Volume 45   Number 5   May 2010   ISSN 0747-7171

Journal of Symbolic Computation

Symbolic Computation in Software Science
Guest Editor: Temur Kutsia
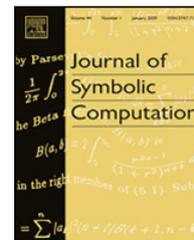
External and internal syntax of the λ-calculus☆

Masahiko Sato [a], Randy Pollack [b,1]

[a] *Graduate School of Informatics, Kyoto University, Japan*

[b] *LFCS, School of Informatics, University of Edinburgh, United Kingdom*

A B S T R A C T

It is well known that formally defining and reasoning about languages with binding (such as logics and λ-calculii) is problematic. There are many approaches to deal with the problem, with much work over recent years stimulated by the desire to formally reason about programming languages and program logics. The various approaches have their own strengths and drawbacks, but no fully satisfactory approach has appeared.

We present an approach based on two levels of syntax: an *internal syntax* which is convenient for machine manipulation, and an *external syntax* which is the usual informal syntax used in many articles and textbooks. Throughout the paper we use pure λ-calculus as an example, but the technique extends to many languages with binding.

Our internal syntax is *canonical*: one representative of every α-equivalence class. It is formalized in Isabelle/HOL, and its properties are mechanically proved. It is also proved to be isomorphic with a nominal representation of λ-calculus in Isabelle/HOL.

Our conventional, human friendly external syntax is naturally related to the internal syntax by a semantic function. We do not define notions directly on the external syntax, since that would require the usual care about α-renaming, but introduce them indirectly from the canonical internal syntax via the semantic function.

© 2010 Elsevier Ltd. All rights reserved.

1. Introduction

There is growing interest in the study of the syntactic structure of expressions equipped with a variable binding mechanism. The importanceof this study can be justified for various reasons,

e.g. 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 assistants and symbolic computation systems which are increasingly used to develop and verify metamathematical results rather than ordinary mathematical results. We cite here only Aydemir et al. (2008) which contains an extensive list of literature on this topic.

We share all these motivations to study this subject, but 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 the result of such a study.

We have already contributed in this study in Sato and Hagiya (1981), Sato (1983, 1985, 1991, 2002, 2008b) by investigating the mathematical structure of symbolic expressions. We think that Frege (1879), McCarthy (1960, 1963), Martin-Löf (Nordström et al., 1990, Chapter 3) and Gabbay and Pitts (1999, 2002) contributed much to the semantic study of syntax. The work which we report here is influenced by these works and in particular by the works of Frege and Gabbay–Pitts. Syntactically our work is a refinement of McKinna and Pollack (1993, 1999).

Frege was the first to formulate the syntax of a logical language with binders. He used two disjoint sets of variables, one for *global variables* using Latin letters and the other for *local variables* using German letters (van Heijenoort, 1967, page 25). Later, Gentzen (1934), for instance, followed this approach. Traditionally, however, logic and λ-calculus have been formulated using only one sort of variables, e.g. Gödel (1931) and Church (1941), perhaps because of the influence of Russell and Whitehead (1910). McCarthy contributed to semantical understanding of syntactic objects by introducing Lisp symbolic expressions (McCarthy, 1960), and the concept *abstract syntax* (McCarthy, 1963), providing functions to analyze and synthesize syntactic objects while hiding the concrete representation of these objects. This approach works well for languages without variable binding, but it was difficult to provide abstract syntax (in McCarthy's sense[2]) for languages with binders until Gabbay and Pitts (1999, 2002) invented nominal techniques which implement abstraction using Fraenkel–Mostowski set theory. This utilizes the *equivariance property* which holds in FM-set theory over an abstract set of atoms to deal with α-equivalence on languages with binders using explicit variable names (as opposed to nameless variables, e.g. de Bruijn indices). Nominal techniques have since been extended to more standard logics (Pitts, 2003; Urban, 2008).

In this paper 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 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 a 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 and Pollack (1999), Pitts (2003), Urban et al. (2007), Vestergaard (2003) etc., working modulo α-equivalence creates many technical difficulties when we reason by structural induction about properties of λ-terms.

---

[2] The term 'abstract syntax' used in 'Higher Order Abstract Syntax' (HOAS) 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 solve this problem by proposing a new way of defining λ-terms using an external syntax mainly for humans, and an internal syntax which implements λ-terms on computers. Our motivation for introducing two kinds of syntax is as follows.

First, 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, and is amenable to inductive reasoning.

Second, in addition to the internal syntax, we introduce *external syntax* which is intended to be used by humans. We can never avoid having an external syntax, since we need to read and write λ-terms; so the problem is to choose an external syntax which is comfortable for humans to use as a medium to talk about abstract but real λ-terms as syntactic objects. For this we choose the standard syntax of λ-calculus given for example in Barendregt (1984), and show how to work in it comfortably and smoothly. This is achieved 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 define substitutions as endomorphisms on $\mathbb{S}$ and point out that permutations of global variables (i.e. bijective substitutions) are automorphisms and that the group of permutations naturally acts on $\mathbb{S}$ and endows the equivariance property on $\mathbb{S}$. This section closes by defining a 'height function' on $\mathbb{S}$ and developing its theory. The point of this function is not explained until Section 3.1. Everything in Section 2 is straightforwardly understandable using induction and recursion over datatypes: no binding occurs.

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. Contrast this with de Bruijn indices (de Bruijn, 1972) where local variables are *nameless* and we need a 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; there is no need of renaming of variables while computing substitutions. In this paper, we take up the untyped λ-calculus as an example of linguistic structure with the mechanism of variable binding. (It is well known since Church (1940) that λ-calculus can be used as an implementation language of other languages with binders.) However richer languages, with several classes of expressions (e.g. types and terms) or more complicated binding structure can easily be accommodated.

In Section 4, we introduce a more conventional external syntax with only one sort of variables which are used both as global (free) variables and local (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. We construct $\Lambda \subset \mathbb{S}$ without using the binding mechanism of the B-algebra $\mathbb{S}$. The external syntax and the internal syntax are naturally related by a surjective semantic function $[\![-]\!] : \Lambda \to \mathbb{L}$ which is homomorphic with respect to the application constructor and collapses α-equality to identity on $\mathbb{L}$.

Section 5 concludes the paper by comparing our results with Gabbay–Pitts' approach and with that of Aydemir et al. (2008), and finally by remarking that the data structure of our internal syntax is isomorphic to those of the representations proposed by Quine (1951), Bourbaki (1968), Sato and Hagiya (1981) and Sato (1983, 1991).

Formalization

Everything in Sections 2 and 3, the development of $\mathbb{S}$, $\mathbb{L}$, and some examples, has been formalized in nominal Isabelle (Urban, 2008). You may download these Isabelle theory files from http://homepages. inf.ed.ac.uk/rpollack/export/SatoPollackIsabelleJSC.tgz. We use a nominal Isabelle atom type for $\mathbb{X}$, and take advantage of convenient automation tools provided by nominal Isabelle. We also show, in

Isabelle, that $\mathbb{L}$ is isomorphic (respecting substitution) with lambda terms as usually represented in nominal Isabelle. Although it is an informal issue whether our formal representations adequately capture the lambda terms in your mind, the fact that two formal representations agree adds to confidence about the faithfulness of both representations.

2. Symbolic expressions

In this section we define the set of *symbolic expressions* as a free algebra generated from a denumerably infinite set $\mathbb{X}$ of *atoms* (also called *global variables*) and from the set $\mathbb{N}$ of natural numbers (which includes 0, and are also called *local variables*). $\mathbb{N}$ is used as a set of names distinct from $\mathbb{X}$. We use '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. We write '$M : \mathbb{S}$' for the judgment 'M is a symbolic expression', but leave the sorts of atoms and natural numbers implicit.

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

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.

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'*. This definition of symbolic expressions reflects our idea that local variables may get bound by a binder but global variables are never bound. Note, however, that there is no indexing or binding explicit in this free construction.

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 nature of the universe, but here, we introduce the *birthday function* $|-| : \mathbb{S} \to \mathbb{N}$ which we think foundationally more basic, as follows.

$$
\begin{aligned}
|X| &\overset{\triangle}{=} 1 \\
|x| &\overset{\triangle}{=} 1 \\
|(M\ N)| &\overset{\triangle}{=} \max(|M|, |N|) + 1 \\
|[x]\,M| &\overset{\triangle}{=} |M| + 1
\end{aligned}
$$

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:

$$\frac{\forall M.\ (\forall N.\ |N| < |M| \implies \Phi(N)) \implies \Phi(M)}{\forall M.\ \Phi(M)}.$$

Using this rule, we can see the validity of the following structural induction rule:

$$\frac{\begin{array}{l} \forall X. \ \Phi(X) \\ \forall x. \ \Phi(x) \\ \forall M, N. \ \Phi(M) \wedge \Phi(N) \implies \Phi((M \ N)) \\ \forall x. \ \forall M. \ \Phi(M) \implies \Phi([x]M) \end{array}}{\forall M. \ \Phi(M)}$$

With each symbolic expression M we assign a set $LV(M)$ called the *free local variables* of M:

$$\begin{aligned} LV(X) &\triangleq \{\} \\ LV(x) &\triangleq \{x\} \\ LV((M \ N)) &\triangleq LV(M) \cup LV(N) \\ LV([x]M) &\triangleq LV(M) - \{x\}. \end{aligned}$$

We say that *x occurs free* in M if $x \in LV(M)$. In general, the body M of an expression $[x]M$ may bind x again, as in $[x][x]x$. In this case we consider that the rightmost occurrence of 'x' is bound by the inner binder; but this is only informal talk.

Similarly, define a set $GV(M)$ called the *global variables* of M:

$$\begin{aligned} GV(X) &\triangleq \{X\} \\ GV(x) &\triangleq \{\} \\ GV((M \ N)) &\triangleq GV(M) \cup GV(N) \\ GV([x]M) &\triangleq GV(M). \end{aligned}$$

We say that *X occurs* in M if $X \in GV(M)$.

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

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

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 Fiore et al. (1999, Section 2). While our B-algebra has an explicit binding operation $[x]M$ 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 operates on these objects.

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

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

is a *free* B-algebra with the free generating set $\mathbb{X}$. In fact, let B be a 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:

$$\begin{aligned} [\rho]X &\triangleq \rho(X) \\ [\rho]x &\triangleq x \\ [\rho](M \ N) &\triangleq ([\rho]M \ [\rho]N) \\ [\rho][x]M &\triangleq [x][\rho]M. \end{aligned}$$

When B is $\mathbb{S}$ and $\rho : \mathbb{X} \to \mathbb{S}$ is a finite map, we call ρ 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.

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

Since substitution is an endomorphism, the substitution operation commutes smoothly with the B-algebra operations.

Notice, however, that substitution is *not* capture avoiding on $\mathbb{S}$, e.g.

$$[y/X]([y]X) = [y]y.$$

This is not the intended behaviour of substitution. Known ways to avoid the difficulty in defining $[P/X]([x]M)$ include renaming x in $[x]M$ or renaming x in P. The former is called α-renaming (e.g. as in Curry and Feys (1958); Stoughton (1988)); the latter is called lifting (e.g. as in de Bruijn (1972)). In Section 3.1 we explain a third way in which we keep the algebraically clean B-algebra substitution defined above, but restrict to a subset of $\mathbb{S}$ in which expressions do not contain free occurrences of *local* variables. This is the historical reason for using distinct species of names for local vs. global variables. See McKinna and Pollack (1993, 1999), Aydemir et al. (2008) for previous modern formalizations using this approach.

An endomorphism $[\rho]$ 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 bijection. 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 $\mathbb{B} = \{\texttt{t}, \texttt{f}\}$ 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*. The importance of the notion of equivariance in abstract treatment of syntax seems to have been first emphasized by Gabbay and Pitts (1999), Pitts (2003). We will apply the notion of equivariance in Section 3 and in Section 4, Theorem 3.

We need to define another operation 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.

$$[P/y]X \overset{\triangle}{=} X$$
$$[P/y]x \overset{\triangle}{=} \begin{cases} P & \text{if } x = y, \\ x & \text{if } x \neq y. \end{cases}$$
$$[P/y](M \ N) \overset{\triangle}{=} ([P/y]M \ [P/y]N)$$
$$[P/y][x]M \overset{\triangle}{=} \begin{cases} [x]M & \text{if } x = y, \\ [x][P/y]M & \text{if } x \neq y. \end{cases}$$

This operation is purely technical: it is needed in our development, but does not correspond to a natural operation on the informal notion of terms. In particular $[P/y]$ 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 $[y](-)$. Also, like substitution, this operation is not capture avoiding.

We can show the following useful lemmas by induction on the construction of M.

Lemma 1 (*Permutation Lemma*). *Both forms of substitution are equivariant: if π is a finite permutation on $\mathbb{X}$, then*

$$[\pi][P/Y]M = [[\pi]P/[\pi]Y][\pi]M \quad and \quad [\pi][P/y]M = [[\pi]P/y][\pi]M.$$

Lemma 2 (*Substitution Lemma*). *If $X \neq Y$ and $X \notin \mathrm{GV}(Q)$, then we have*

$$[Q/Y][P/X]M = [[Q/Y]P/X][Q/Y]M.$$

Lemma 3 (*Substitutions Cancel*). *If $X \notin \mathrm{GV}(M)$ then $M = [x/X][X/x]M$.*

Lemma 4 (*Substitutions Commute*). *If $X \neq Y$ and $x \notin \mathrm{LV}(N)$ then*

$$[Y/x][N/X]M = [N/X][Y/x]M.$$

Now we define 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 (see Section 3.1).

$$\mathrm{H}_X(Y) \triangleq \begin{cases} 1 & \text{if } X = Y, \\ 0 & \text{if } X \neq Y. \end{cases}$$

$$\mathrm{H}_X(x) \triangleq 0.$$

$$\mathrm{H}_X((M \ N)) \triangleq \max(\mathrm{H}_X(M), \mathrm{H}_X(N)).$$

$$\mathrm{H}_X([x]M) \triangleq \begin{cases} \mathrm{H}_X(M) & \text{if } \mathrm{H}_X(M) = 0 \text{ or } \mathrm{H}_X(M) > x, \\ x+1 & \text{otherwise.} \end{cases}$$

We call $\mathrm{H}_X(M)$ the *height of X in M*. H looks like a very special function, but in recent work we have observed that it is just a concrete example of a class of height functions that can be used for our representation. For more details on this point of view see Pollack and Sato (2009). Here we only present (Lemma 5, 6 and 7) the three essential properties that any good height function must satisfy.

Lemma 5 (*Equivariance*). H *is an equivariant function:* $[\pi]\mathrm{H}_X(M) = \mathrm{H}_{[\pi]X}([\pi]M)$.

As a corollary we have $Y \notin \mathrm{GV}(M) \implies \mathrm{H}_X(M) = \mathrm{H}_Y([Y/X]M)$.

Lemma 6 (*Height Preservation*). *If $X \neq Y$ and $X \notin \mathrm{GV}(Q)$, then*

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

Lemmas 5 and 6 are proved by induction on the structure of M.

The third essential property for height functions is that $\mathrm{H}_X(M)$ does not occur in binding position on any path between the root of M and any occurrence of X in M. To express this we define an auxiliary function $\mathrm{E}_X(M) : \mathbb{X} \times \mathbb{S} \to (\mathbb{N} \text{ set})$ computing the set of local names occurring in binding position between the root of M and any occurrence of X in M. The definition (by structural recursion) is straightforward.

$$\mathrm{E}_X(Y) \triangleq \{\}$$

$$\mathrm{E}_X(y) \triangleq \{\}$$

$$\mathrm{E}_X((M \ N)) \triangleq \mathrm{E}_X(M) \cup \mathrm{E}_X(N)$$

$$\mathrm{E}_X([x]M) \triangleq \begin{cases} \{\} & \text{if } X \notin \mathrm{GV}(M) \text{ (no paths to } X \text{ in } M) \\ \{x\} \cup \mathrm{E}_X(M) & \text{if } X \in \mathrm{GV}(M) \text{ (every path contains } x). \end{cases}$$

Lemma 7 (*Freshness of Bound Names*). $\mathrm{H}_X(M) \notin \mathrm{E}_X(M)$.

Proof. The lemma follows from the generalization $x \geq \mathrm{H}_X(M) \implies x \notin \mathrm{E}_X(M)$, which is proved by induction on the structure of M. In case M is an abstraction, notice $X \in \mathrm{GV}(N) \implies \mathrm{H}_X(N) > x$. $\square$

In practice we need the following corollary of Lemma 7.

Lemma 8 (*Height Lemma*). *If $x = H_X(M)$ and $x \notin LV(M)$, then*

$$[N/x][x/X]M = [N/X]M.$$

Proof. This follows from Lemma 7 using the generalization

$$x \notin LV(M) \wedge x \notin E_X(M) \implies ([N/x][x/X]M = [N/X]M)$$

proved by induction on the structure of M.

In case $M = [v]S$ we know (a) $x \notin LV([v]S)$, (b) $x \notin E_X([v]S)$, and the induction hypothesis

$$x \notin LV(S) \wedge x \notin E_X(S) \implies ([N/x][x/X]S = [N/X]S).$$

If $x = v$ we need to show

$$[v]([v/X]S) = [v]([N/X]S)$$

which is trivial since we know $v \notin E_X([v]S)$ (from (b)) which implies $X \notin GV(S)$.

More interestingly, if $x \neq v$ we know $x \notin LV(S)$ (using (a)) and $x \notin E_X(S)$ (since $X \in E_X(S)$ contradicts (b)), so the induction hypothesis can be applied to finish the proof. $\square$

We finish this section with some technical lemmas that clarify the behaviour of the two substitution operations.

Lemma 9. *The two substitution operations can be decomposed as follows.*

(1) $[N/x]M = [N/X][X/x]M$ *if $X \notin GV(M)$.*
(2) $[N/X]M = [N/x][x/X]M$ *if $x = H_X(M) \notin LV(M)$.*

Proof.

(1) By structural induction on M. In case $M = [y]M$ consider the subcases $x = y$ and $x \neq y$.
(2) $[N/x][x/X]M = [N/X][X/x][x/X]M$ by part (1) since $X \notin GV([x/X]M)$
                 $= [N/X]M$ by Lemma 8 since $x \notin LV(M)$. $\square$

Lemma 10.

(1) *If $X \notin GV(P, Q)$ then*

$$[X/x]P = [X/x]Q \implies P = Q.$$

(2) *If $H_X(P) \notin LV(P)$, $H_X(Q) \notin LV(Q)$ and $H_X(P) = H_X(Q)$ then*

$$[H_X(P)/X]P = [H_Y(Q)/Y]Q \implies P = [X/Y]Q.$$

Also, if $X \neq Y$ then $Y \notin GV(P)$ and $X \notin GV(Q)$.

Proof.

(1) By double induction on the structure of P and Q.
(2) Using Lemma 9. $\square$

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 (Section 4) for the following two reasons. First, 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. Second, 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 internal λ-terms are 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 $\mathtt{app}$ nor $\mathtt{lam}$ 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 '$(\mathtt{app}\, M\, N)$' as an abbreviation of '$(\mathtt{app}\ (M\ N))$'.

$$\frac{}{X : \mathbb{L}} \qquad \frac{M : \mathbb{L} \quad N : \mathbb{L}}{(\mathtt{app}\, M\, N) : \mathbb{L}} \qquad \frac{M : \mathbb{L} \quad x = \mathrm{H}_X(M)}{(\mathtt{lam}\ [x][x/X]M) : \mathbb{L}}\ (*) \tag{1}$$

The third rule $(*)$, as a constructor, takes two arguments, X and M, and constructs a new λ-expression whose bound variable, x, is determined. It is easy to see that $M : \mathbb{L}$ implies $\mathrm{LV}(M) = \{\}$ (which we often use implicitly below); the converse is false. A λ-term is called an *application* if it is defined by the second rule in Eq. (1), and an *abstract* if defined by the third rule. Each abstract $M = (\mathtt{lam}\ [x]P)$ defines a function $f_M : \mathbb{S} \to \mathbb{S}$ by putting $f_M(N) \overset{\triangle}{=} [N/x]P$ 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*.

3.1. Motivation

Symbolic expressions are not yet a good candidate for representing lambda terms for two reasons:

(1) Some symbolic expressions are ill formed by having local variables (natural numbers) that are not bound, e.g. the symbolic expression x.
(2) The set of well formed symbolic expressions in the above sense does not canonically represent the set of informal lambda terms; i.e. there are "alpha-convertible" symbolic expressions, such as $[x]x$ and $[y]y$.

The first of these problems is handled by inductively defining a predicate (i.e. a subset) on $\mathbb{S}$ picking out the symbolic expressions having no free local variables. In McKinna and Pollack (1993, 1999) this predicate is called *variable closed* (vclosed), defined by:

$$\frac{}{\text{vclosed}\ X} \qquad \frac{\text{vclosed}\ M \quad \text{vclosed}\ N}{\text{vclosed}\ (\mathtt{app}\, M\, N)} \qquad \frac{\text{vclosed}\ M}{\text{vclosed}\ (\mathtt{lam}\ [x][x/X]M)}\ (+)$$

'vclosed M' is equivalent to $\mathrm{LV}(M) = \{\}$, and also has a useful induction principle. It is clear that substitution, $[\_/X]\_$, is capture avoiding on vclosed (there are no free local variables to get captured) and that vclosed is closed under substitution. However vclosed still has the second problem mentioned above. A consequence of this weakness, for example, is that the Church–Rosser theorem for β-reduction does not hold concretely of vclosed symbolic expressions.[3]

The second problem, the failure to canonically represent informal lambda terms, can be solved using a *locally nameless* variation on symbolic expressions. In this variation, global variables are represented by a class of atoms, $\mathbb{X}$, as in our symbolic expressions, but local variables are natural numbers serving as de Bruijn indices rather than natural numbers serving as names. This approach goes back to the original paper on indices by de Bruijn (1972), and its formalization in a computer proof tool is detailed in Aydemir et al. (2008). This locally nameless representation works very well, but the

---

[3] Nonetheless McKinna and Pollack (1993, 1999) developed considerable metatheory of Pure Type Systems without the need to reason about α-conversion. This is possible because Tait–Martin-Löf parallel reduction defined over vclosed symbolic expressions does have the Church–Rosser property "on the nose", so β-conversion defined using parallel reduction is well behaved.

syntactic representation of terms does not have a name at binding points, which differs from informal practice. Further, some of the annoying index manipulation of de Bruijn representation remains.

A main contribution of the present paper, developed by the first author, is an alternative solution to this problem of canonical representation. Rule $(+)$ above, viewed as a constructor of lambda terms, takes X and M and constructs $(\text{lam } [x][x/X]M)$ for *any* x. To make the representation canonical it suffices to choose x canonically in this construction, and that is the purpose of the height function $H_X(M)$; compare rules $(*)$ and $(+)$. Of course, this canonical choice must be well behaved. Since $\mathbb{L} \subseteq$ vclosed, it is clear that substitution is capture avoiding on $\mathbb{L}$. But it is not obvious that $\mathbb{L}$ is closed under substitution (Theorem 1). So we proceed to develop a theory of $\mathbb{L}$.

3.2. Properties of the internal notation

We explain the notion of equivariance for the set $\mathbb{S} = \mathbb{S}[\mathbb{X} \cup \{\text{app}, \text{lam}\}]$. Equivariance 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 Gabbay and Pitts (2002).)

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 $\mathbb{S}' = \mathbb{S}[\mathbb{X} \cup \{\text{app}, \text{lam}\} \cup \mathbb{C}]$ 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:

$$\frac{}{[\pi]X : \mathbb{L}} \qquad \frac{[\pi]M : \mathbb{L} \quad [\pi]N : \mathbb{L}}{(\text{app } [\pi]M\,[\pi]N) : \mathbb{L}} \qquad \frac{[\pi]M : \mathbb{L} \quad H_{[\pi]X}([\pi]M) = x}{(\text{lam } [x][x/[\pi]X][\pi]M) : \mathbb{L}} \; (\dagger)$$

They are instances of the corresponding rules (1), including the side condition in $(\dagger)$ since H is equivariant (Lemma 5).

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 '$(\text{app } X\,X) : \mathbb{L}$' to a false statement '$(\text{lam } X\,X) : \mathbb{L}$' for any $X \in \mathbb{X}$.

Lemma 11. *If $P \in \mathbb{L}$ and $Y \notin GV(P)$ then $[Y/X]P \in \mathbb{L}$.*

Proof. From equivariance of $\mathbb{L}$ and the definition of $[Y/X]P$. $\square$

Lemma 12. *The following are equivalent:*

(1) $(\text{lam } [v]S) \in \mathbb{L}$
(2) $S = [v/X]P$ and $v = H_X(P)$ for some X and $P \in \mathbb{L}$
(3) $v = H_Z([Z/v]S)$ and $[Z/v]S \in \mathbb{L}$ for every $Z \notin GV(S)$.

Case (2) is the inversion of '$(\text{lam } [v]S) \in \mathbb{L}$' by rule $(*)$ in the definition of $\mathbb{L}$ (Eq. (1))

$$\frac{M : \mathbb{L} \quad x = H_X(M)}{(\text{lam } [x][x/X]M) : \mathbb{L}} \; (*).$$

In this "forward" rule, X and M vary together. To see where case (3) comes from notice rule $(*)$ could equivalently be stated as a "backwards" rule, analogous to that used in McKinna and Pollack (1993, 1999), Aydemir et al. (2008)

$$\frac{X \notin GV(M) \quad [X/x]M : \mathbb{L} \quad x = H_X([X/x]M)}{(\text{lam } [x]M) : \mathbb{L}} \; (**).$$

In this form X varies independently of M, and it is clear that any sufficiently fresh X will do in the premises, so the following rule is also equivalent

$$\frac{\forall X.\ (X \notin \mathrm{GV}(M) \implies [X/x]M : \mathbb{L} \wedge x = \mathrm{H}_X([X/x]M))}{(\mathtt{lam}\ [x]M) : \mathbb{L}}\ (***).$$

Case (3) of Lemma 12 is the inversion of '$(\mathtt{lam}\ [v]S) \in \mathbb{L}$' by rule $(***)$. Formally, the proof below stands by itself, independent of this explanation.

Proof (*Proof of Lemma* 12). We prove the interesting case, (2) $\implies$ (3). Let X and P be as in (2) and choose $Z \notin \mathrm{GV}(S) = \mathrm{GV}([v/X]P)$. We need to show

$$v = \mathrm{H}_Z([Z/v][v/X]P) \quad \text{and} \quad [Z/v][v/X]P \in \mathbb{L}.$$

By Lemma 8 it suffices to show

$$v = \mathrm{H}_Z([Z/X]P) \quad \text{and} \quad [Z/X]P \in \mathbb{L}.$$

In case $Z = X$ this holds by assumption, so assume $Z \neq X$. Since $Z \notin \mathrm{GV}([v/X]P)$, we have $Z \notin \mathrm{GV}(P)$. Thus the first conjunct follows from Lemma 5 and the second from Lemma 11. $\square$

Remark 1 (*Inverting "forward" vs "backwards" Rules*). Inverting '$(\mathtt{lam}\ [v]S) \in \mathbb{L}$' by rule $(*)$ we obtain

$$\exists X\, P.\ (\mathtt{lam}\ [v]S) = (\mathtt{lam}\ [v][v/X]P) \wedge P : \mathbb{L}$$

(for discussion about mechanised inversion, see McBride (1998), Cornes and Terrasse (1995)). Since $\mathtt{lam}$ is injective, we have

$$\exists X\, P.\ S = [v/X]P \wedge P : \mathbb{L}. \tag{2}$$

However, even for given v and X, $[v/X](-)$ is not injective, so P cannot be determined from Eq. (2). For example

$$[1/X](\mathtt{app}\, X\, 1) = (\mathtt{app}\, 1\, 1) = [1/X](\mathtt{app}\, X\, X).$$

(Notice that the inverse image of $[v/X](-)$ does not respect $\mathbb{L}$: $(\mathtt{app}\, X\, 1) \notin \mathbb{L}$ while $(\mathtt{app}\, X\, X) \in \mathbb{L}$.) On the other hand, inverting '$(\mathtt{lam}\ [v]S) \in \mathbb{L}$' by rule $(**)$ we obtain

$$\exists X.\ [X/v]S \in \mathbb{L}$$

directly, which is more useful in practice. The forward and backward rules are equivalent, but for inversion, the backward rule is more convenient. The same situation occurs for many relations defined on $\mathbb{S}$.

Remark 2 (*Decidability of* $\mathbb{L}$). Given any $M \in \mathbb{S}$, we can decide whether or not $M \in \mathbb{L}$. For example, if M is of the form $(\mathtt{lam}\ [v]S)$, then (using Lemma 12) it suffices to choose $Z \notin \mathrm{GV}(S)$ and check case (3), which we can do recursively since $|[Z/v]S| = |S| < |M|$. Deciding the other cases is similar.

We have the following key theorem which guarantees that λ-terms are closed under substitution.

Theorem 1 (*Substitution*). *If* $P, Q : \mathbb{L}$ *then* $[Q/Y]P : \mathbb{L}$.

The proof of Theorem 1 encounters a problem that is well known and discussed in the literature, e.g. McKinna and Pollack (1993, 1999), Urban et al. (2007), Pitts (2003). We want to do induction on the derivation of $P : \mathbb{L}$. In case P is an abstraction, the induction hypothesis mentions some particular X. To make the argument go through we need to swap X for a new atom Z chosen to be sufficiently fresh (in this case, not appearing in Y or Q). Roughly, this is possible because $\mathbb{L}$ is equivariant. However, it is more convenient to once-and-for-all derive a strengthened induction principle for $\mathbb{L}$ than to reason about atom permutation in many examples. Just as the equivalence between rules $(*)$ and $(***)$ motivated Lemma 12 above, it motivates the following derived induction principle.

Lemma 13 (*Strengthened Induction for* $\mathbb{L}$). *The following rule is admissible*

> (1) $\forall X.\ \Phi(X)$
> (2) $\forall M, N.\ M : \mathbb{L} \wedge \Phi(M) \wedge N : \mathbb{L} \wedge \Phi(N) \implies \Phi((\text{app } M\,N))$
> (3) $\forall x, M.\quad (\forall X.\ X \notin \text{GV}(M) \implies$
> $\qquad\qquad\qquad x = \text{H}_X([X/x]M) \wedge [X/x]M : \mathbb{L} \wedge \Phi([X/x]M)) \implies$
> $\qquad \Phi((\text{lam } [x]M))$
> ───
> $\qquad\qquad\qquad \forall N.\ N : \mathbb{L} \implies \Phi(N)$

This lemma is 'strengthened' in the sense that the induction hypothesis for the lam case (premise (3)) applies to any $X \notin \text{GV}(M)$.

Proof. Assuming the three premises, show $N : \mathbb{L} \implies \Phi(N)$ by induction on $|N|$ followed by case analysis on $N : \mathbb{L}$. The interesting case is when $N = (\text{lam } [y][y/X]M)$ with $y = \text{H}_X(M)$. The induction hypothesis is

$$\forall Q.\ (|Q| < |N| \wedge Q : \mathbb{L}) \implies \Phi(Q).$$

We need to show $\Phi((\text{lam } [y][y/X]M))$, so want to apply premise (3). Choosing

$$Z \notin \text{GV}([y/X]M)$$

we need to show

$$y = \text{H}_Z([Z/y][y/X]M), \qquad [Z/y][y/X]M : \mathbb{L}, \qquad \Phi([Z/y][y/X]M).$$

Since we know $N : \mathbb{L}$, we can instantiate case (3) of Lemma 12 with Z, showing the first two of these goals. All that remains is to show the third goal, which follows from the induction hypothesis. □

Proof of Theorem 1. Induct on $P : \mathbb{L}$ using the derived induction principle of Lemma 13. The interesting case is when $P = (\text{lam } [x]M)$. The induction hypothesis is

$$\forall X.\ X \notin \text{GV}(M) \implies x = \text{H}_X([X/x]M) \wedge [X/x]M : \mathbb{L} \wedge [Q/Y][X/x]M : \mathbb{L}.$$

Choose Z not occurring in Y, M or Q; from induction hypothesis have

$$x = \text{H}_Z([Z/x]M) \qquad \text{and} \qquad [Q/Y][Z/x]M : \mathbb{L}. \qquad\qquad (3)$$

The goal is to show $[Q/Y](\text{lam } [x]M) : \mathbb{L}$. Using Lemma 12 (taking P in case (2) to be $[Z/x][Q/Y]M$) it suffices to show

$$[Z/x][Q/Y]M : \mathbb{L}, \qquad [Q/Y]M = [x/Z][Z/x][Q/Y]M, \qquad x = \text{H}_Z([Z/x][Q/Y]M).$$

Using Lemma 4 and the hypothesis $Q : \mathbb{L}$ we have $[Z/x][Q/Y]M = [Q/Y][X/x]M$, so the first and third of these goals follow from Lemma 6 and Eq. (3). The second follows from Lemma 3. □

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

Proof. Inverting $(\text{lam } [x]P) : \mathbb{L}$ we know $x = \text{H}_X(P)$, $M = [x/X]P$ and $P : \mathbb{L}$ for some X, P (Lemma 12). Using Lemma 9 we have

$$[N/x]M = [N/x][x/X]P = [N/X]P.$$

The RHS is a λ-term by Theorem 1. □

3.3. Some examples: Reduction and typing

We present *simple type assignment* to lambda terms. Let $\mathbb{A}$ be a set of *atomic types*, ranged over by A, B, C. Let S, T, U range over *simple types*, $\mathbb{T}$, freely generated from $\mathbb{A}$ by the rules:

$$\frac{}{A : \mathbb{T}} \qquad \frac{S : \mathbb{T} \quad T : \mathbb{T}}{S \to T : \mathbb{T}}\ .$$

A *type basis* (or *type context*), Γ, is a set of pairs (X, T) such that no two different pairs have the same first component. Type assignment is the relation defined by the rules[4]:

$$\frac{(X, T) \in \Gamma}{\Gamma \vdash X : T} \qquad \frac{\Gamma \vdash M : S \to T \quad \Gamma \vdash M : S}{\Gamma \vdash (\text{app } M \, N) : T}$$

$$\frac{\Gamma \cup (X, S) \vdash M : T \quad x = \text{H}_X(M)}{\Gamma \vdash (\text{lam } [x][x/X]M) : S \to T} \ (*).$$

It is easy to show that type assignment is equivariant, and that $\Gamma \vdash M : T$ implies $M : \mathbb{L}$. The side condition on rule $(*)$ is necessary for the latter property.

We can define β-reduction on the set $\mathbb{L}$ of λ-terms along standard lines, as in Barendregt (1984). First, the β rule

$$\frac{(\text{lam } M) : \mathbb{L} \quad N : \mathbb{L}}{(\text{app } (\text{lam } M) \, N) \to (\text{lam } M)(N)} \ (\beta).$$

Since $(\text{lam } M) : \mathbb{L}$ implies that M is of the form $[x]P$, we have $(\text{lam } M)(N) = [N/x]P$, which is a lambda term by Theorem 2. We also need congruence rules for the constructors app and lam of lambda terms

$$\frac{M_1 \to M_2 \quad N : \mathbb{L}}{(\text{app } M_1 \, N) \to (\text{app } M_2 \, N)} \qquad \frac{M : \mathbb{L} \quad N_1 \to N_2}{(\text{app } M \, N_1) \to (\text{app } M \, N_2)}$$

$$\frac{M \to N \quad x = \text{H}_X(M) \quad y = \text{H}_X(N)}{(\text{lam } [x][x/X]M) \to (\text{lam } [y][y/X]N)} \ (\xi).$$

The subtlety that rule (ξ) (reduction under a binder) may change the bound name is essential for this concrete representation with canonical names to work.[5]

It is easy to see that this relation is equivariant. The side conditions on the rules guarantee that $M \to N$ implies $M : \mathbb{L}$ and $N : \mathbb{L}$. Another natural property is that if $M \to N$ and $X \notin \text{GV}(M)$ then $X \notin \text{GV}(N)$.

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

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

In 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 rename the bound variable y to w in the first reduction step to avoid capturing the free variable y.

In order to translate this smoothly into our internal notation we use two functions $\cdot : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$ and $lam : \mathbb{X} \times \mathbb{L} \to \mathbb{L}$ defined by:

$$(M \cdot N) \stackrel{\triangle}{=} (\text{app } M \, N),$$
$$lam_X(M) \stackrel{\triangle}{=} (\text{lam } [x][x/X]M) \quad \text{where } x = \text{H}_X(M).$$

---

[4] We are being slightly informal here about validity of type contexts.

[5] Along similar lines see the discussion in McKinna and Pollack (1993, 1999) of Tait/Martin-Löf parallel reduction, and of a dependent typing rule for abstractions where an abstraction and its dependent type may use different bound names.

The above informal term corresponds to the following λ-term.

$$(lam_Z(lam_X((lam_Y((Z{\cdot}Y)){\cdot}(X{\cdot}Z)))){\cdot}Y)$$
$$= (lam_Z(lam_X((lam_Y((\texttt{app } Z\ Y)){\cdot}(\texttt{app } X\ Z)))){\cdot}Y)$$
$$= (lam_Z(lam_X(((\texttt{lam [1]}(\texttt{app } Z\ 1)){\cdot}(\texttt{app } X\ Z)))){\cdot}Y)$$
$$= (lam_Z((\texttt{lam [1]}(\texttt{app } (\texttt{lam [1]}(\texttt{app } Z\ 1))\ (\texttt{app } 1\ Z)))){\cdot}Y)$$
$$= ((\texttt{lam [2]}(\texttt{lam [1]}(\texttt{app } (\texttt{lam [1]}(\texttt{app } 2\ 1))\ (\texttt{app } 1\ 2)))){\cdot}Y)$$
$$= (\texttt{app } (\texttt{lam [2]}(\texttt{lam [1]}(\texttt{app } (\texttt{lam [1]}(\texttt{app } 2\ 1))\ (\texttt{app } 1\ 2))))\ Y).$$

We can compute this term as follows.

$$(\texttt{app } (\texttt{lam [2]}(\texttt{lam [1]}(\texttt{app } (\texttt{lam [1]}(\texttt{app } 2\ 1))\ (\texttt{app } 1\ 2))))\ Y)$$
$$\rightarrow (\texttt{lam [1]}(\texttt{app } (\texttt{lam [1]}(\texttt{app } Y\ 1))\ (\texttt{app } 1\ Y)))$$
$$\rightarrow (\texttt{lam [1]}(\texttt{app } Y\ (\texttt{app } 1\ Y))).$$

We will use the functions $\cdot$ 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 traditional syntax of λ-terms with named variables. In our formulation of the external syntax we use global variables but not 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 '$(\texttt{lam } X\ M)$' for '$(\texttt{lam } (X\ M))$' and will continue to write '$(\texttt{app } M\ N)$' for '$(\texttt{app } (M\ N))$'.

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

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 $[\![-]\!] : \Lambda \rightarrow \mathbb{L}$ which, for each $M \in \Lambda$, defines its *denotation* $[\![M]\!] \in \mathbb{L}$ as follows.

$$[\![X]\!] \stackrel{\triangle}{=} X,$$
$$[\![(\texttt{app } M\ N)]\!] \stackrel{\triangle}{=} ([\![M]\!]{\cdot}[\![N]\!]),$$
$$[\![(\texttt{lam } X\ M)]\!] \stackrel{\triangle}{=} lam_X([\![M]\!]).$$

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 $FV(M)$, *the set of free variables in M*, simply by putting: $FV(M) \stackrel{\triangle}{=} GV([\![M]\!])$. After *defining* $FV(M)$ this way, we can *prove* the following equations which characterize the set $FV(M)$ in terms of the language of the external syntax.

$$FV(X) = \{X\},$$
$$FV((\texttt{app } M\ N)) = FV(M) \cup FV(N),$$
$$FV((\texttt{lam } X\ M)) = FV(M) - \{X\}.$$

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

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

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

since

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

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

We now verify the adequacy (see Harper et al. (1993)) of our definition of α-equivalence against the definition of α-equivalence due to Gabbay and Pitts (2002), Pitts (2003). Their definition, in our notation, is as follows.

$$
\frac{M : \Lambda}{M =_\alpha M} \qquad \frac{M =_\alpha P \quad N =_\alpha Q}{(\texttt{app}\,M\,N) =_\alpha (\texttt{app}\,P\,Q)}
$$

$$
\frac{[X /\!\!/ Z]M =_\alpha [Y /\!\!/ Z]N \quad Z \notin GV(M) \cup GV(N)}{(\texttt{lam}\,X\,M) =_\alpha (\texttt{lam}\,Y\,N)}
$$

The adequacy is established by interpreting these rules in our internal syntax and showing soundness and completeness (Theorem 3) which is preceded by the following lemma.

Lemma 14. *If $M =_\alpha N$, then $H_X(M) = H_X([\![M]\!]) = H_X([\![N]\!]) = H_X(N)$ for all $X \in \mathbb{X}$.*

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

Theorem 3 (*Soundness and Completeness*). *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 $Z \notin GV(M) \cup GV(N)$. By induction hypothesis, we have $[\![[X /\!\!/ Z]M]\!] = [\![[Y /\!\!/ Z]N]\!]$. Our goal is to show that $[\![(\texttt{lam}\,X\,M)]\!] = [\![(\texttt{lam}\,Y\,N)]\!]$. We have

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

and

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

where $x = H_X([\![M]\!])$ and $y = H_Y([\![N]\!])$. Now, by the freshness of Z and by Lemma 14, we have $x = H_X([\![M]\!]) = H_Z([X /\!\!/ Z][\![M]\!]) = H_Z([Y /\!\!/ Z][\![N]\!]) = H_Y([\![N]\!]) = y$. 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:

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

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

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

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

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.

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

$$\frac{}{[P/X](\text{lam } X \, M) \Downarrow (\text{lam } X \, M)} \qquad \frac{[P/X]M \Downarrow N \quad X \neq Y \quad Y \notin \text{FV}(P)}{[P/X](\text{lam } Y \, M) \Downarrow (\text{lam } Y \, N)}$$

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

This substitution relation 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 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 algebraically as endomorphisms on $\mathbb{S}$ and permutations as automorphisms on $\mathbb{S}$.

We conclude the paper by comparing our formulation with that by Gabbay and Pitts (2002), that by Aydemir et al. (2008) and finally those by Quine (1951), Bourbaki (1968), Sato and Hagiya (1981) and Sato (1983).

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 an induction principle for λ-terms can be introduced and proven to be correct. The set of 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 λ-terms and capture avoiding substitution can be manipulated rigorously using arguments similar to standard informal arguments, which are otherwise very difficult to make rigorous. They use the equivariance property under finite permutations of atoms extensively. Pitts later introduced the notion of *nominal sets* (Pitts, 2003, 2006) and showed that essentially the same results can be obtained within the framework of standard mathematics. These nominal ideas have been implemented in higher order logic as the nominal package in Isabelle/HOL (Urban, 2008). Nominal Isabelle makes nominal reasoning available to non-experts, with significant automation. However, even with automation one sometimes has to reason explicitly about α-equivalence in nominal Isabelle, while that is never necessary with our canonical internal representation $\mathbb{L}$.

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. (2008) uses two sorts of variables, one for global variables and the other for local variables just like our internal syntax. However, their local variables, also natural numbers, serve as de Bruijn indices, so their binders are nameless while ours carry explicit names (natural numbers are names). In spite of this difference, substitution of a term for a global variable goes as smoothly for them as in our case, since both formulations use two sorts of variables. However, their substitution operations are not characterized as homomorphisms due to the lack of algebraic structure on their terms.

Another difference concerns the formation of abstraction. To explain the difference, note that our introduction rule of abstracts, (∗), could equivalently be formulated, in a backward way so to speak, as mentioned in the discussion following Lemma 12:

$$\frac{X \notin \mathrm{GV}(M) \quad x = \mathrm{H}_X([X/x]M) \quad [X/x]M : \mathbb{L}}{(\texttt{lam } [x]M) : \mathbb{L}} \quad (**).$$

Although this is a technically correct rule, it 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 Sato (2002), we believe that every *mathematical object*, including of course every λ-term, must be constructed by applying a *constructor* function to *already created objects*. This rule does not follow this *ontological condition*, so we chose instead the abstraction introduction rule in Section 3.

If formulated in the nameless style of Aydemir et al. (2008), the forward rule for constructing abstracts in $\mathbb{L}$ would become (cf. the TYPING-ABS rule in Aydemir et al. (2008, Figure 1)):

$$\frac{X \notin \mathrm{GV}(M) \quad M^X : \mathbb{L}}{(\texttt{lam } M) : \mathbb{L}} \quad (\ddagger)$$

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 rule ($\ddagger$) is the *opening up* of M by X which corresponds to our instantiation of $(\texttt{lam } [x]M)$ 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)))` can be instantiated just by

substituting *X* for two occurrences of 2. We may thus say that the representation of λ-terms by the method of Aydemir et al. (2008) is more complex than our method and that their rule for introducing abstraction is ontologically unnatural as it requires to mentally construct the term beforehand.

Finally, we remark that as a data structure our internal language is isomorphic to representations of binding by Quine (1951, page 70), Bourbaki (1968, Chapter 1), Sato and Hagiya (1981) and Sato (1983, 1991). These representations are nameless since abstraction is realized by links between the bound nodes and the binding node. This is usually implemented on a computer using pointers for links. However, except for Sato and Hagiya (1981) and Sato (1983, 1991), these data structures do not admit well-founded induction principle, since they contain *cycles*. Unlike these, our representation admits reasoning by induction on the birthday of each expression, and has a nice algebraic structure.

5.1. Ongoing work

At the time of preparing the final version of this paper, we are working on a more abstract presentation of the internal language. As mentioned in Section 3, instead of the concrete height function, H, we consider the properties that a height function must have such that $\mathbb{L}$ is isomorphic to some well known formalization of λ-terms, such as the nominal Isabelle representation (Urban, 2008), or the locally nameless representation (Aydemir et al., 2008). You can see where we stand at time of writing by looking at Pollack and Sato (2009).

Beyond theory, there is application. We claim that our representation is more elegant than the closest comparable work, Aydemir et al. (2008). But is it more practical to use? We must do more interesting examples than those mentioned in Section 3.3 to find out.

Acknowledgements

References

Aydemir, B., Charguéraud, A., Pierce, B., Pollack, R., Weirich, S., 2008. Engineering formal metatheory. In: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles on Programming Languages. ACM Press, pp. 3–15.

Barendregt, H., 1984. The Lambda Calculus: Its Syntax and Semantics, revised Edition. In: Studies in Logic and the Foundations of Mathematics, vol. 103. North-Holland.

Bourbaki, N., 1968. Elements of Mathematics I, Theory of Sets. Addison-Wesley. English translation of *Eléments de Mathématique: Théories des Ensembles, Livre I*, Hermann, 1957.

Church, A., 1940. A simple theory of types. Journal of Symbolic Logic 5, 56–68.

Church, A., 1941. The Calculi of Lambda-Conversion. In: Annals of Mathematical Studies, vol. 6. Princeton University Press.

Cornes, C., Terrasse, D., 1995. Automating inversion of inductive predicates in coq. In: Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5–8, 1995, Selected papers.

Curry, H.B., Feys, R., 1958. Combinatory Logic, vol. 1. North Holland.

de Bruijn, N., 1972. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem. Koninklijke Nederlandse Akademie van Wetenschappen. Indagationes Mathematicae 34 (5).

Fiore, M., Plotkin, G., Turi, D., 1999. Abstract syntax and variable binding (extended abstract). In: Proc. 14th LICS Conf. IEEE. Computer Society Press, pp. 193–202.

Frege, G., 1879. Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Halle, translated in van Heijenoort (1967, pp. 1–82).

Gabbay, M., Pitts, A., 1999. A new approach to abstract syntax involving binders. In: Longo, G. (Ed.), Proceedings of the 14th Annual Symposium on Logic in Computer Science, LICS'99, pp. 214–224.

Gabbay, M.J., Pitts, A.M., 2002. A new approach to abstract syntax with variable binding. Formal Aspects of Computing 13, 341–363.

Gentzen, G., 1934. Untersuchungen über das logische schliessen. Mathematische Zeitschrift 39. English translation in Szabo (1969).

Gödel, K., 1931. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatsh. Math. Phys. 38, 173–198. English translation in van Heijenoort (1967).

Harper, R., Honsell, F., Plotkin, G., 1993. A framework for defining logics. J. ACM 40 (1), 143–184. Preliminary version in LICS'87.

McBride, C., 1998. Inverting inductively defined relations in LEGO. In: Giménez, E., Paulin-Mohring, C. (Eds.), TYPES'96: Workshop on Types for Proofs and Programs, Aussois; Selected Papers. In: LNCS, vol. 1512. Springer-Verlag.

McCarthy, J., 1960. Recursive functions of symbolic expressions and their computation by machine, part I. Communications of the ACM 3 (4), 184–195.

McCarthy, J., 1963. A basis for a mathematical theory of computation. In: Braffort, Hirschberg (Eds.), Computer Programming and Formal Systems. North-Holland, pp. 33–70.

McKinna, J., Pollack, R., 1993. Pure Type Systems formalized. In: Bezem, M., Groote, J.F. (Eds.), Proceedings of the International Conference on Typed Lambda Calculi and Applications. TLCA'93, Utrecht. In: LNCS, vol. 664. Springer-Verlag, pp. 289–305.

McKinna, J., Pollack, R., 1999. Some lambda calculus and type theory formalized. Journal of Automated Reasoning 23 (3–4), 373–409.

Nordström, B., Petersson, K., Smith, J., 1990. Programming in Martin-Löf's Type Theory. An Introduction. Oxford Univ. Press. out of print, but available from www.cs.chalmers.se/Cs/Research/Logic/book/.

Pitts, A., 2003. Nominal logic, a first order theory of names and binding. Information and Computation 186, 165–193.

Pitts, A.M., 2006. Alpha-structural recursion and induction. Journal of the ACM 53, 459–506.

Pollack, R., Sato, M., 2009. A canonical locally named representation of binding. In: 4th Informal ACM SIGPLAN Workshop on Mechanizing Metatheory, WMM'09. Slides available on
http://www.seas.upenn.edu/~sweirich/wmm/wmm09-programme.html.

Quine, W., 1951. Mathematical Logic, Revised edition. Harvard University Press.

Russell, B., Whitehead, A., 1910. Principia Mathematica, vol. 1. Cambridge Univ. Press.

Sato, M., 1983. Theory of symbolic expressions, I. Theoretical Computer Science 22, 19–55.

Sato, M., 1985. Theory of symbolic expressions, II. Publ. RIMS, Kyoto Univ., pp. 455–540.

Sato, M., 1991. An abstraction mechanism for symbolic expressions. In: Lifschitz, V. (Ed.), Artificial Intelligence and Mathematical Theory of Computation (Papers in Honor of John McCarthy). Academic Press, pp. 381–391.

Sato, M., 2002. Theory of judgments and derivations. In: Arikawa, Shinohara (Eds.), Progress in Discovery Science. In: LNAI, vol. 2281. Springer-Verlag, pp. 78–122.

Sato, M., 2008a. External and internal syntax of the λ-calculus. In: Buchberger, Ida, Kutsia (Eds.), Proc. of the Austrian–Japanese Workshop on Symbolic Computation in Software Science, SCSS 2008. No. 08–08 in RISC-Linz Report Series. pp. 176–195.

Sato, M., 2008b. A framework for checking proofs naturally. Journal of Intelligent Information Systems 31 (2), 111–125.

Sato, M., Hagiya, M., 1981. Hyperlisp. In: Proceedings of the International Symposium on Algorithmic Language. North-Holland, pp. 251–269.

Stoughton, A., 1988. Substitution revisited. Theoretical Computer Science 17, 317–325.

Szabo, M.E. (Ed.), 1969. The Collected Papers of Gerhard Gentzen. North Holland.

Urban, C., 2008. Nominal techniques in Isabelle/HOL. Journal of Automated Reasoning 40 (4).

Urban, C., Berghofer, S., Norrish, M., 2007. Barendregt's variable convention in rule inductions. In: Automated Deduction – CADE-21. In: LNCS, vol. 4603. Springer-Verlag, pp. 35–50.

van Heijenoort, J., 1967. From Frege to Gödel: A Source Book in Mathematical Logic. Harvard University Press, Cambridge, Mass, pp. 1879–1931.

Vestergaard, R., 2003. The primitive proof theory of the λ-calculus. Ph.D. thesis, Heriot-Watt University.