{]}[Q/Y][x/X]M\texttt{)} = \texttt{(lam [}x\texttt{]}[x/X][Q/Y]M\texttt{)}$$

since $[Q/Y]x = x$. Moreover, since $X \neq Y$ and $X \notin \mathrm{GV}(Q)$, we have

$$\mathrm{H}_X([Q/Y]M) = \mathrm{H}_X(M) = x$$

by the Height Preservation Lemma 4. Hence we can apply the rule:

$$\frac{X : \mathbb{X} \quad [Q/Y]M : \mathbb{L}}{\texttt{(lam [}x\texttt{]}[x/X][Q/Y]M\texttt{)} : \mathbb{L}}$$

and obtain the desired result: $[Q/Y]\texttt{(lam [}x\texttt{]}[x/X]M\texttt{)} : \mathbb{L}$.

Case 3: $X \neq Y$ and $X \in \mathrm{GV}(Q)$. In this case, we choose a fresh global variable Z such that $Z \neq X$, $Z \neq Y$, $Z \notin \mathrm{GV}(M)$ and $Z \notin \mathrm{GV}(Q)$. Then, we can easily see that $[x/X]M = [x/Z][Z/X]M$ by the freshness of Z. Hence, by the Substitution Lemma, we have

$$\begin{aligned}[Q/Y][x/X]M &= [Q/Y][x/Z][Z/X]M \\ &= [[Q/Y]x/Z][Q/Y][Z/X]M \\ &= [x/Z][Q/Y][Z/X]M.\end{aligned}$$

So, our goal now becomes:

$$\texttt{(lam [}x\texttt{]}[x/Z][Q/Y][Z/X]M\texttt{)} : \mathbb{L}.$$

Since $|[Z/X]M| = |M|$, we have $[Q/Y][Z/X]M : \mathbb{L}$ by induction hypothesis. Moreover we have

$$\mathrm{H}_Z([Q/Y][Z/X]M) = \mathrm{H}_X([Q/Y]M) = \mathrm{H}_X(M) = x.$$

Hence, we can now apply the following rule to obtain the desired goal:

$$\frac{Z : \mathbb{X} \quad [Q/Y][Z/X]M : \mathbb{L}}{\texttt{(lam [}x\texttt{]}[x/Z][Q/Y][Z/X]M\texttt{)} : \mathbb{L}}\,.$$

□

Lemma 7 (Instantiation Lemma). *If X is a global variable, M, N are λ-terms and x is the height of X in M, then* $\texttt{(lam [}x\texttt{]}[x/X]M\texttt{)}(N)$ *is a λ-term.*

Proof. We have $\texttt{(lam [}x\texttt{]}[x/X]M\texttt{)}(N) = [N/x][x/X]M$. Since $\mathrm{LV}(M) = \{\}$, by the Height Lemma 5, we have $[X/x][x/X]M = M$ and this implies

$$[N/X][X/x][x/X]M = [N/X]M \tag{1}$$

On the other hand, since $X \notin \mathrm{GV}([x/X]M)$, by the Freshness Lemma 6, we have

$$[N/X][X/x][x/X]M = [N/x][x/X]M \tag{2}$$

Hence, from (1) and (2), we have $[N/x][x/X]M = [N/X]M$. This is a λ-term by the Substitution Theorem 1. □

The followig theorem follows immediately from the above lemma.

Theorem 2 (Instantiation). *If* $\texttt{(lam}\ M\texttt{)}$ *and* N *are* λ*-terms, then so is* $\texttt{(lam}\ M\texttt{)}(N)$.

We are now ready to define the $\lambda\beta$-calculus on the set $\mathbb{L}$ of λ-terms. First we have the following β-reduction rule.

$$\frac{\texttt{(lam}\ M\texttt{)} : \mathbb{L} \quad N : \mathbb{L}}{\texttt{(app (lam}\ M\texttt{)}\ N\texttt{)} \to_\beta \texttt{(lam}\ M\texttt{)}(N)}$$

We recall that $\texttt{(lam}\ M\texttt{)}(N)$ is the instantiation of $\texttt{(lam}\ M\texttt{)}$ by N. Since $\texttt{(lam}\ M\texttt{)} : \mathbb{L}$ implies that M is of the form $\texttt{[}x\texttt{]}P$, we have $\texttt{(lam}\ M\texttt{)}(N) = [N/x]P$.

The reduction relation $M \to N$ of the $\lambda\beta$-calculus is defined here as the binary relation on $\mathbb{S}$ inductively generated by the following rules.

$$\frac{M \to_\beta N}{M \to N} \qquad \frac{X : \mathbb{X}}{X \to X} \qquad \frac{M \to P \quad N \to Q}{\texttt{(app}\ M\ N\texttt{)} \to \texttt{(app}\ P\ Q\texttt{)}}$$

$$\frac{X : \mathbb{X} \quad M \to N}{\texttt{(lam [}x\texttt{]}[x/X]M\texttt{)} \to \texttt{(lam [}y\texttt{]}[y/X]N\texttt{)}}\ (*) \qquad \frac{M \to N \quad N \to P}{M \to P}$$

Note: The fourth rule $(*)$ may be applied only when the following side condition is met:

The height of X is x in M and y in N.

We need this conditition to ensure that the conclusion of the rule indeed becomes a relation on λ-terms. Since the height of X may be different in M and N we have to use maybe different binders x and y as the binders of M and N.

We have the following lemma which is useful when we compute a β-redex inside the scope of a binder.

Lemma 8. *If* $\texttt{(lam [}x\texttt{]}M\texttt{)}$ *is a* λ*-term,* X, Y *are global variables such that* $X \notin \mathrm{GV}(M)$ *and* $Y \notin \mathrm{GV}(M)$*, and* Q *is a* λ*-term, then* $[Q/X][X/x]M = [Q/Y][Y/x]M$.

Example 1. We give an example of reduction by considering the reduction of a λ-term which corresponds to the λ-term:

$$(\lambda z.\ (\lambda x.\ (\lambda y.\ zy)(xz)))y$$

in traditional notation. In the traditional language, this term is reduced as follows.

$$(\lambda z.\ (\lambda x.\ (\lambda y.\ zy)(xz)))y \to \lambda x.\ (\lambda w.\ yw)(xy) \to \lambda x.\ y(xy)$$

Note that we renamed the bound variable y to w to avoid capturing of the free variable y.

In order to translate this into our λ-term smoothly, we use the following two functions, $app : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$ and $lam : \mathbb{X} \times \mathbb{L} \to \mathbb{L}$ defined by:

- $app(M, N) \stackrel{\triangle}{=} \texttt{(app}\ M\ N\texttt{)}$,
- $lam(X, M) \stackrel{\triangle}{=} \texttt{(lam [}x\texttt{]}[x/X]M\texttt{)}$ where $x = \mathrm{H}_X(M)$.

Now, the above term corresponds to the following λ-term.

$$\begin{aligned}
&app(lam(Z, lam(X, app(lam(Y, app(Z, Y)), app(X, Z)))), Y)\\
&= app(lam(Z, lam(X, app(lam(Y, \texttt{(app}\ Z\ Y\texttt{)}), \texttt{(app}\ X\ Z\texttt{)}))), Y)\\
&= app(lam(Z, lam(X, app(\texttt{(lam [1](app}\ Z\ 1\texttt{))}, \texttt{(app}\ X\ Z\texttt{)}))), Y)\\
&= app(lam(Z, \texttt{(lam [1](app (lam [1](app}\ Z\ 1\texttt{)) (app}\ 1\ Z\texttt{))))}), Y)\\
&= app(\texttt{(lam [2](lam [1](app (lam [1](app 2 1)) (app 1 2))))}, Y)\\
&= \texttt{(app (lam [2](lam [1](app (lam [1](app 2 1)) (app 1 2)))) }Y\texttt{)}
\end{aligned}$$

We can compute this term as follows.

$$\begin{aligned}
&\texttt{(app (lam [2](lam [1](app (lam [1](app 2 1)) (app 1 2)))) }Y\texttt{)}\\
&\to \texttt{(lam [1](app (lam [1](app }Y\ 1\texttt{)) (app 1 }Y\texttt{)))}\\
&= lam(X, \texttt{(app (lam [1](app }Y\ 1\texttt{)) (app }X\ Y\texttt{)))}\\
&\to lam(X, \texttt{(app }Y\ \texttt{(app }X\ Y\texttt{)))}\\
&= \texttt{(lam [1](app }Y\ \texttt{(app 1 }Y\texttt{)))}
\end{aligned}$$

We will use the functions *app* and *lam* in the next section to interpret λ-terms in the external syntax by the internal language. □

4 The external syntax

The data structure of the external syntax we introduce in this section is essentially the same as that of the traditional syntax of λ-terms with named variables. In our formulation of the external syntax we will use only global variables and will not use local variables. Also we do not use the binding structure of B-algebra. The mathematical structure of the external syntax is a simple binary tree structure, and as a price for the simplicity of the structure, the definition

of substitution involving α-renaming is much more complex than that for the internal syntax. So, in this section, we will not directly work in the language of the external syntax, but instead we will introduce various notions indirectly by translating the syntactic objects of the external language into the objects of the internal language.

We use the same set $\mathbb{S} = \mathbb{S}[\mathbb{X} \cup \{\texttt{app}, \texttt{lam}\}]$ of symbolic expressions as the base set for defining the set Λ of λ-terms in the external syntax. The set Λ is defined inductively as follows. We will write '`(lam` X M`)`' for '`(lam (`X M`))`' and will continue to write '`(app` M N`)`' for '`(app (`M N`))`'.

$$\frac{X : \mathbb{X}}{X : \Lambda} \qquad \frac{M : \Lambda \quad N : \Lambda}{\texttt{(app}\ M\ N\texttt{)} : \Lambda} \qquad \frac{X : \mathbb{X} \quad M : \Lambda}{\texttt{(lam}\ X\ M\texttt{)} : \Lambda}$$

In this section, to distinguish λ-terms in the external syntax from λ-terms in the internal syntax, we will call $M \in \Lambda$ a Λ-*term* and $M \in \mathbb{L}$ an $\mathbb{L}$-*term*.

We define an onto function $[\![-]\!] : \Lambda \to \mathbb{L}$ which, for each $M \in \Lambda$, defines its *denotation* $[\![M]\!] \in \mathbb{L}$ as follows.

1. $[\![X]\!] \triangleq X$.
2. $[\![\texttt{(app}\ M\ N\texttt{)}]\!] \triangleq app([\![M]\!], [\![N]\!])$.
3. $[\![\texttt{(lam}\ X\ M\texttt{)}]\!] \triangleq lam(X, [\![M]\!])$.

Note: The surjectivity of $[\![-]\!]$ can be verified by induction on the construction of $M \in \mathbb{L}$.

Our view is that each $M \in \Lambda$ is simply a name of the λ-term $[\![M]\!] \in \mathbb{L}$. It is therefore natural to define notions about M in terms of notions about $[\![M]\!]$. As an example, for any $M \in \Lambda$, we can define $\mathrm{FV}(M)$, *the set of free variables in* M, simply by putting: $\mathrm{FV}(M) \triangleq \mathrm{GV}([\![M]\!])$. After *defining* $\mathrm{FV}(M)$ this way, we can *prove* the following equations which characterize the set $\mathrm{FV}(M)$ in terms of the language of the external syntax.

1. $\mathrm{FV}(X) = \{X\}$.
2. $\mathrm{FV}(\texttt{(app}\ M\ N\texttt{)}) = \mathrm{FV}(M) \cup \mathrm{FV}(N)$.
3. $\mathrm{FV}(\texttt{(lam}\ X\ M\texttt{)}) = \mathrm{FV}(M) - \{X\}$.

A Λ-term M is *closed* if $\mathrm{FV}(M) = \{\}$.

Defining the α-equivalence relation on Λ is also straightforward. Given $M, N \in \Lambda$, we define M and N to be α-*equivalent*, written '$M =_\alpha N$', if $[\![M]\!] = [\![N]\!]$. For example, we have

$$\texttt{(lam}\ X\ \texttt{(lam}\ Y\ \texttt{(app}\ X\ Y\texttt{)))} =_\alpha \texttt{(lam}\ Y\ \texttt{(lam}\ X\ \texttt{(app}\ Y\ X\texttt{)))},$$

since

$$\begin{aligned}
[\![\texttt{(lam}\ X\ \texttt{(lam}\ Y\ \texttt{(app}\ X\ Y\texttt{)))}]\!] &= lam(X, lam(Y, app(X, Y))) \\
&= lam(X, lam(Y, \texttt{(app}\ X\ Y\texttt{)})) \\
&= lam(X, \texttt{(lam [1](app}\ X\ 1\texttt{))}) \\
&= \texttt{(lam [2](lam [1](app 2 1)))}
\end{aligned}$$

and we have the same result for $[\![\texttt{(lam } Y \texttt{ (lam } X \texttt{ (app } Y\ X\texttt{)))}]\!]$.

We now verify the adequacy (see Harper et al. [14]) of our definition of the α-equivalence against the definition of the α-equivalence due to Gabbay and Pitts [10, 20]. Their definition, in our notation, is as follows.

$$\frac{M : \Lambda}{M =_\alpha M} \qquad \frac{M =_\alpha P \quad N =_\alpha Q}{\texttt{(app } M\ N\texttt{)} =_\alpha \texttt{(app } P\ Q\texttt{)}} \qquad \frac{[X/\!\!/Z]M =_\alpha [Y/\!\!/Z]N}{\texttt{(lam } X\ M\texttt{)} =_\alpha \texttt{(lam } Y\ N\texttt{)}}\ (*)$$

The rule $(*)$ may be applied only when $Z \notin \mathrm{GV}(M) \cup \mathrm{GV}(N)$. The adequacy is established by interpreting these rules in our internal syntax and showing the Soundness and Completeness Theorem 3 below, which is preceded by the following lemma.

Lemma 9. *If $M =_\alpha N$, then $\mathrm{H}_X(M) = \mathrm{H}_X([\![M]\!]) = \mathrm{H}_X([\![N]\!]) = \mathrm{H}_X(N)$ for all $X \in \mathbb{X}$.*

Proof. By induction on the derivation of $M =_\alpha N$. □

Theorem 3. *The judgment $M =_\alpha N$ is derivable by using the above rules if and only if $[\![M]\!] = [\![N]\!]$.*

Proof. We show the soundness part by induction on $|M|$. We only consider the third rule. Suppose that $[X/\!\!/Z]M =_\alpha [Y/\!\!/Z]N$ and $Z \notin \mathrm{GV}(M) \cup \mathrm{GV}(N)$. By induction hypothesis, we have $[\![[X/\!\!/Z]M]\!] = [\![[Y/\!\!/Z]N]\!]$. Our goal is to show that $[\![\texttt{(lam } X\ M\texttt{)}]\!] = [\![\texttt{(lam } Y\ N\texttt{)}]\!]$. We have

$$[\![\texttt{(lam } X\ M\texttt{)}]\!] = \mathit{lam}(X, [\![M]\!]) = \texttt{(lam [}x\texttt{]}[x/X][\![M]\!]\texttt{)},$$

and

$$[\![\texttt{(lam } Y\ N\texttt{)}]\!] = \mathit{lam}(Y, [\![N]\!]) = \texttt{(lam [}y\texttt{]}[y/Y][\![N]\!]\texttt{)},$$

where $x = \mathrm{H}_X([\![M]\!])$ and $y = \mathrm{H}_Y([\![N]\!])$. Now, by the freshness of Z and by Lemma 9, we have $x = \mathrm{H}_X([\![M]\!]) = \mathrm{H}_Z([X/\!\!/Z][\![M]\!]) = \mathrm{H}_Z([Y/\!\!/Z][\![N]\!]) = \mathrm{H}_Y([\![N]\!]) = y$. So, letting $z = x = y$, we will be done if we can show that $[x/X][\![M]\!] = [y/Y][\![N]\!]$. This is indeed the case since:

$$\begin{aligned}
&[\![[X/\!\!/Z]M]\!] = [\![[Y/\!\!/Z]N]\!] \\
\Longrightarrow\ &[X/\!\!/Z][\![M]\!] = [Y/\!\!/Z][\![N]\!] \quad \text{(by equivariance)} \\
\Longrightarrow\ &[z/Z][X/\!\!/Z][\![M]\!] = [z/Z][Y/\!\!/Z][\![N]\!] \quad \text{(by freshness of } Z\text{)} \\
\Longrightarrow\ &[X/\!\!/Z][x/X][\![M]\!] = [Y/\!\!/Z][y/Y][\![N]\!] \quad \text{(by Permutation Lemma)} \\
\Longrightarrow\ &[x/X][\![M]\!] = [y/Y][\![N]\!] \quad \text{(by GV Lemma, freshness of } Z\text{)}.
\end{aligned}$$

The completeness part is also proved by induction on $|M|$. We consider only the case where $[\![M]\!] = [\![N]\!]$ is of the form $\texttt{(lam [}z\texttt{]}[z/Z]P\texttt{)}$ with $P \in \mathbb{L}$ and $z = \mathrm{H}_Z(P)$.

In this case, $M = \texttt{(lam}\ X\ M'\texttt{)}$ for some X, M' and $N = \texttt{(lam}\ Y\ N'\texttt{)}$ for some Y, N'. Hence, $[\![M]\!] = \texttt{(lam [}z\texttt{]}[z/X][\![M']\!]\texttt{)}$ and $[\![N]\!] = \texttt{(lam [}z\texttt{]}[z/Y][\![N']\!]\texttt{)}$, so that we have $[z/X][\![M']\!] = [z/Y][\![N']\!]$. Hence, we have

$$\begin{aligned}
& [z/X][\![M']\!] = [z/Y][\![N']\!] \\
\Longrightarrow\ & [X/\!\!/Z][z/X][\![M']\!] = [Y/\!\!/Z][z/Y][\![N']\!] \\
\Longrightarrow\ & [z/Z][\![[X/\!\!/Z]M']\!] = [z/Z][\![[Y/\!\!/Z]N']\!] \\
\Longrightarrow\ & [Z/z][z/Z][\![[X/\!\!/Z]M']\!] = [Z/z][z/Z][\![[Y/\!\!/Z]N']\!] \\
\Longrightarrow\ & [\![[X/\!\!/Z]M']\!] = [\![[Y/\!\!/Z]N']\!] \quad \text{(by Height Lemma)} \\
\Longrightarrow\ & [X/\!\!/Z]M' =_\alpha [Y/\!\!/Z]N' \quad \text{(by induction hypothesis)} \\
\Longrightarrow\ & M =_\alpha N.
\end{aligned}$$

□

We can at once obtain the transitivity of the α-equivalence relation by this theorem. This gives a semantical proof of a syntactical property of Λ-terms.

We now turn to the definition of substitution on Λ-terms. Since we can define substitution only modulo $=_\alpha$, we define substitution not as a function but as a relation

$$[N/X]M \Downarrow P$$

on $\Lambda \times \mathbb{X} \times \Lambda \times \Lambda$ which we read '*the result (modulo $=_\alpha$) of substituting N for X in M is P*'. The substitution relation is defined by the following rules. The fifth rule $(*)$ below may be applied when $Y \notin \mathrm{FV}(P)$.

$$\frac{P : \Lambda}{[P/X]X \Downarrow P} \qquad \frac{P : \Lambda \quad X \neq Y}{[P/X]Y \Downarrow Y} \qquad \frac{[P/X]M \Downarrow M' \quad [P/X]N \Downarrow N'}{[P/X]\texttt{(app}\ M\ N\texttt{)} \Downarrow \texttt{(app}\ M'\ N'\texttt{)}}$$

$$\frac{}{[P/X]\texttt{(lam}\ X\ M\texttt{)} \Downarrow \texttt{(lam}\ X\ M\texttt{)}} \qquad \frac{[P/X]M \Downarrow N \quad X \neq Y}{[P/X]\texttt{(lam}\ Y\ M\texttt{)} \Downarrow \texttt{(lam}\ Y\ N\texttt{)}}\ (*)$$

$$\frac{\texttt{(lam}\ Y\ M\texttt{)} =_\alpha \texttt{(lam}\ Z\ N\texttt{)} \quad [P/Z]\texttt{(lam}\ Z\ N\texttt{)} \Downarrow Q}{[P/X]\texttt{(lam}\ Y\ M\texttt{)} \Downarrow Q}$$

The substitution relation we just defined enjoys the following soundness and completeness theorems.

Theorem 4 (Soundness of Substitution).
If $[N/X]M \Downarrow P$, then $[[\![N]\!]/X][\![M]\!] = [\![P]\!]$.

Theorem 5 (Completeness of Substitution). *If $[N'/X]M' = P'$ in $\mathbb{L}$, then $[N/X]M \Downarrow P$, $[\![M]\!] = M'$, $[\![N]\!] = N'$ and $[\![P]\!] = P'$ for some $N, M, P \in \Lambda$.*

By these theorems, we can see that for any N, X, M we can always find a P such that $[N/X]M \Downarrow P$ and all such Ps are α-equivalent with each other.

We omit the development of $=_{\alpha\beta}$ relation on Λ which is a routine work by now.

5 Conclusion

We have introduced the notion of a B-algebra as a magma with an additional operation of local variable binding, and defined the set $\mathbb{S} = \mathbb{S}[\mathbb{X}]$ of symbolic expressions over a set $\mathbb{X}$ of global variables as the free B-algebra with the free generating set $\mathbb{X}$. This setting allowed us to define (simultaneous) substitutions as endomorphisms on $\mathbb{S}$ and permutations as automorphisms on $\mathbb{S}$. As far as we know, this is the first algebraic formulation of *substitution as homomorphism* applicable to symbolic expressions with a variable binding mechanism.

We conclude the paper by comparing our formulation with that by Gabbay-Pitts [10], that by Aydemir et al. [1] and finally those by Quine [22], Bourbaki [3], Sato-Hagiya [23] and Sato [24].

The formulation by Gabbay-Pitts uses FM-set theory over a set of atoms and atoms play the role of variables when they *implement* λ-terms in FM-set theory. Since FM-set theory is close to standard ZFC-set theory except for the indistinguishability of atoms and failure of the axiom of choice, their construction of λ-terms is set-theoretic and non-constructive, although induction principle for so constructed λ-terms can be introduced and proven to be correct. The λ-terms defined in this way is shown to be isomorphic to the standard λ-terms in Λ modulo α-equivalence. A good point of this formulation is that capture avoiding substitution can be manipulated *rigorously* using arguments similar to standard informal arguments on λ-terms modulo α-equivalence. As pointed out in Section 1, standard informal arguments are often very difficult to formalize rigorously. They use the equivariance property under finite permutations of atoms extensively. Pitts later introduced the notion of *nominal sets* [20, 21] and showed that essentially the same results can be obtained within the framework of standard mathematics.

In contrast with this, our formulation of λ-terms in the internal syntax use two sorts of variables, and define λ-terms constructively by inductive rules of construction. We also use the equivariance property of permutations extensively, but, for us, a permutation is just a special instance of more general notion of the simultaneous substitution. In our setting, substitutions and permutations are endomorphisms and automorphisms on $\mathbb{S}$, respectively, and all the substitutions on λ-terms are always capture avoiding with no need of renaming local variables.

The formulation by Aydemir et al. uses two sorts of variables, one for global variables and the other for local variables just like our internal syntax. However, they use de Bruijn indices for local variables, so that their local variables are *nameless* while ours have explicit names (natural numbers are names!). Their binders do not have names but ours have names. In spite of this difference, substitution of a term for a global variable goes as smoothly as our case since both formulations use two sorts of variables. However, their subsitution operations are not characterized as homomorphisms due to the lack of algebraic structure on their terms.

Another difference concerns the formation rules of abstraction. To explain the difference, we note that our introduction rule of abstracts could equivalently

formulated, in a backward way so to speak, as follows.

$$\frac{X : \mathbb{X} \quad [X/x]M : \mathbb{L}}{\texttt{(lam [}x\texttt{]}\,M\texttt{)} : \mathbb{L}}\ (*)$$

Note: The rule $(*)$ may be applied only if $X \notin \mathrm{GV}(M)$ and $x = \mathrm{H}_X([X/x]M)$.

Although this is a technically correct rule, we must say that this rule is unnatural from our *ontological point of view*. This is because in order to apply this rule and obtain a new λ-term as the result of the application, we must somehow know the very λ-term we wish to construct. As we already stressed in [27], we believe that every *mathematical object*, including of course every λ-term, must be constructed by applying a *constructor* function to *already created objects*. But, this rule does not follow this *ontological condition*, and this is why we did not adopt the above rule but instead adopted the abstraction formation rule in Section 3. Now, if formulated in the style of Aydemir et al. [1], the rule for constructing abstracts in $\mathbb{L}$ would become like this (cf. the TYPING-ABS rule in [1, Figure 1]):

$$\frac{X : \mathbb{X} \quad M^X : \mathbb{L}}{\texttt{(lam}\ M\texttt{)} : \mathbb{L}}\ (**)$$

Just like our rule $(*)$, the rule $(**)$ may be applied only when $X \notin \mathrm{GV}(M)$. In this rule, local variables are represented by de Bruijn indices, and $\lambda x.\, x\lambda y.\, yx$, for instance, becomes

```
(lam (app 0 (lam (app 0 1))))
```

while it becomes

```
(lam [2](app 2 (lam [1](app 1 2))))
```

in our formulation. The term M^X in the second premise of the rule $(**)$ is the *opening up* of M by X which corresponds to our instantiation of `(lam [`x`]`M`)` by X, namely, $[X/x]M$. So continuing our example, opening up by X and instantiation by X, respectively, becomes

`(app` X `(lam (app 0` X`)))` and `(app` X `(lam [1](app 1` X`)))`.

Note that in opening up `(app 0 (lam (app 0 1)))` by X we had to replace 0 by X in one place and 1 by X in another place while `[2](app 2 (lam [1](app 1 2)))` could be instantiated by X just by substituting X for two occurrences of 2.

We may thus say that the representation of λ-terms by the method of [1] is more complex than our method and that what we presume to be their rule for *introducing* a new λ-term is ontologically unnatural as it requires us to *mentally* construct the term beforehand. We note, however, that it is possible to replace the rule $(**)$ with an ontologically natural rule which parallels our rule we gave in Section 3. See 4.5 of Aydemir et al. [1] for such a rule where they examine a rule given by Gordon [13]. They did not adopt this rule for technical reasons.

We also remark that our internal syntax is more human friendly than that of [1] and hence, if we so wish, can be used as an external syntax replacing the external syntax we gave in Section 4.

Finally, we remark that, as a data structure, our representation of expressions with binders is, in a sense, isomorphic to those by Quine [22, page 70], Bourbaki [3, Chapter 1], Sato-Hagiya [23] and Sato [24, 26]. Their representations are nameless since abstraction is realized by providing links between the binding node and the nodes which refer back to the binding node. These representations are usually implemented on a computer by realizing links in terms of pointers. However, except for Sato and Hagiya [23] and Sato [24, 26], these data structures do not admit well-founded induction principle, since these data structures contain *cycles*. Unlike these, our representation admits reasoning by induction on the birthday of each expression, and has a nice algebraic structure.

Acknowledgments

I wish to thank René Vestergaard and Murdoch Gabbay for fruitful discussions on the mechanism of variable binding. We also thank Andrew Pitts for useful comments on an earlier draft of the paper.

References

1. Aydemir, B., Charguéraud, A., Pierce, B.C., Pollack R. and Weirich, S., Engineering Formal Metatheory, In *POPL '08: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles on Programming Languages*, ACM Press, pp. 3–15, 2008.
2. Barendregt, H., *The Lambda Calculus*, North-Holland, 1984.
3. Bourbaki, N., *Theory of Sets*, Hermann, 1968.
4. Church, A., A Formulation of the Simple Theory of Types, *Juornal of Symbolic Logic*, **5**, pp. 56–68, 1940.
5. Church, A., *The Calculi of Lambda Conversions*, Princeton University Press, 1941.
6. de Bruijn, N.G. Lambda Calculus Notation with Nameless Dummies, A Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem, *Indag. Math.*, **34**, pp. 381–392, 1972.
7. Fiore, M., Plotkin, G. and Turi, D., Abstract Syntax and Variable Binding, Proc. 14th Annual IEEE Symposium on Logic in Computer Science, pp. 193–202, 1999.
8. Frege, G., *Begriffsschrift, eine der arithmetishen nachgebildete Formelsprache des reinen Denkens*, Halle, 1879. (English translation in [15])
9. Gabbay, M.J. and Pitts, A.M., A new approach to abstract syntax involving binders, in *14th Annual Symposium on Logic in Computer Science*, pp. 214–224, IEEE Computer Society Press, 1999.
10. Gabbay, M.J. and Pitts, A.M., A new approach to abstract syntax involving binders, *Formal Aspects of Computing*, 2002.
11. Gentzen, G., Untersuchungen über das logische Schließen, I, Mathematische Zeitschrift, **39**, pp. 175–210, 1935, English translation in [30, pp. 68 – 131].
12. Gödel, K., Über formal unentscheidbare Sätze der Pricipia mathematica und verwandter Systeme I, *Monatshefte für Mathematik und Physik*, **38**, pp. 173–198, 1931, English translation in [15].

13. Gordon, A.D., A mechanisation of name-carrying syntax up to alpha-conversion, in Joyce, J.J. and Seger, C.-J.H. (eds.), *Higher-order Logic Theorem Proving and its Applications, Proceedings, 1993*, **780**, Lecture Notes in Computer Science, pp. 414–426, Springer, 1994.
14. Harper, R., Honsell, R. and Plotkin, G., A framework for defining logics, *Journal of the ACM*, **40**, pp. 143–184, 1993.
15. Heijenoort, J.v. (ed.), *From Frege to Gödel, A Source Book in Mathematical Logic, 1879 – 1931*, Harvard University Press, 1977.
16. McCarthy, J., Recursive Functions of Symbolic Expressions and their Computation by Machine (Part 1), *Comm. ACM*, **3**, 184–195, 1960.
17. McCarthy, J., A basis for a mathematical theory of computation, in P. Braffort and D. Hirschberg (eds.), *Computer Programming and Formal Systems*, pp. 33 – 70, North-Holland, 1963.
18. McKinna, J. and Pollack, R., Some Lambda Calculus and Type Theory Formalized, *Journal of Automated Reasoning*, **23**, pp. 373–409, 1999.
19. Nordström, B., Petersson, K. and Smith, J.M., *Programming in Martin-Löf's Type Theory*, Oxford University Press, 1990.
20. Pitts, A.M., Nominal logic: a first order theory of names and binding, *Information and Computation*, **186**, pp. 165–193, 2003.
21. Pitts, A.M., Alpha-Structural Recursion and Induction, J. ACM, **53**, pp. 459 – 506, 2006.
22. Quine, W., *Mathematical Logic (Revised Edition)*, MIT Press, 1951.
23. Sato M. and Hagiya, M., Hyperlisp, *Proceedings of the International Symposium on Algorithmic Language*, 251-269, North-Holland, 1981.
24. Sato M., Theory of Symbolic Expressions, I, *Theoretical Computer Science*, **22**, 19-55, 1983.
25. Sato M., Theory of Symbolic Expressions, II, *Publ. RIMS, Kyoto Univ.*, **21**, 455–540, 1985.
26. Sato M., An Abstraction Mechanism for Symbolic Expressions, 1991, in V. Lifschitz ed., *Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy)*, Academic Press, 381-391, 1991.
27. Sato M., Theory of Judgments and Derivations, in Arikawa, S. and Shinohara, A. eds., *Progress in Discovery Science*, Lecture Notes in Artificial Intelligence **2281**, pp. 78 – 122, Springer, 2002.
28. Sato M., A framework for checking proofs naturally, *Journal of Intelligent Information Systems*, **31**, 111–125, 2008.
29. Sato M., External and Internal Syntax of the λ-calculus, in *Proc. of the Symbolic Computation in Software Science Austrian-Japanese Workshop, SCSS 2008.*
30. Szabo, M.E. (ed.), *The collected papers of Gerhard Gentzen*, North-Holland, 1969.
31. Urban, C., Berghfer, S. and Norrish, M., *Barendregt's Variable Convention in Rule Induction*, in *Proc. of the 21st International Conference on Automated Deduction (CADE)*, Lecture Notes in Artificial Intelligence, **4603**, pp. 35–50, 2007.
32. Vestergaard, R., *The Primitive Proof Theory of the λ-Calculus*, Ph.D Thesis, Heriot-Watt University, 2003.
33. Whitehead, A.N. and Russell, B., *Principia mathematica, vol. 1*, Cambridge University Press, 1910.