A Simply Typed Context Calculus with First-class Environments

Masahiko Sato[*] Takafumi Sakurai[†]
Yukiyoshi Kameyama[‡]

March 10, 2002

Abstract

We introduce a simply typed λ-calculus $\lambda\kappa\varepsilon$ which has both contexts and environments as first-class values. In $\lambda\kappa\varepsilon$, holes in contexts are represented by ordinary variables of appropriate types and hole filling is represented by the functional application together with a new abstraction mechanism which takes care of packing and unpacking of the term which is used to fill in the holes of the context. $\lambda\kappa\varepsilon$ is a conservative extension of the simply typed $\lambda\beta$-calculus, enjoys subject reduction property, is confluent and strongly normalizing.

The traditional method of defining substitution does not work for our calculus. So, we also introduce a new method of defining substitution. Although we introduce the new definition of substitution out of necessity, the new definition turns out to be conceptually simpler than the traditional definition of substitution.

1 Introduction

Informally speaking, a context (in λ-calculus) is a λ-term with some holes in it. For example, writing $[\ ]$ for a hole, $\lambda y.\ [\ ]$ is a context, and by filling

[*]Graduate School of Informatics, Kyoto University, Yoshida-Honmachi, Sakyo-ku, Kyoto, 606-8501, Japan, `masahiko@kuis.kyoto-u.ac.jp`

[†]Department of Mathematics and Informatics, Chiba University, Yayoi 1-33, Inage, Chiba, 263-8522, Japan, `sakurai@math.s.chiba-u.ac.jp`

[‡]Institute of Information Sciences and Electronics, University of Tsukuba, Tennodai 1-1-1, Tsukuba, 305-8573, Japan, `kam@is.tsukuba.ac.jp`

the hole in it with $x + y$, we get $\lambda y.\ x + y$. By this operation, the variable y in $x + y$ gets *captured* and becomes bound in $\lambda y.\ x + y$, and the variable x remains to be free. So, unlike substitution, hole filling may introduce new and intended bound variables. The notion of contexts is important in theoretical investigations such as contextual equivalence and evaluation contexts, but it is getting important in modeling practical computations as well. For instance, to model distributed computing, we often need to represent open terms (terms with free variables) and dynamic binding of variables, and that kind of representation is also necessary to treat modules as first-class citizens. Such a representation can be given by formalizing contexts and the hole-filling operations, but we refer to [10] for related discussions. Here we point out that the key idea is to link distributed objects through the names of variables.

Recently there have been several attempts to formalize the notion of context and thereby make computing with contexts possible. For example, Talcott [19], Lee-Friedman [11], Dami [5], Hashimoto-Ohori [10], Sands [16], Mason [12] and Bognar-de Vrijer [4] made notable contributions. However, as far as we know, there is as yet no proposal of a language which has contexts as first-class values and which is at the same time *pure* in the following sense. We understand that a functional language is *pure*[1] if (i) it is a conservative extension of the untyped or simply typed $\lambda\beta$-calculus, (ii) confluent and (iii) strongly normalizing (SN) if the language is typed and has preservation of strong normalization (PSN) property if the language is untyped. The conservative extension property guarantees that the language is logically well-behaved and the confluence property and SN or PSN would guarantee that the language is computationally well-behaved.

In this paper, we introduce the calculus $\lambda\kappa\varepsilon$ (κ is for context and ε is for environment) which is pure in the above sense and which has contexts and environments as its first class values, so that we can bind contexts and environments to variables and return them as the values of computations. $\lambda\kappa\varepsilon$ is a simply typed calculus, and in $\lambda\kappa\varepsilon$, holes are represented by ordinary variables of appropriate types (which we will call hole types) and hole filling is represented by the functional application together with a new abstraction mechanism which takes care of packing and unpacking of the term which is used to fill in the holes of the context.

We now illustrate some of the difficulties we face in formalizing the notion

[1]We have introduced this notion of purity in [18].

of context, and explain our solution informally by relating it to previous works. First, let us consider the context:

$$a[\,] \equiv \lambda x.\ (\lambda y.\ x + [\,])3.$$

If we fill the hole in $a[\,]$ with the term $x + y$, we get

$$a[x + y] \equiv \lambda x.\ (\lambda y.\ x + (x + y))3.$$

By β-reducing it, we can convert $a[x+y]$ to $\lambda x.\ x+(x+3)$. Since we wish to compute with contexts, we would also like to reduce the β-redex $(\lambda y.\ x+[\,])3$ in $a[\,]$. If we reduce it naïvely, we get $x+[\,]$, so that $a[\,]$ reduces to $\lambda x.\ x+[\,]$. Now, if we fill the hole in $\lambda x.\ x + [\,]$ with $x + y$, we get $\lambda x.\ x + (x + y)$. This shows that hole filling and β-reduction do not commute if we define hole filling and β-reduction as above. In this example, we can note that the hole in the original context $a[\,]$ is within the scope of λx and λy, while the hole in the β-reduced context is only within the scope of λx. This means that a part of the information as to which variables should be captured at the hole is lost if one reduces a β-redex which has a hole in it. Hashimoto-Ohori [10] did not solve this problem. Instead they put restriction on the β-reduction rule in their system and prohibited such β-reductions like the above example.

To solve this problem, we introduce the type A^E which represents the set of objects obtained by abstracting objects of type A with respect to a set $E = \{x_1, \ldots, x_n\}$ of variables. Canonical objects of type A^E are abstracts of the form $\kappa E.\ a$ where a is of type A and the κE binder declares that the variables in E should be understood as local in a. Moreover, E is also a type and its canonical elements are *environments* of the form $\{a_1/x_1, \ldots, a_n/x_n\}$. Then, an object a of type A^E can be instantiated to an object b of type A by applying the abstract a to an environment $e = \{a_1/x_1, \ldots, a_n/x_n\}$. We write $a{\bullet}e$ for the application of the abstract a to the environment e. For example, $(\kappa\{x, y\}.\ x + y){\bullet}\{1/x, 2/y\}$ can be reduced to 3.

In this setting, we can represent the above context $a[\,]$ as

$$C \equiv \lambda x.\ (\lambda y.\ x + X{\bullet}\{x/x, y/y\})3$$

where X represents the hole and its type is of the form $A^{\{x,y\}}$. Now, suppose that we wish to fill the hole X with the term $x+y$. Then, we can achieve this hole filling by substituting $\kappa\{x, y\}.\ x + y$ for X in C. By this substitution, we have:

$$D \equiv \lambda x.\ (\lambda y.\ x + (\kappa\{x, y\}.\ x + y){\bullet}\{x/x, y/y\})3.$$

D can be reduced to $\lambda x.\ (\lambda y.\ x + (x + y))3$, which can be further reduced to $\lambda x.\ x + (x + 3)$ as expected. Let us now see what happens if we reduce the β-redex in C first and then fill the hole with $x + y$. By β-reducing C, we get $\lambda x.\ x + X{\cdot}\{x/x, 3/y\}$. Substituting $\kappa\{x, y\}.\ x + y$ for X in this term, we have

$$\lambda x.\ x + (\kappa\{x, y\}.\ x + y){\cdot}\{x/x, 3/y\},$$

which we can further reduce to $\lambda x.\ x + (x + 3)$. We can thus see that hole filling and β-reduction commute in this case.

The idea of decorating a hole with an environment is due to Talcott [19], and Mason [12] also used this idea in his calculus of contexts that has contexts as first-class values. However, in Mason's system, environments appear in a term containing holes only as annotations. This means that such environments are objects outside the system. We present our calculus $\lambda\kappa\varepsilon$ as an extension of $\lambda\varepsilon$ [18] which is a simply typed λ-calculus that has environments as first class values. So, environments are first-class objects in $\lambda\kappa\varepsilon$. Moreover, Mason defines hole filling only as a meta-level operation. Therefore, although contexts are first-class values in his system, one cannot compute hole filling within his system. In contrast to this, we can compute hole filling within our system. For example, we can express the above example of filling $a[\,]$ with $x + y$ as follows:

$$(\lambda X.\ \lambda x.\ (\lambda y.\ x + X{\cdot}\{x/x, y/y\})3)(\kappa\{x, y\}.\ x + y).$$

We can compute the above term in $\lambda\kappa\varepsilon$, and we get $\lambda x.\ x + (x + 3)$.

We now turn to another problem in the formalization of contexts. Consider the informal context $\lambda x.\ [\,]$. If we fill the hole in this context with x, we get the term $\lambda x.\ x$. This term is α-equivalent to $\lambda y.\ y$. What is the context which is α-equivalent to $\lambda x.\ [\,]$ and which, when filled with x, becomes $\lambda y.\ y$? It is certainly preferable that such a context exists, since, otherwise, hole filling and α-conversion will not always commute. A naïve attempt is to α-convert $\lambda x.\ [\,]$ to $\lambda y.\ [\,]$. But this does not work, since filling $\lambda y.\ [\,]$ with x results in $\lambda y.\ x$ which is not α-equivalent to $\lambda y.\ y$. We can solve this problem easily in our setting as follows. In $\lambda\kappa\varepsilon$, the context $\lambda x.\ [\,]$ is written as $\lambda x.\ X{\cdot}\{x/x\}$ and this context is α-equivalent to $\lambda y.\ X{\cdot}\{y/x\}$. Filling these holes in these two contexts with x is achieved by substituting $\kappa\{x\}.\ x$ for X in these contexts, and the results are $\lambda x.\ (\kappa\{x\}.\ x){\cdot}\{x/x\}$ and $\lambda y.\ (\kappa\{x\}.\ x){\cdot}\{y/x\}$ respectively. Then they are reduced to $\lambda x.\ x$ and $\lambda y.\ y$ as expected.

In this paper we also introduce a new method of defining substitution. As we explain below, the traditional method of defining substitution does not work for our calculus. We are therefore forced to use the new method, but, we believe our new definition of substitution is mathematically cleaner than the traditional method of defining substitution. We now give an example where the traditional method of defining substitution fails to work. By way of comparison, we first consider the λ term $a \equiv \lambda x.\ x + y$. What is the result of substituting x for y in a? We must be careful enough to avoid the variable clash and rename the bound variable x in a to a fresh variable, say, z, and we get $\lambda z.\ z + x$ as the result of the substitution. Now consider the abstract $b \equiv \kappa\{x\}.\ x + y$. What will be the result c of substituting x for y in b? If we perform substitution by the same method as above, we get $\kappa\{z\}.\ z + x$ which is wrong for the following reason. Note that $b{\cdot}\{2/x\}$ reduces to $2 + y$. So, $c{\cdot}\{2/x\}$ must reduce to $2 + x$. However, we cannot reduce $(\kappa\{z\}.\ z + x){\cdot}\{2/x\}$ since the argument $\{2/x\}$ does not match the binder $\kappa\{z\}$. By the same token, the term $(\kappa\{z\}.\ z + x){\cdot}\{2/x\}$ is not even typable. We can thus see that, unlike variables bound by the λ binder, we cannot rename variables bound by the κ binder. To cope with this situation, we introduce a new method of defining substitution where we rename *free* variables (if necessary) to achieve the capture avoiding substitution. So, our substitution will yield $\kappa\{x\}.\ x + \sharp x$ as the result of substituting x for y in b, where $\sharp x$ in the scope of the $\kappa\{x\}$ binder is a renamed form of the *free* variable x and it stands for the free variable x.

The paper is organized as follows. In section 2, we introduce the type system of $\lambda\kappa\varepsilon$, and introduce derivation rules that are used to define (typed) terms together with their types and free variables. There, we define variables so that they naturally contain both ordinary variables with names and variables as de Bruijn indices. In section 3, we define substitution as a meta-level operation. In section 4, we give reduction rules of $\lambda\kappa\varepsilon$ and give some examples of computations in $\lambda\kappa\varepsilon$. In section 5, we show that $\lambda\kappa\varepsilon$ enjoys a number of desirable properties such as confluence and strong normalizability. In section 6, we compare our calculus with some related works. In section 7, we give concluding remarks.

2 The Type System

In this section, we define the type system of $\lambda\kappa\varepsilon$ by defining the notion of a derivation of a typing judgment. A typing judgement is an expression of the form $\Gamma \vdash a : A$, and if it is derivable then it means that the expression a is a term whose type is A and whose set of free variables is Γ.

In the following we assume that we have given a finite set of atomic types which we do not specify further in this paper. We also assume that we have infinitely many *identifiers* (i). Then, we define *variables* and *types* simultaneously as follows.

A *variable* (we will use x, y, z as meta-variables for variables) is a triple $\langle k, i, A\rangle$ where k is a natural number, i is an identifier and A is a type. A variable $\langle k, i, A\rangle$ is called a *pure variable* if $k = 0$. *Types* (A, B) are defined by the following grammar:

$$A, B \ ::= \ K \mid E \mid A \Rightarrow B \mid A^E$$

where K ranges over atomic types and E over finite sets of variables.

In the following, we will use *declaration* as a synonym for a finite set of variables. We use E, F and Γ, Δ etc. as meta variables for declarations. A declaration $\{x_1, \ldots, x_n\}$ will also be called an *environment type* since it is the type of environments whose canonical forms are elements of the form $\{a_1/x_1, \ldots, a_n/x_n\}$.

If $x \equiv \langle k, i, A\rangle$, then we call k the *level* of x, i the *name* of x and A the *type* of x. In this case, we sometimes write x^A for x and also write $\sharp^l x$ for $\langle k + l, i, A\rangle$, $\sharp x$ for $\sharp^1 x$, and $\natural(x)$ for $\langle 0, i, A\rangle$. As we explained in the introduction, we need to distinguish different variables which have the same name, and we use levels for this purpose. Levels can be considered as a generalization of de Bruijn indices, as is shown by the example in the next section. Let x and y be variables. We write $x \le y$ if $y \equiv \sharp^l x$ for some $l \ge 0$.

We write V for the set of all the variables. Let E be a finite set of variables. We define $\Uparrow^E$ as the unique bijection from V to $\mathsf{V} - E$ which preserves the names of variables (that is, $\natural(\Uparrow^E(x)) = \natural(x)$ for any variable x) and is monotone in the sense that for any variables x, y such that $x \le y$ we have $\Uparrow^E(x) \le \Uparrow^E(y)$. We also define $\Downarrow_E$ as the inverse function of $\Uparrow^E$. Note that $\Downarrow_E(x)$ is defined only when $x \not\in E$. For example, if E is empty, then $\Uparrow^E(x) = x$ for any variable x. If E is $\{\sharp^1 x, \sharp^3 x\}$, then $\Uparrow^E(\sharp^0 x) = \sharp^0 x$, $\Uparrow^E(\sharp^2 x) = \sharp^4 x$, $\Uparrow^E(\sharp^3 x) = \sharp^5 x$, $\Downarrow_E(\sharp^2 x) = \sharp^1 x$, and $\Downarrow_E(\sharp^3 x)$ is undefined. We will use $\Uparrow^E$ later to systematically rename variables to avoid collision with the variables in E.

Let Γ, E, and F be declarations. We define the declarations $\Gamma{\Uparrow^E}$ and $\Gamma{\Downarrow_E}$ as follows:

$$\begin{array}{rcl}\Gamma{\Uparrow^E} & := & \{\Uparrow^E(x) \mid x \in \Gamma\},\\ \Gamma{\Downarrow_E} & := & \{\Downarrow_E(x) \mid x \in \Gamma\},\end{array}$$

where $\Gamma{\Downarrow_E}$ is defined only when $\Gamma \cap E$ is empty. Furthermore, given two declarations E and F, we define a function $\Updownarrow^E_F : \mathsf{V} \to \mathsf{V}$ as follows.

$$\Updownarrow^E_F(x) := \begin{cases} \Uparrow^E(x) & \text{if } x \in F,\\ \Downarrow_F(x) & \text{if } x \in E{\Uparrow^F},\\ \Uparrow^E(\Uparrow^F(\Downarrow_E(\Downarrow_F(x)))) & \text{otherwise.}\end{cases}$$

Since the above definition involves partial functions $\Downarrow_E$ and $\Downarrow_F$, the function $\Updownarrow^E_F$ may also be partial. But, the fact is that $\Updownarrow^E_F$ is a bijection from V to V, and we can see this as follows. We will be done if we can check the claim that $\Updownarrow^F_E(\Updownarrow^E_F(x)) = x$ holds for any $x \in \mathsf{V}$, since by the same argument we can see that $\Updownarrow^E_F(\Updownarrow^F_E(x)) = x$ holds for any $x \in \mathsf{V}$. We prove the claim by cases.

(i) If $x \in F$, then $\Updownarrow^F_E(\Updownarrow^E_F(x)) = \Updownarrow^F_E(\Uparrow^E(x)) = x$, since $\Uparrow^E(x) \in F{\Uparrow^E}$.

(ii) If $x \in E{\Uparrow^F}$, then there exists a $y \in E$ such that $x = \Uparrow^F(y)$. Since $\Downarrow_F(x)$ is defined and equal to y, we have

$$\Updownarrow^F_E(\Updownarrow^E_F(x)) = \Updownarrow^F_E(\Downarrow_F(x)) = \Updownarrow^F_E(y) = \Uparrow^F(y) = x.$$

(iii) Otherwise, $\Downarrow_F(x)$ is defined since $x \notin F$, and $\Downarrow_E(\Downarrow_F(x))$ is defined since $\Downarrow_F(x) \notin E$. Hence $\Updownarrow^E_F(x)$ is defined. It is easy to see $\Downarrow_F(\Downarrow_E(\Updownarrow^E_F(x)))$ is defined and equal to $\Downarrow_E(\Downarrow_F(x))$. Hence

$$\Updownarrow^F_E(\Updownarrow^E_F(x)) = \Uparrow^F(\Uparrow^E(\Downarrow_E(\Downarrow_F(x)))) = x.$$

We give a few examples here. If E is $\{x\}$, then $\Updownarrow^E_E(x) = \sharp x$, $\Updownarrow^E_E(\sharp x) = x$, and $\Updownarrow^E_E(\sharp^2 x) = \sharp^2 x$. If E is $\{x, \sharp x\}$ and F is $\{x\}$, then $\Updownarrow^E_F(x) = \sharp^2 x$, $\Updownarrow^E_F(\sharp x) = x$, $\Updownarrow^E_F(\sharp^2 x) = \sharp^1 x$, and $\Updownarrow^E_F(\sharp^3 x) = \sharp^3 x$.

Using the function $\Updownarrow^E_F$, we define the declaration $\Gamma{\Updownarrow^E_F}$ as follows.

$$\Gamma{\Updownarrow^E_F} \ := \ \{\Updownarrow^E_F(x) \mid x \in \Gamma\}.$$

As we will see in section 3, we will use $\Updownarrow^E_F$ to rename variables when the order of two binders are exchanged. The definition of $\Updownarrow^E_F$ may seem complicated, but if E and F are sets of pure variables (that is, bound variables are pure), the definition is simplified as follows.

Lemma 2.1 *Let E and F be sets of pure variables. We have*

$$\Updownarrow^E_F(x) = \begin{cases} \Uparrow^E(x) & \text{if } x \in F, \\ \Downarrow_F(x) & \text{if } x \in E\Uparrow^F, \\ x & \text{otherwise.} \end{cases}$$

Proof. Suppose $x \notin F$ and $x \notin E\Uparrow^F$. By $x \notin F$, there exists y such that $x = \Uparrow^F(y)$. Then, we have $\Uparrow^F(y) \notin E\Uparrow^F$, that is, $y \notin E$. Therefore, there exists z such that $y = \Uparrow^E(z)$. Then, we have $x = \Uparrow^F(\Uparrow^E(z))$ and $\Updownarrow^E_F(x) = \Uparrow^E(\Uparrow^F(\Downarrow_E(\Downarrow_F(x)))) = \Uparrow^E(\Uparrow^F(z))$. So, $\Uparrow^E(\Uparrow^F(z)) = \Uparrow^F(\Uparrow^E(z))$ implies $\Updownarrow^E_F(x) = x$. We will be done when we show $\Uparrow^E(\Uparrow^F(z)) = \Uparrow^F(\Uparrow^E(z))$ in the following.

Let n be the level of z. Then, there exists a pure variable z_0 such that $z = \sharp^n z_0$. Since E and F are sets of pure variables, we have

$$\Uparrow^E(\Uparrow^F(z)) = \begin{cases} \sharp^{n+2} z_0 & \text{if } z_0 \in E \cap F, \\ \sharp^{n+1} z_0 & \text{if } z_0 \in (E - F) \cup (F - E), \\ \sharp^n z_0 & \text{otherwise.} \end{cases}$$

So, we have $\Uparrow^E(\Uparrow^F(z)) = \Uparrow^F(\Uparrow^E(z))$. □

If E and F do not have common variable names (that is, E and F are 'irrelevant'), $\Updownarrow^E_F$ is an identity function.

Lemma 2.2 *Let E and F be declarations such that $\{\natural(x) \mid x \in E\} \cap \{\natural(x) \mid x \in F\} = \emptyset$. We have $\Updownarrow^E_F(x) = x$.*

Proof. In this proof, we write $\natural(E)$ for $\{\natural(x) \mid x \in E\}$. First, note that $\natural(x) \notin \natural(E)$ implies $\Uparrow^E(x) = \Downarrow_E(x) = x$. We prove the lemma by cases.

(i) If $x \in F$, then $\natural(x) \notin \natural(E)$ because $\natural(E) \cap \natural(F) = \emptyset$. Therefore, $\Updownarrow^E_F(x) = \Uparrow^E(x) = x$.

(ii) If $x \in E\Uparrow^F$, then $\natural(x) \in \natural(E\Uparrow^F) = \natural(E)$. Therefore, $\Updownarrow^E_F(x) = \Downarrow_F(x) = x$.

(iii) Otherwise, we have three cases. (1) If $\natural(x) \in \natural(E)$, then $\Updownarrow^E_F(x) = \Uparrow^E(\Uparrow^F(\Downarrow_E(\Downarrow_F(x)))) = \Uparrow^E(\Downarrow_E(x)) = x$. (2) If $\natural(x) \in \natural(F)$, then $\Updownarrow^E_F(x) = x$ is proved similarly. (3) Otherwise, trivial. $\square$

Lemma 2.3 *We have the following identities.*
1. $\Gamma\Uparrow^E \cap E = \emptyset$.
2. $\Gamma\Uparrow^E\Downarrow_E = \Gamma$. *If* $\Gamma \cap E = \emptyset$, *then* $\Gamma\Downarrow_E\Uparrow^E = \Gamma$.
3. $((\Gamma - E)\Downarrow_E - F)\Downarrow_F = ((\Gamma\Updownarrow^F_E - F)\Downarrow_F - E)\Downarrow_E$.

Proof. 1, 2. Easy.
3. Since $(\Gamma - E) \cap E = \emptyset$, there exists Γ' such that $\Gamma - E = \Gamma'\Uparrow^E$. Then, we have $\Gamma = (\Gamma \cap E) \cup \Gamma'\Uparrow^E$. Similarly, there exists Γ'' such that $\Gamma' = (\Gamma' \cap F) \cup \Gamma''\Uparrow^F$. Therefore, we have

$$\begin{aligned}((\Gamma - E)\Downarrow_E - F)\Downarrow_F &= (\Gamma'\Uparrow^E\Downarrow_E - F)\Downarrow_F \\ &= (\Gamma' - F)\Downarrow_F = \Gamma''\Uparrow^F\Downarrow_F = \Gamma''.\end{aligned}$$

On the other hand, we have

$$\begin{aligned}\Gamma\Updownarrow^F_E &= ((\Gamma \cap E) \cup (\Gamma' \cap F)\Uparrow^E \cup \Gamma''\Uparrow^F\Uparrow^E)\Updownarrow^F_E \\ &= (\Gamma \cap E)\Uparrow^F \cup (\Gamma' \cap F)\Uparrow^E\Downarrow_E \cup \Gamma''\Uparrow^F\Uparrow^E\Downarrow_E\Downarrow_F\Uparrow^E\Uparrow^F \\ &= (\Gamma \cap E)\Uparrow^F \cup (\Gamma' \cap F) \cup \Gamma''\Uparrow^E\Uparrow^F.\end{aligned}$$

So, we have

$$(\Gamma\Updownarrow^F_E - F)\Downarrow_F = ((\Gamma \cap E)\Uparrow^F \cup \Gamma''\Uparrow^E\Uparrow^F)\Downarrow_F = (\Gamma \cap E) \cup \Gamma''\Uparrow^E.$$

Therefore, we have

$$((\Gamma\Updownarrow^F_E - F)\Downarrow_F - E)\Downarrow_E = (\Gamma \cap E) \cup \Gamma''\Uparrow^E - E\Downarrow_E = \Gamma''\Uparrow^E\Downarrow_E = \Gamma''.$$

$\square$

A *typing judgment* is an expression of the form $\Gamma \vdash a : A$ where Γ is a declaration and A is a type. We have the typing rules in Figure 1 that are used to derive typing judgments, where those rules whose names end with 'I' ('E') introduce (eliminate, respectively) the types mentioned in the rule names.

An expression a is said to be a *term* if a typing judgment of the form $\Gamma \vdash a : A$ is derivable for some Γ and A. In this case, we say that Γ is the set

$$\frac{}{\{x\} \vdash x^A : A}\ (\mathsf{axiom})$$

$$\frac{\Gamma \vdash b : B}{(\Gamma - \{x\}) \Downarrow_{\{x\}} \vdash \lambda x^A.\, b : A \Rightarrow B}\ (\Rightarrow I) \qquad \frac{\Gamma \vdash b : A \Rightarrow B \quad \Delta \vdash a : A}{\Gamma \cup \Delta \vdash ba : B}\ (\Rightarrow E)$$

$$\frac{\Gamma \vdash a : A}{(\Gamma - E) \Downarrow_E \vdash \kappa E.\, a : A^E}\ (\mathsf{abs}I) \qquad \frac{\Gamma \vdash a : A^E \quad \Delta \vdash e : E}{\Gamma \cup \Delta \vdash a \cdot e : A}\ (\mathsf{abs}E)$$

$$\frac{\Gamma_1 \vdash a_1 : A_1 \quad \cdots \quad \Gamma_n \vdash a_n : A_n}{\Gamma_1 \cup \ldots \cup \Gamma_n \vdash \{a_1/x_1^{A_1}, \ldots, a_n/x_n^{A_n}\} : \{x_1, \ldots, x_n\}}\ (\mathsf{env}I)$$

$$\frac{\Gamma \vdash e : E \quad \Delta \vdash a : A}{\Gamma \cup (\Delta - E) \Downarrow_E \vdash e[\![a]\!] : A}\ (\mathsf{env}E)$$

In $(\mathsf{env}I)$, the variables $x_1, \ldots, x_n$ must be mutually distinct.

Figure 1: Typing rules of $\lambda\kappa\varepsilon$

of *free variables* in a and write $\mathrm{FV}(a)$ for it and also say that A is the *type* of a and write $\mathrm{TY}(a)$ for it. Note that if $e \equiv \{a_1/x_1, \ldots, a_n/x_n\}$, then the order of a_i/x_i $(i = 1, \ldots, n)$ in e does not matter and $\mathrm{TY}(e)$ is $\{x_1, \ldots, x_n\}$. We will say that these variables are *bound by* e. We also write T for the set of all the terms. A term is *canonical* if it is of the form $\lambda x.b$, $\{a_1/x_1, \ldots, a_n/x_n\}$ or $\kappa E.\, a$, that is, if it is obtained by one of the introduction rules. A term is said to be an *environment term* if its type is an environment type. A *canonical environment term* is a canonical term which is at the same time an environment term.

In $(\Rightarrow I)$, since free variables in b are within the scope of λx^A, $\Downarrow_{\{x\}}$ should be applied to $\Gamma - \{x\}$ to refer to the variables in b from the outside of the binder. By the same reason, $\Downarrow_E$ is used in $(\mathsf{abs}I)$ and $(\mathsf{env}E)$. We have explained the intuitive meaning of the typing rules $(\mathsf{abs}I)$ and $(\mathsf{abs}E)$ for introducing and eliminating abstractions in section 1. The remaining typing rules come from $\lambda\varepsilon$, and the reader is referred to [18] for the detailed explanation of these rules. Here, we only remark that the term $e[\![a]\!]$ in the

($\mathsf{env}E$) rule means to evaluate a in the environment e. So, for example, if

$$e \equiv \{\lambda x.\ \lambda y.\ x + y/z,\ 1/x,\ 2/u\},$$

then $e[\![zxy]\!]$ is evaluated to $1 + y$. Note that z and x in zxy are bound by e and y is free in $e[\![zxy]\!]$.

We give below a simple example of a derivation. In the example below, we assume that x and y are distinct pure variables.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{y\} \vdash y : A \Rightarrow A \Rightarrow B} \quad \dfrac{}{\{\sharp\sharp x\} \vdash \sharp\sharp x : A}}{\{y, \sharp\sharp x\} \vdash y(\sharp\sharp x) : A \Rightarrow B} \qquad \dfrac{}{\{x\} \vdash x : A}}{\dfrac{\{y, \sharp\sharp x, x\} \vdash y(\sharp\sharp x)x : B}{\{y, \sharp x\} \vdash \lambda x.\ y(\sharp\sharp x)x : A \Rightarrow B}}}{\{\} \vdash \kappa\{\sharp x, y\}.\ \lambda x.\ y(\sharp\sharp x)x : (A \Rightarrow B)^{\{\sharp x, y\}}}$$

It is easy to see that if $\Gamma \vdash a : A$ is derivable, then we can completely recover the entire derivation tree uniquely by inspecting the typed term a[2].

We have two kinds of abstractions λ and κ. As suggested by the type of κ-abstracts, κ abstracts named variables, while we intend that λ abstracts nameless variables. It is possible to eliminate names in λ-abstracts by taking the distinguished name ι, replacing λx^A by $\kappa\{\langle 0, \iota, A\rangle\}$, and using the de Bruijn index method that we explain in the next section. But we did not do so, because we want to design $\lambda\kappa\varepsilon$ so that it extends the traditional λ-calculus directly. However, unlike the traditional λ-calculus, we do not identify α-equivalent terms when we define substitution, but introduce α-equivalence later as an auxiliary notion that is necessary to describe conservativity over λ-calculus. (See also the comments after the definition of α-equivalence in section 3.)

3 Substitution and α-equivalence

In this section we define substitution as a meta-level syntactic operation. Our definition is conceptually simpler than the ordinary definition of substitution where α-conversion is sometimes necessary to avoid the unwanted capture of variables. Our method of defining substitution is a simple extension of the method due to de Bruijn [6].

[2]Strictly speaking, in order to have this property, we have to identify those derivations which are the same up to the difference of ordering of the premises of the ($\mathsf{env}I$) rule.

Before going into technical details, we explain our method by comparing it with the traditional method of defining substitution for terms with named variables [2] and also with the method invented by de Bruijn [6]. In the traditional method, for example, substitution of x for y in $\lambda x.\ y$ is done by first α-converting $\lambda x.\ y$ to, say, $\lambda z.\ y$ and then replacing y by x. Thus, the result of substitution is $\lambda z.\ x$. The α-conversion was necessary to avoid unwanted capturing of x by the λx binder in the original term. So, in this approach, one has to define terms as equivalence classes of concrete terms modulo α-equivalence, and therefore, we have to check the well-definedness of the substitution, since we first define the substitution operation on concrete terms. Also, in this approach, one has to define α-equivalence before substitution, but the definition of α-conversion requires the notion of renaming variables which is similar to substitution.

We think that such complication in the traditional definition of substitution comes from the fact that avoidance of capturing free variables was achieved by the renaming of the name of the λ-binder. Our approach here is to avoid the capture of free variables by systematically renaming the free variables which would otherwise be captured[3]. For instance, in case of the above example of substituting x for y in $\lambda x.\ y$, we rename x to $\sharp x$ and substitute $\sharp x$ for y, so that the result of the substitution becomes $\lambda x.\ \sharp x$. We note that in the resulting term $\lambda x.\ \sharp x$, the variable $\sharp x$ is different from x within the scope of λx, and that $\sharp x$ refers to x outside the scope of λx. From this explanation, it should be easy to understand that the result of substituting $x * z$ for y in $\lambda x.\ \lambda x.\ x + y$ is $\lambda x.\ \lambda x.\ x + (\sharp^2 x * z)$. As can be seen from this example, we rename only those variables that would otherwise be captured. Therefore, in case capturing does not occur, the result of substitution obtained by our method is the same as that obtained by the traditional method.

After defining the substitution, we define α-equivalence of terms. As we will see later, our definition of α-equivalence is also simpler than the existing definitions of α-equivalence, and the proof that the relation is indeed an equivalence relation is also simpler. Again, some examples will clarify the intuitive idea. Suppose that x, y, z are distinct pure variables. Then the following terms are all α-equivalent with each other.

$$\begin{aligned} &\lambda x.\ \lambda y.\ (\lambda z.\ y(zx))(yx) \\ \equiv_\alpha\ &\lambda \sharp^1 x.\ \lambda \sharp^1 x.\ (\lambda \sharp^1 x.\ \sharp^2 x(\sharp^1 x \sharp^3 x))(\sharp^1 x \sharp^2 x) \end{aligned}$$

[3]The idea of renaming free variables is introduced in [17].

$$\begin{aligned}
&\equiv_\alpha \lambda\sharp^1 x.\ \lambda\sharp^2 x.\ (\lambda\sharp^3 x.\ \sharp^2 x(\sharp^3 x\sharp^1 x))(\sharp^2 x\sharp^1 x)\\
&\equiv_\alpha \lambda\sharp^2 x.\ \lambda\sharp^1 x.\ (\lambda\sharp^0 x.\ \sharp^2 x(\sharp^0 x\sharp^4 x))(\sharp^1 x\sharp^3 x)\\
&\equiv_\alpha \lambda x.\ \lambda y.\ (\lambda x.\ y(z\sharp x))(yx).
\end{aligned}$$

If we write $1, 2$ and 3 for $\sharp^1 x, \sharp^2 x$ and $\sharp^3 x$, respectively, in the second term above, we get $\lambda 1.\ \lambda 1.\ (\lambda 1.\ 2(13))(12)$. Therefore, this term is essentially the same as the representation of the first term in de Bruijn indices. Similarly, the third term becomes $\lambda 1.\ \lambda 2.\ (\lambda 3.\ 2(31))(21)$, and this term is essentially the same as the representation of the first term in de Bruijn levels, or, using the terminology of Gunter [9], corresponds to the CCC model representation. We can therefore see that our terms are natural extensions of both traditional concrete terms with variable names and name free terms a la de Bruijn that use indices and levels.

Let $\phi : \mathsf{V} \to \mathsf{V}$ be a (possibly) partial function such that $\phi(x)$ may be undefined for some $x \in \mathsf{V}$. We extend this function to the function $\overline{\phi} : \mathsf{T} \to \mathsf{T}$ as follows. $\overline{\phi}$ will be total if and only if ϕ is total.

1. $\overline{\phi}(x) := \phi(x)$.
2. $\overline{\phi}(\lambda x.\ a) := \lambda x.\ \overline{\phi_{\{x\}}}(a)$.
3. $\overline{\phi}(ba) := \overline{\phi}(b)\overline{\phi}(a)$.
4. $\overline{\phi}(\kappa F.\ a) := \kappa F.\ \overline{\phi_F}(a)$.
5. $\overline{\phi}(a{\cdot}f) := \overline{\phi}(a){\cdot}\overline{\phi}(f)$.
6. $\overline{\phi}(\{a_1/x_1, \ldots, a_n/x_n\}) := \{\overline{\phi}(a_1)/x_1, \ldots, \overline{\phi}(a_n)/x_n\}$.
7. $\overline{\phi}(e[\![a]\!]) := \overline{\phi}(e)[\![\overline{\phi_{\mathrm{TY}(e)}}(a)]\!]$.

where, for each declaration E, $\phi_E : \mathsf{V} \to \mathsf{V}$ is defined by

$$\phi_E(x) := \begin{cases} x & \text{if } x \in E, \\ \Uparrow^E(\phi(\Downarrow_E(x))) & \text{otherwise.} \end{cases}$$

(We note that ϕ is total if and only if ϕ_E is total.)

We define the push operation $\uparrow^E$ by putting $a{\uparrow^E} := \overline{\Uparrow^E}(a)$, the pull operation $\downarrow_E$ by putting $a{\downarrow_E} := \overline{\Downarrow_E}(a)$, and the exchange operation $\updownarrow^E_F$ by putting $a{\updownarrow^E_F} := \overline{\Updownarrow^E_F}(a)$.

Let us give a few examples here. Let E be $\{x, \sharp x\}$.

$$\begin{array}{rcl}(\lambda x.\ x(\sharp^3 x))\uparrow^E & \equiv & \overline{\Uparrow^E}(\lambda x.\ x(\sharp^3 x)) \\ & \equiv & \lambda x.\ \overline{\Uparrow^E{}_{\{x\}}}(x(\sharp^3 x)) \\ & \equiv & \lambda x.\ (\overline{\Uparrow^E{}_{\{x\}}}(x))(\overline{\Uparrow^E{}_{\{x\}}}(\sharp^3 x)) \\ & \equiv & \lambda x.\ x(\Uparrow^{\{x\}}(\Uparrow^E(\Downarrow_{\{x\}}(\sharp^3 x)))) \\ & \equiv & \lambda x.\ x(\sharp^5 x).\end{array}$$

Note that $\mathrm{FV}(\lambda x.\ x(\sharp^3 x)) = \{\sharp^2 x\}$, and $\{\sharp^2 x\}\Uparrow^E = \{\sharp^4 x\}$, which is equal to $\mathrm{FV}(\lambda x.\ x(\sharp^5 x))$. Similarly, we have $(\lambda x.\ x(\sharp^3 x))\downarrow_E \equiv \lambda x.\ x(\sharp x)$.

For the exchange operation, we have:

$$\begin{array}{rcl}(\lambda x.\ (\sharp x)(\sharp^2 x))\updownarrow^{\{x\}}_{\{x\}} & \equiv & \lambda x.\ (\sharp^2 x)(\sharp x), \\ (\lambda x.\ (\sharp x)(\sharp^2 x))\updownarrow^{\{y\}}_{\{x\}} & \equiv & \lambda x.\ (\sharp x)(\sharp^2 x),\end{array}$$

where x and y are distinct pure variables.

We now define the substitution operation as follows. Let $s \equiv \{c_1/x_1, \ldots, c_n/x_n\}$ be a canonical environment term. Note that $\mathrm{TY}(s) = \{x_1, \ldots, x_n\}$ in this case. For each term a we define a term $a[s]$ inductively as follows.

1. $x[s] := \begin{cases} c_i & \text{if } x \equiv x_i \text{ for some } i, \\ \Downarrow_{\mathrm{TY}(s)}(x) & \text{otherwise.} \end{cases}$

2. $(\lambda x.\ b)[s] := \lambda x.\ b\updownarrow^{\mathrm{TY}(s)}_{\{x\}}[s\uparrow^{\{x\}}]$.

3. $(ba)[s] := b[s]a[s]$.

4. $(\kappa E.\ a)[s] := \kappa E.\ a\updownarrow^{\mathrm{TY}(s)}_{E}[s\uparrow^{E}]$.

5. $(a{\bullet}e)[s] := a[s]{\bullet}e[s]$.

6. $(\{a_1/x_1, \ldots, a_n/x_n\})[s] := \{a_1[s]/x_1, \ldots, a_n[s]/x_n\}$.

7. $(e[\![a]\!])[s] := e[s][\![(a\updownarrow^{\mathrm{TY}(s)}_{\mathrm{TY}(e)})[s\uparrow^{\mathrm{TY}(e)}]]\!]$.

We call $a[s]$ the result of *substituting* $c_1, \ldots, c_n$ *for* $x_1, \ldots, x_n$ *in* a.

Again we give a few examples. Let s be $\{\sharp^3 x/x,\ (x\ \sharp x)/\sharp x\}$. Then we have:

$$(\lambda x.\ x\ \sharp^2 x)[s] \equiv \lambda x.\ (x\ \sharp^2 x)\updownarrow^{\{x,\sharp x\}}_{\{x\}}[s\uparrow^{\{x\}}]$$

$$
\begin{aligned}
&\equiv \lambda x.\ (\sharp^2 x\ \sharp x)[\{\sharp^4 x/x,\ (\sharp x\ \sharp^2 x)/\sharp x\}]\\
&\equiv \lambda x.\ (\Downarrow_{\{x,\sharp x\}}(\sharp^2 x))(\sharp x\ \sharp^2 x)\\
&\equiv \lambda x.\ x(\sharp x\ \sharp^2 x),\\
(\lambda x.\ xy)[\{z/y\}] &\equiv \lambda x.\ xz,
\end{aligned}
$$

where x, y, z are distinct pure variables. In the first example, x in $\lambda x.\ x\ \sharp^2 x$ is bound by λx and $\sharp^2 x$ is bound by s. When s goes into the scope of λx, x and $\sharp^2 x$ should be renamed to $\sharp^2 x$ and $\sharp x$, respectively so that they are bound by λx and $s{\uparrow}^{\{x\}}$.

We are now ready to define the notion of α-equivalence. We prepare an infinite set of identifiers $\iota_1, \iota_2, \ldots$ that are not used in $\lambda\kappa\varepsilon$. We write ι_k^A or just ι_k for a variable $\langle 0, \iota_k, A\rangle$. An extended term is a term constructed from variables of $\lambda\kappa\varepsilon$ and variables $\iota_1, \iota_2, \ldots$. We also assume that we have a total order $\prec$ on variables. We define a transformation ${}^{*\lambda}$ that transforms a term t to an extended term $t^{*\lambda}$ using the following rules.

1. $x^{*\lambda} := x$.
2. $(\lambda x.\ a)^{*\lambda} := \lambda\iota_1.\ a^{*\lambda}[\{\iota_1/x\}]$.
3. $(ba)^{*\lambda} := b^{*\lambda}a^{*\lambda}$.
4. $(\kappa E.\ a)^{*\lambda} := \kappa E.\ a^{*\lambda}$.
5. $(a{\bullet}f)^{*\lambda} := a^{*\lambda}{\bullet}f^{*\lambda}$.
6. $\{a_1/x_1, \ldots, a_n/x_n\}^{*\lambda} := \{a_1^{*\lambda}/x_1, \ldots, a_n^{*\lambda}/x_n\}$.
7. $(e[\![b]\!])^{*\lambda} := e^{*\lambda}[\![b^{*\lambda}]\!]$.

We define another transformation ${}^{*\varepsilon}$ using the rules obtained by replacing ${}^{*\lambda}$ of rules 1–6 by ${}^{*\varepsilon}$ and the following rules.

7. $(e[\![b]\!])^{*\varepsilon} := e^{*\varepsilon}[\![b^{*\varepsilon}]\!]$, if e is not a canonical environment term.
8. $(\{a_1/x_1, \ldots, a_n/x_n\}[\![b]\!])^{*\varepsilon}$
   $:=\{a^{*\varepsilon}_{p(1)}/\iota_1, \ldots, a^{*\varepsilon}_{p(n)}/\iota_n\}[\![b^{*\varepsilon}[\{\iota_1/x_{p(1)}, \ldots, \iota_n/x_{p(n)}\}]]\!]$
   where p is a permutation on $1..n$ such that $x_{p(1)} \prec \cdots \prec x_{p(n)}$ and the type of ι_k is the type of $x_{p(k)}$.

Let a and b be terms. If $a^{*\lambda} \equiv b^{*\lambda}$, we write $a \equiv_\alpha^\lambda b$ and say *a is α^λ-equivalent to b.* If $a^{*\varepsilon} \equiv b^{*\varepsilon}$, we write $a \equiv_\alpha^\varepsilon b$ (or $a \equiv_\alpha b$) and say *a is α^ε-equivalent to b* (or *a is α-equivalent to b*). In the following, we apply ${}^{*\lambda}$ and ${}^{*\varepsilon}$ to extended terms (the definition should be clear), but we do not apply the notion of α^λ- or α^ε-equivalence to extended terms.

We have introduced α^λ-equivalence to expresses our intention that λ abstracts nameless variables. But it is not satisfactory because α^λ-equivalence is not preserved by reduction. Therefore, we have introduced α^ε-equivalence as an equivalence that is preserved by reduction. (See Theorem 5.17.)

The α-equivalence in $\lambda\kappa\varepsilon$ also satisfies the following properties:

Theorem 3.1 *1. If $\Gamma \vdash a : A$ and $a \equiv_\alpha^\lambda b$, then $\Gamma \vdash b : A$.*
2. If $\Gamma \vdash a : A$ and $a \equiv_\alpha^\varepsilon b$, then $\Gamma \vdash b : A$.
3. Let a and b be terms, s be a canonical environment term, and ρ be λ or ε. If $a \equiv_\alpha^\rho b$, then $a[s] \equiv_\alpha^\rho b[s]$.

Proof. We can prove 1 and 2 by the induction on the derivation $\Gamma \vdash a : A$. We will prove 3 in section 5, because we need some properties of substitution and α-equivalence to prove this theorem. $\Box$

Here, we refer to some preceding works that have relations with our exchange operator and our definition of substitution. The definition of substitution by Fiore, Plotkin, and Turi [7] is similar to ours, that is, the 2nd clause of our definition corresponds to the 2nd clause of the definition of the substitution operation $\sigma : \delta\Lambda \times \Lambda \to \Lambda$ [7], and we think that it should be possible to establish a precise correspondence. We also make a remark about label-selective λ-calculus [8]. In label-selective λ-calculus, abstracts and arguments are labeled and the labels are used in argument-passing. The label consists of a symbol and a number and the number changes when the label is pushed into the scope of λ-abstracts. Since argument-passing corresponds to selecting the field of record by label, we can say label-selective λ-calculus incorporates an idea similar to ours.

4 Reduction Rules

In this section we give reduction rules of the $\lambda\kappa\varepsilon$ calculus. We first define $\mapsto_{\lambda\kappa\varepsilon}$ as the union of the following three relations $\mapsto_\lambda$, $\mapsto_\kappa$ and $\mapsto_\varepsilon$.

The relation $\mapsto_\lambda$ is defined by the following single rule:

($\boldsymbol{\lambda}$) $(\lambda x.\, b)a \mapsto_\lambda \{a/x\}[\![b]\!]$,

and the relation $\mapsto_\kappa$ is defined by the following single rule:

($\boldsymbol{\kappa}$) $(\kappa E.\, a)\cdot e \mapsto_\kappa e[\![a]\!]$.

The relation $\mapsto_\varepsilon$ is defined by the following 8 conversion rules.

(gc) $e[\![a]\!] \mapsto_\varepsilon a{\downarrow_{\mathrm{TY}(e)}}$, if $\mathrm{TY}(e) \cap \mathrm{FV}(a) = \emptyset$.

(var) $\{a_1/x_1, \ldots, a_n/x_n\}[\![x_i]\!] \mapsto_\varepsilon a_i$ $(1 \leq i \leq n)$.

(fun) $e[\![\lambda x.\, b]\!] \mapsto_\varepsilon \lambda x.\, (e{\uparrow^{\{x\}}})[\![b{\updownarrow^{\mathrm{TY}(e)}_{\{x\}}}]\!]$.

(funapp) $e[\![ba]\!] \mapsto_\varepsilon e[\![b]\!]e[\![a]\!]$.

(abs) $e[\![\kappa E.\, a]\!] \mapsto_\varepsilon \kappa E.\, (e{\uparrow^{E}})[\![a{\updownarrow^{\mathrm{TY}(e)}_{E}}]\!]$.

(absapp) $e[\![a\cdot f]\!] \mapsto_\varepsilon e[\![a]\!]\cdot e[\![f]\!]$.

(env) $e[\![\{a_1/x_1, \ldots, a_n/x_n\}]\!] \mapsto_\varepsilon \{e[\![a_1]\!]/x_1, \ldots, e[\![a_n]\!]/x_n\}$.

(eval) $e[\![f[\![x]\!]]\!] \mapsto_\varepsilon e[\![f]\!][\![x]\!]$, if $x \in \mathrm{TY}(f)$.

The rules other than (eval) are internalized forms of the clauses 1–6 of the definition of substitution in section 3. In these rules we have the environment term e in place of the canonical environment term s, and the rule (gc) is a generalization of the second case of clause 1. We can also internalize clause 7 directly and get a correct rule. But, we do not do so since it will result in a system where the strong normalization property does not hold. Instead we have the (eval) rule which corresponds to a special case of clause 7. Although the (eval) rule is a weak version of clause 7, we will see in Theorem 5.9 that we can faithfully compute substitution internally by using these reduction rules, and at the same time the system enjoys the strong normalizability (Theorem 5.26). In fact, as can be seen in, e.g., Melliès [13] and Bloo and Rose [3], the strong normalizability of calculi of explicit substitutions and explicit environments is a subtle problem. The reader is referred to [18] for a detailed discussion on our choice of the (eval) rule.

We write $a \to_\lambda b$ if b is obtained from a by replacing a subterm c in a by d such that $c \mapsto_\lambda d$. Similarly $\to_\kappa$, $\to_\varepsilon$ and $\to_{\lambda\kappa\varepsilon}$ are defined. The reflexive and transitive closures of these reductions are denoted with asterisk (*), such as $\overset{*}{\to}_\varepsilon$. The equivalence relation generated by $\to_{\lambda\kappa\varepsilon}$ is denoted by $=_{\lambda\kappa\varepsilon}$, namely, the reflexive, symmetric, and transitive closure of $\to_{\lambda\kappa\varepsilon}$. Similarly $=_\varepsilon$ is defined.

We give a few examples of reduction sequences. ($s \equiv \{\sharp^3 x/x,\ (x\ \sharp x)/\sharp x\}$ in the second example.)

$$\begin{aligned}
(\lambda x.\ \lambda y.\ x)y &\rightarrow_\lambda \{y/x\}[\![\lambda y.\ x]\!] \\
&\rightarrow_\varepsilon \lambda y.\ (\{y/x\}{\uparrow^{\{y\}}})[\![x {\updownarrow^{\{x\}}_{\{y\}}}]\!] \\
&\equiv \quad \lambda y.\ \{\sharp y/x\}[\![x]\!] \rightarrow_\varepsilon \lambda y.\ \sharp y.
\end{aligned}$$

$$\begin{aligned}
s[\![\lambda x.\ x\ \sharp^2 x]\!] &\rightarrow_\varepsilon \lambda x.\ (s{\uparrow^{\{x\}}})[\![(x\ \sharp^2 x){\updownarrow^{\{x,\sharp x\}}_{\{x\}}}]\!] \\
&\equiv \quad \lambda x.\ \{\sharp^4 x/x,\ (\sharp x\ \sharp^2 x)/\sharp x\}[\![(\sharp^2 x\ \sharp x)]\!] \\
&\overset{*}{\rightarrow}_\varepsilon \lambda x.\ ((\sharp^2 x){\downarrow_{\{x,\sharp x\}}})(\sharp x\ \sharp^2 x) \\
&\equiv \quad \lambda x.\ x(\sharp x\ \sharp^2 x).
\end{aligned}$$

$$\begin{aligned}
(\lambda X.\ \lambda y.\ X{\cdot}\{y/y\})(\kappa\{y\}.\ y) &\rightarrow_\lambda \{\kappa\{y\}.\ y/X\}[\![\lambda y.\ X{\cdot}\{y/y\}]\!] \\
&\overset{*}{\rightarrow}_\varepsilon \lambda y.\ (\kappa\{y\}.\ y){\cdot}\{y/y\} \\
&\rightarrow_\kappa \lambda y.\ \{y/y\}[\![y]\!] \overset{*}{\rightarrow}_\varepsilon \lambda y.\ y.
\end{aligned}$$

In the first example, y is renamed to $\sharp y$ so that it is not captured by the λy binder. The second example corresponds to the example given after the definition of substitution. The third example shows the hole-filling operation where y is captured by the λy binder.

We take an example from Hashimoto-Ohori's paper [10]. Consider the term $(\lambda z.\ C[x+z])x$ where C is an (informal) context $(\lambda x.\ [\ ] + y)3$ and $C[x+z]$ represents the hole-filling operation in the λ-calculus. In Hashimoto-Ohori's calculus, this term can be written as

$$a \equiv (\lambda z.\ (\delta X.(\lambda u.\ X^{\{u/x\}} + y)3) \odot_{\{x/v\}} (v+z))x$$

where X represents a hole, δX abstracts the hole X, and $\odot$ is a hole-filling operator. $\{u/x\}$ and $\{x/v\}$ (called renamers) annotate X and $\odot$ respectively. They are introduced to solve the problem of variable capturing. In our system, the above term can be written as

$$a \equiv (\lambda z.\ (\lambda X.\ (\lambda u.\ X{\cdot}\{u/x\} + y)3)(\kappa\{x\}.\ (x+z)))x.$$

We can compute this term in many ways, but, here we give three reduction sequences.

$$
\begin{aligned}
a &\to_\lambda \{x/z\}[\![(\lambda X.\ (\lambda u.\ X\cdot\{u/x\} + y)3)(\kappa\{x\}.\ x + z)]\!]\\
&\xrightarrow{*}_\varepsilon (\lambda X.\ (\lambda u.\ X\cdot\{u/x\} + y)3)(\kappa\{x\}.\ x + \sharp x)\\
&\to_\lambda \{\kappa\{x\}.\ x + \sharp x/X\}[\![(\lambda u.\ X\cdot\{u/x\} + y)3]\!]\\
&\xrightarrow{*}_\varepsilon (\lambda u.\ (\kappa\{x\}.\ x + \sharp x)\cdot\{u/x\} + y)3\\
&\to_\kappa (\lambda u.\ \{u/x\}[\![x + \sharp x]\!] + y)3\\
&\xrightarrow{*}_\varepsilon (\lambda u.\ u + x + y)3\\
&\to_\lambda \{3/u\}[\![u + x + y]\!] \xrightarrow{*}_\varepsilon 3 + x + y
\end{aligned}
$$

$$
\begin{aligned}
a &\to_\lambda (\lambda z.\ \{\kappa\{x\}.\ x + z/X\}[\![(\lambda u.\ X\cdot\{u/x\} + y)3]\!])x\\
&\xrightarrow{*}_\varepsilon (\lambda z.\ (\lambda u.\ (\kappa\{x\}.\ x + z)\cdot\{u/x\} + y)3)x\\
&\to_\lambda \{x/z\}[\![(\lambda u.\ (\kappa\{x\}.\ x + z)\cdot\{u/x\} + y)3]\!]\\
&\xrightarrow{*}_\varepsilon (\lambda u.\ (\kappa\{x\}.\ x + \sharp x)\cdot\{u/x\} + y)3\\
&\to_\lambda \{3/u\}[\![(\kappa\{x\}.\ x + \sharp x)\cdot\{u/x\} + y]\!]\\
&\xrightarrow{*}_\varepsilon (\kappa\{x\}.\ x + \sharp x)\cdot\{3/x\} + y\\
&\to_\kappa \{3/x\}[\![x + \sharp x]\!] + y \xrightarrow{*}_\varepsilon 3 + x + y
\end{aligned}
$$

$$
\begin{aligned}
a &\to_\lambda (\lambda z.\ (\lambda X.\ \{3/u\}[\![X\cdot\{u/x\} + y]\!])(\kappa\{x\}.\ x + z))x\\
&\xrightarrow{*}_\varepsilon (\lambda z.\ (\lambda X.\ X\cdot\{3/x\} + y)(\kappa\{x\}.\ x + z))x\\
&\to_\lambda (\lambda z.\ \{\kappa\{x\}.\ x + z/X\}[\![X\cdot\{3/x\} + y]\!])x\\
&\xrightarrow{*}_\varepsilon (\lambda z.\ (\kappa\{x\}.\ x + z)\cdot\{3/x\} + y)x\\
&\to_\kappa (\lambda z.\ \{3/x\}[\![x + z]\!] + y)x\\
&\xrightarrow{*}_\varepsilon (\lambda z.\ 3 + z + y)x\\
&\to_\lambda \{x/z\}[\![3 + z + y]\!] \xrightarrow{*}_\varepsilon 3 + x + y
\end{aligned}
$$

We remark that, in the second reduction sequence above, we have first reduced the innermost β-redex $(\lambda u.\ X\cdot\{u/x\} + y)3$. Such a reduction is not possible in Hashimoto-Ohori's calculus since in their system the β-conversion is prohibited when the redex contains a free hole. Though the roles of $X^{\{u/x\}}$

and $X\cdot\{u/x\}$ are similar, u in $X^{\{u/x\}}$ should always be a variable, while u in $X\cdot\{u/x\}$ can be substituted by an arbitrary term. This is the reason why our calculus need not put any restriction to the ($\boldsymbol{\lambda}$)-reduction rule (the β-conversion).

We also remark on the hole-filling operations without going into the technical details. In Hashimoto-Ohori's calculus, the renamer ν in $\odot_\nu$ works as a variable binder to the second operand of $\odot$ (i.e. to the term to be filled into the hole). Because their typing rule of $M \odot_\nu N$ causes a side effect to the type of the free hole in N, they had to put the restriction that each free hole may occur at most once. Our κE binder, which plays the similar role to the renamer ν in $\odot_\nu$, does not have such a problem, because it is merely an abstraction.

Therefore, our calculus $\lambda\kappa\varepsilon$ can be regarded as a natural and flexible extension to Hashimoto-Ohori's calculus.

5 Properties of $\lambda\kappa\varepsilon$

In this section, we show that $\lambda\kappa\varepsilon$ enjoys a number of desirable properties. First, we give an alternative definition of substitution (Lemma 5.5) which is useful in proving the properties of $\lambda\kappa\varepsilon$. Then, we show that the meta-level operation of substitution is internally realized by the operation of evaluation (Theorem 5.9). We also show that $\lambda\kappa\varepsilon$ enjoys subject reduction property (Theorem 5.13), confluence property (Theorem 5.14), conservativity over the simply typed $\lambda\beta$-calculus (Theorem 5.24), and strong normalizability (Theorem 5.26). Theorems 5.14, 5.24, 5.26 establish the purity of $\lambda\kappa\varepsilon$, and as a corollary to the confluence of $\lambda\kappa\varepsilon$, we see that the operations of hole filling and β-reduction always commute.

It is sometimes useful to define the substitution operation in a similar way as push/pull/exchange operations. Let $\phi : \mathsf{V} \to \mathsf{T}$ be a (possibly) partial function. We extend ϕ to the function $\overline{\phi} : \mathsf{T} \to \mathsf{T}$ in the same way as $\overline{\phi}$ defined in section 3, except that ϕ_E is now defined as follows.

$$\phi_E(x) := \begin{cases} x & \text{if } x \in E, \\ \phi(x\downarrow_E)\uparrow^E & \text{otherwise.} \end{cases}$$

It is easy to see that if ϕ is total, then $\overline{\phi}$ is also a total function. That is, if $a \in \mathsf{T}$, we have $\overline{\phi}(a) \in \mathsf{T}$.

Let s be $\{b_1/x_1, \ldots, b_n/x_n\}$. We define a total function $[s] : \mathsf{V} \to \mathsf{T}$ as follows:

$$[s](x) := \begin{cases} b_i & \text{if } x \equiv x_i \text{ for some } i, \\ \Downarrow_{\{x_1,\ldots,x_n\}}(x) & \text{otherwise.} \end{cases}$$

We give an example of this alternative definition of substitution. Let s be $\{\sharp^3 x/x,\ (x\ \sharp x)/\sharp x\}$.

$$\begin{aligned} \overline{[s]}(\lambda x.\ x\ \sharp^2 x) &\equiv \lambda x.\ \overline{[s]_{\{x\}}}(x\ \sharp^2 x) \\ &\equiv \lambda x.\ ([s]_{\{x\}}(x))([s]_{\{x\}}(\sharp^2 x)) \\ &\equiv \lambda x.\ x([s](\sharp^2 x \downarrow_{\{x\}}) \uparrow^{\{x\}}) \\ &\equiv \lambda x.\ x([s](\sharp x) \uparrow^{\{x\}}) \\ &\equiv \lambda x.\ x((x\ \sharp x) \uparrow^{\{x\}}) \\ &\equiv \lambda x.\ x(\sharp x\ \sharp^2 x). \end{aligned}$$

We show in Lemma 5.5 that $\overline{[s]}(a)$ coincides with the substitution $a[s]$ defined in section 3. In the following, we will write $\phi_{E_1 \cdots E_n}$ for $(\cdots(\phi_{E_1})\cdots)_{E_n}$ and sometimes just write ϕ for $\overline{\phi}$, and ϕa for $\phi(a)$, if there is no fear of confusion.

Lemma 5.1 *Let $\phi : \mathsf{V} \to \mathsf{T}$ be a (possibly) partial function, a be a term, and E, E_1, ..., E_n, F be declarations.*
1. If $\Downarrow_{E_{i+1}} \cdots \Downarrow_{E_n} x \in E_i$ for some i, then $\phi_{E_1 \cdots E_n}(x) \equiv x$.
2. $\phi_{E_1 \cdots E_n}(\Uparrow^{E_n} \cdots \Uparrow^{E_1} x) \equiv \phi(x) \uparrow^{E_1} \cdots \uparrow^{E_n}$.
3. $\Updownarrow^E_F \Uparrow^F \Uparrow^E x \equiv \Uparrow^E \Uparrow^F x$.

Proof. 1, 2. Straightforward.
3. Note that $\Uparrow^F \Uparrow^E x \notin F$ and $\Uparrow^F \Uparrow^E x \notin E \Uparrow^F$. □

Lemma 5.2 *Let $\phi : \mathsf{V} \to \mathsf{T}$ be a (possibly) partial function, a be a term, and E, E_1, ..., E_n be declarations. Then, $\overline{\phi_E}(a \uparrow^E) \equiv \overline{\phi}(a) \uparrow^E$ holds. As a corollary, $\overline{\phi_{E_1 \cdots E_n}}(a \uparrow^{E_1} \cdots \uparrow^{E_n}) \equiv \overline{\phi}(a) \uparrow^{E_1} \cdots \uparrow^{E_n}$.*

Proof. First, we prove the lemma in the case ϕ is restricted to $\psi : \mathsf{V} \to \mathsf{V}$. We prove, by induction on the construction of a, that the identity

$$\psi_{E\overline{F}}(\Uparrow^E{}_{\overline{F}}(a)) \equiv \Uparrow^E{}_{\overline{F}}(\psi_{\overline{F}}(a)) \qquad (\star)$$

holds for any sequence of declarations $\overline{F} \equiv F_1 \cdots F_n$.

1. *a is a variable*:

   (a) *$\Downarrow_{F_{i+1}} \cdots \Downarrow_{F_n} a \in F_i$ for some i*: In this case, both sides of $(\star)$ are a by Lemma 5.1 clause 1.

   (b) *Otherwise*: In this case, there is a variable y such that $a \equiv \Uparrow^{F_n} \cdots \Uparrow^{F_1} y$. By Lemma 5.1 clause 2, we have

$$\begin{aligned} \text{LHS of } (\star) &\equiv \psi_{E\overline{F}}(\Uparrow^{F_n} \cdots \Uparrow^{F_1} \Uparrow^{E} y) \equiv \Uparrow^{F_n} \cdots \Uparrow^{F_1} \Uparrow^{E} (\psi(y)), \\ \text{RHS of } (\star) &\equiv {\Uparrow^{E}}_{\overline{F}} \Uparrow^{F_n} \cdots \Uparrow^{F_1} (\psi(y)) \equiv \Uparrow^{F_n} \cdots \Uparrow^{F_1} \Uparrow^{E} (\psi(y)). \end{aligned}$$

   (The last $\equiv$ holds because $\psi(y)$ is a variable.)

2. $a \equiv e[\![b]\!]$: We have

$$\begin{aligned} \text{LHS of } (\star) &\equiv \psi_{E\overline{F}}({\Uparrow^{E}}_{\overline{F}}(e))[\![\psi_{E\overline{F}\mathrm{TY}(e)}({\Uparrow^{E}}_{\overline{F}\mathrm{TY}(e)}(b))]\!], \\ \text{RHS of } (\star) &\equiv {\Uparrow^{E}}_{\overline{F}}(\psi_{\overline{F}}(e))[\![{\Uparrow^{E}}_{\overline{F}\mathrm{TY}(e)}(\psi_{\overline{F}\mathrm{TY}(e)}(b))]\!]. \end{aligned}$$

   Therefore, by the induction hypothesis, we have the identity.

3. *Otherwise*: Similar to the case 2 or trivial.

Next, we prove the lemma in the general case. Similarly to the restricted case, we prove $\phi_{E\overline{F}}({\Uparrow^{E}}_{\overline{F}}(a)) \equiv {\Uparrow^{E}}_{\overline{F}}(\phi_{\overline{F}}(a))$ by induction on a. The induction proceeds in exactly the same way except that, in case 1-(b), we need the lemma in the restricted case to show

$${\Uparrow^{E}}_{\overline{F}} \Uparrow^{F_n} \cdots \Uparrow^{F_1} (\phi(y)) \equiv \Uparrow^{F_n} \cdots \Uparrow^{F_1} \Uparrow^{E} (\phi(y)),$$

because $\phi(y)$ is a term. (${\Uparrow^{E}}_{F_1 \cdots F_i}$ is used as ψ.) □

Lemma 5.3 *Let a be a term, s be a canonical environment term, and E, F be declarations.*
1. $\overline{[s]}(a \uparrow^{\mathrm{TY}(s)}) \equiv a$.
2. $a \uparrow^{E} \downarrow_{E} \equiv a$, $a \updownarrow^{E}_{F} \updownarrow^{F}_{E} \equiv a$.
3. $a \uparrow^{E} \uparrow^{F} \updownarrow^{E}_{F} \equiv a \uparrow^{F} \uparrow^{E}$.

Proof. We can prove these identities similarly to Lemma 5.2. We prove by induction on a, but we need to generalize each function ϕ to $\phi_{\overline{F}}$ by parameterizing with a sequence of declarations $\overline{F}$, so that the function can handle bound variables correctly when it is pushed into the scope in the induction step. In the base case of the induction, we apply the case analysis of variables given in the proof of Lemma 5.2. As for 1, we can prove by induction on a that $[s]_{\overline{F}}({\Uparrow^{\mathrm{TY}(s)}}_{\overline{F}}(a)) \equiv a$ holds for any sequence of declarations $\overline{F}$. □

Lemma 5.4 *Let s be a canonical environment term. Then, for any term a and declaration E, $a\updownarrow_E^{\mathrm{TY}(s)}[s\uparrow^E] \equiv \overline{[s]_E}(a)$ holds.*

Proof. Put $X := \mathrm{TY}(s)$. We prove, by induction on the construction of a, that the identity

$$((\Updownarrow^X_{E_n} \circ \Updownarrow^X_{E_{n-1}E_n} \circ \cdots \circ \Updownarrow^X_{E_0E_1\cdots E_n})(a))[s\uparrow^{E_0}\uparrow^{E_1}\cdots\uparrow^{E_n}] \equiv [s]_{E_0E_1\cdots E_n}(a) \quad (\star)$$

holds for any declarations E_0, E_1, …, E_n.

1. *a is a variable*: We put $a_0 := a$, $a_{i+1} := \Updownarrow^X_{E_iE_{i+1}\cdots E_n}(a_i)$, $y_{n+1} := a$, and $y_i := \Downarrow_{E_i} y_{i+1}$. (If $y_{i+1} \in E_i$ for some i, y_j $(j \leq i)$ is undefined.)

   (a) *$y_{i+1} \in E_i$ for some i*: We have $a_j \equiv a$ $(j \leq i)$ by Lemma 5.1 clause 1 and $\Updownarrow^X_{E_i} y_{i+1} \equiv \Uparrow^X y_{i+1}$ by assumption. Therefore, we have $a_{i+1} \equiv \Uparrow^{E_n}\cdots\Uparrow^{E_{i+1}}\Uparrow^X y_{i+1}$. By Lemma 5.1 clause 2,3, we have

   $$\begin{aligned} a_{i+2} &\equiv \Updownarrow^X_{E_{i+1}E_{i+2}\cdots E_n}(\Uparrow^{E_n}\cdots\Uparrow^{E_{i+1}}\Uparrow^X y_{i+1}) \\ &\equiv \Uparrow^{E_n}\cdots\Uparrow^{E_{i+2}}\Updownarrow^X_{E_{i+1}}\Uparrow^{E_{i+1}}\Uparrow^X y_{i+1} \\ &\equiv \Uparrow^{E_n}\cdots\Uparrow^{E_{i+2}}\Uparrow^X\Uparrow^{E_{i+1}} y_{i+1} \\ &\equiv \Uparrow^{E_n}\cdots\Uparrow^{E_{i+2}}\Uparrow^X y_{i+2} \end{aligned}$$

   We can proceed similarly and finally we have $a_{n+1} \equiv \Uparrow^X y_{n+1}$. Then,

   $$\begin{aligned} &\text{LHS of } (\star) \equiv \Downarrow_X\Uparrow^X y_{n+1} \equiv y_{n+1}, \\ &\text{RHS of } (\star) \equiv a \end{aligned}$$

   by Lemma 5.1 clause 1. Therefore, $(\star)$ holds.

   (b) *$y_1 \notin E_0$ and $y_1 \in X\Uparrow^{E_0}$*: We have $\Updownarrow^X_{E_0} y_1 \equiv \Downarrow_{E_0} y_1 \equiv y_0 \in X$ by assumption. Therefore, we have $a_1 \equiv \Uparrow^{E_n}\cdots\Uparrow^{E_1} y_0$. Since $\Uparrow^{E_1} y_0 \notin E_1$ and $\Uparrow^{E_1} y_0 \in X\Uparrow^{E_1}$, we have

   $$\begin{aligned} a_2 &\equiv \Uparrow^{E_n}\cdots\Uparrow^{E_2}\Updownarrow^X_{E_1}\Uparrow^{E_1} y_0 \\ &\equiv \Uparrow^{E_n}\cdots\Uparrow^{E_2}\Downarrow_{E_1}\Uparrow^{E_1} y_0 \\ &\equiv \Uparrow^{E_n}\cdots\Uparrow^{E_2} y_0 \end{aligned}$$

We can proceed similarly and finally we have $a_{n+1} \equiv y_0$. Then,

$$\begin{aligned} &\text{LHS of } (\star) \equiv y_0[s{\uparrow}^{E_0}{\uparrow}^{E_1}\cdots{\uparrow}^{E_n}], \\ &\text{RHS of } (\star) \equiv [s](y_0){\uparrow}^{E_0}{\uparrow}^{E_1}\cdots{\uparrow}^{E_n}. \end{aligned}$$

Therefore, $(\star)$ holds.

(c) $y_1 \notin E_0$ *and* $y_1 \notin X{\Uparrow}^{E_0}$: We have ${\Updownarrow}^X_{E_0} y_1 \equiv {\Uparrow}^X{\Uparrow}^{E_0}{\Downarrow}_X{\Downarrow}_{E_0} y_1$ by assumption. Therefore, we have $a_1 \equiv {\Uparrow}^{E_n}\cdots{\Uparrow}^{E_1}{\Uparrow}^X{\Uparrow}^{E_0}{\Downarrow}_X y_0$. By Lemma 5.1 clause 2,3, we have

$$\begin{aligned} a_2 &\equiv {\Uparrow}^{E_n}\cdots{\Uparrow}^{E_2}{\Updownarrow}^X_{E_1}{\Uparrow}^{E_1}{\Uparrow}^X{\Uparrow}^{E_0}{\Downarrow}_X y_0 \\ &\equiv {\Uparrow}^{E_n}\cdots{\Uparrow}^{E_2}{\Uparrow}^X{\Uparrow}^{E_1}{\Uparrow}^{E_0}{\Downarrow}_X y_0 \end{aligned}$$

We can proceed similarly and finally we have

$$a_{n+1} \equiv {\Uparrow}^X{\Uparrow}^{E_n}\cdots{\Uparrow}^{E_2}{\Uparrow}^{E_1}{\Uparrow}^{E_0}{\Downarrow}_X y_0.$$

Put $y' := {\Uparrow}^{E_n}\cdots{\Uparrow}^{E_2}{\Uparrow}^{E_1}{\Uparrow}^{E_0}{\Downarrow}_X y_0$. Then,

$$\begin{aligned} &\text{LHS of } (\star) \equiv {\Downarrow}_X{\Uparrow}^X y' \equiv y', \\ &\text{RHS of } (\star) \equiv ({\Downarrow}_X y_0){\uparrow}^{E_0}{\uparrow}^{E_1}\cdots{\uparrow}^{E_n} \equiv y'. \end{aligned}$$

Therefore, $(\star)$ holds.

2. $a \equiv e[\![b]\!]$: Put $E := \mathrm{TY}(e)$, $\phi := {\Updownarrow}^X_{E_n} \circ {\Updownarrow}^X_{E_{n-1}E_n} \circ \cdots \circ {\Updownarrow}^X_{E_0E_1\cdots E_n}$, $\phi' := {\Updownarrow}^X_{E_nE} \circ {\Updownarrow}^X_{E_{n-1}E_nE} \circ \cdots \circ {\Updownarrow}^X_{E_0E_1\cdots E_nE}$, and $t := s{\uparrow}^{E_0}{\uparrow}^{E_1}\cdots{\uparrow}^{E_n}$. Then,

$$\begin{aligned} &\text{LHS of } (\star) \equiv \phi(e)[\![\phi'(b)]\!][t] \equiv \phi(e)[t][\![\phi'(b){\updownarrow}^X_E[t{\uparrow}^E]]\!], \\ &\text{RHS of } (\star) \equiv [s]_{E_0E_1\cdots E_n}(e)[\![[s]_{E_0E_1\cdots E_nE}(b)]\!]. \end{aligned}$$

Therefore, by the induction hypothesis, we have the identity.

3. *Otherwise*: Similar to the case 2 or trivial. □

Lemma 5.5 *Let s be a canonical environment term. Then, for any term a, $a[s] \equiv \overline{[s]}(a)$ holds.*

Proof. By induction on the construction of a, using Lemma 5.4 in the case a is of the form $\lambda x.\ b$, $\kappa E.\ b$, or $e[\![b]\!]$. □

Lemma 5.6 corresponds to the Substitution Lemma [2] in the λ-calculus, that is, $M[x := K][y := L] \equiv M[y := L][x := K[y := L]]$ if $x \not\equiv y$ and $x \notin \mathrm{FV}(L)$.

Lemma 5.6 (Substitution Lemma) *Let s and t be canonical environment terms. Then, for any term a, $a[s][t] \equiv a \updownarrow^{\mathrm{TY}(t)}_{\mathrm{TY}(s)} [t \uparrow^{\mathrm{TY}(s)}][s[t]]$ holds.*

Note that the effect of exchanging the order of two substitutions s and t is adjusted by applying the exchange operation $\updownarrow^{\mathrm{TY}(t)}_{\mathrm{TY}(s)}$ to a and the push operation $\uparrow^{\mathrm{TY}(s)}$ to t. For example, let a be $(x\ \sharp x)$, s be $\{z/x\}$, and t be $\{\sharp^3 x/x,\ (x\ \sharp x)/\sharp x\}$ in the lemma. Then, we have

$$\begin{aligned}
(x\ \sharp^2 x)[s][t] &\equiv (z\ \sharp x)[t] \\
&\equiv z\ (x\ \sharp x), \\
(x\ \sharp^2 x) \updownarrow^{\{x,\sharp x\}}_{\{x\}} [t \uparrow^{\{x\}}][s[t]] &\equiv (\sharp^2 x\ \sharp x)[\{\sharp^4 x/x,\ (\sharp x\ \sharp^2 x)/\sharp x\}][s[t]] \\
&\equiv (\Downarrow_{\{x,\sharp x\}} \sharp^2 x\ (\sharp x\ \sharp^2 x))[s[t]] \\
&\equiv (x\ (\sharp x\ \sharp^2 x))[\{z/x\}] \\
&\equiv z\ (\Downarrow_{\{x\}} (\sharp x)\ \Downarrow_{\{x\}} (\sharp^2 x)) \\
&\equiv z\ (x\ \sharp x).
\end{aligned}$$

(See also the example below the definition of the substitution in section 3.)

Proof of Lemma 5.6. Put $X := \mathrm{TY}(s)$ and $Y := \mathrm{TY}(t)$. We prove, by induction on the construction of a, that the identity

$$[t]_{\overline{E}}([s]_{\overline{E}}(a)) \equiv [[t](s)]_{\overline{E}}([t \uparrow^X]_{\overline{E}}(\Updownarrow^Y_{X\,\overline{E}}(a))) \qquad (\star)$$

holds for any sequence of declarations $\overline{E} \equiv E_1 \cdots E_n$. The lemma follows from the case $n = 0$ of this identity and Lemma 5.5.

1. *a is a variable*: If $\Downarrow_{E_{i+1}} \cdots \Downarrow_{E_n} a \in E_i$ for some i, both sides of $(\star)$ are a by Lemma 5.1 clause 1. Otherwise, we put $y := \Downarrow_{E_1} \cdots \Downarrow_{E_n} a$. In this case, we can prove $(\star)$ by case analysis of y.

    (a) $y \in X$: We have

    $$\text{LHS of } (\star) \equiv [t]_{\overline{E}}([s](y) \uparrow^{E_1} \cdots \uparrow^{E_n}) \equiv [t]([s](y)) \uparrow^{E_1} \cdots \uparrow^{E_n}$$

by Lemma 5.2. On the other hand, since $\Updownarrow^Y_X y \equiv \Uparrow^Y y$ in this case and $[t\uparrow^X](\Uparrow^Y y) \equiv y$ by Lemma 5.3 clause 1, we have

$$[t\uparrow^X]_{\overline{E}}(\Updownarrow^Y_{X\overline{E}}(a)) \equiv [t\uparrow^X](\Uparrow^Y y)\uparrow^{E_1}\cdots\uparrow^{E_n} \equiv y\uparrow^{E_1}\cdots\uparrow^{E_n},$$

so, by Lemma 5.2,

$$\text{RHS of } (\star) \equiv [[t](s)]_{\overline{E}}(y\uparrow^{E_1}\cdots\uparrow^{E_n}) \equiv [[t](s)](y)\uparrow^{E_1}\cdots\uparrow^{E_n}.$$

Let $s \equiv \{s_1/x_1, \ldots, s_m/x_m\}$, then $y \equiv x_i$ for some i. Therefore,

$$\begin{aligned}&[t]([s](y)) \equiv [t](s_i),\\ &[[t](s)](y) \equiv [\{[t](s_1)/x_1, \ldots, [t](s_m)/x_m\}](y) \equiv [t](s_i)\end{aligned}$$

Therefore, $(\star)$ holds.

(b) $y \in Y\Uparrow^X$: We have

$$\text{LHS of } (\star) \equiv [t]_{\overline{E}}([s](y)\uparrow^{E_1}\cdots\uparrow^{E_n}) \equiv [t](\Downarrow_X y)\uparrow^{E_1}\cdots\uparrow^{E_n}$$

by Lemma 5.2 and Lemma 5.3 clause 1. On the other hand, since $\Updownarrow^Y_X y \equiv \Downarrow_X y$, we have

$$[t\uparrow^X]_{\overline{E}}(\Updownarrow^Y_{X\overline{E}}(a)) \equiv [t\uparrow^X](\Downarrow_X y)\uparrow^{E_1}\cdots\uparrow^{E_n},$$

so, by Lemma 5.2,

$$\begin{aligned}\text{RHS of } (\star) &\equiv [[t](s)]_{\overline{E}}([t\uparrow^X](\Downarrow_X y)\uparrow^{E_1}\cdots\uparrow^{E_n})\\ &\equiv [[t](s)]([t\uparrow^X](\Downarrow_X y))\uparrow^{E_1}\cdots\uparrow^{E_n}.\end{aligned}$$

Let $t \equiv \{t_1/x_1, \ldots, t_m/x_m\}$, then $\Downarrow_X y \equiv x_i$ for some i. Therefore,

$$\begin{aligned}&[t](\Downarrow_X y) \equiv t_i,\\ &[[t](s)]([t\uparrow^X](\Downarrow_X y)) \equiv [[t](s)](t_i\uparrow^X) \equiv t_i\uparrow^X\downarrow_X \equiv t_i.\end{aligned}$$

Therefore, $(\star)$ holds.

(c) *Otherwise*: We have

$$\text{LHS of } (\star) \equiv [t]_{\overline{E}}([s](y)\uparrow^{E_1}\cdots\uparrow^{E_n}) \equiv (\Downarrow_Y\Downarrow_X y)\uparrow^{E_1}\cdots\uparrow^{E_n}.$$

On the other hand, since $\Updownarrow^Y_X y \equiv \Uparrow^Y\Uparrow^X\Downarrow_Y\Downarrow_X y$,

$$\begin{aligned}\text{RHS of } (\star) &\equiv (\Downarrow_X\Downarrow_Y\Uparrow^Y\Uparrow^X\Downarrow_Y\Downarrow_X y)\uparrow^{E_1}\cdots\uparrow^{E_n}\\ &\equiv (\Downarrow_Y\Downarrow_X y)\uparrow^{E_1}\cdots\uparrow^{E_n}.\end{aligned}$$

2. $a \equiv e[\![b]\!]$: Put $\sigma_1 := [[t](s)]_{\overline{E}\,\mathrm{TY}(e)}$, $\sigma_2 := [t\uparrow^X]_{\overline{E}\,\mathrm{TY}(e)}$, and $\phi := \Updownarrow^Y_{X\overline{E}\,\mathrm{TY}(e)}$. Then, we have

$$\begin{aligned}\text{LHS of } (\star) &\equiv [t]_{\overline{E}}([s]_{\overline{E}}(e))[\![[t]_{\overline{E}\,\mathrm{TY}(e)}([s]_{\overline{E}\,\mathrm{TY}(e)}(b))]\!],\\ \text{RHS of } (\star) &\equiv [[t](s)]_{\overline{E}}([t\uparrow^X]_{\overline{E}}(\Updownarrow^Y_{X\overline{E}}(e)))[\![\sigma_1(\sigma_2(\phi(b)))]\!].\end{aligned}$$

   Therefore, by the induction hypothesis, we have the identity.

3. *Otherwise*: Similar to the case 2 or trivial. □

Lemma 5.7 *Let $\phi : \mathsf{V} \to \mathsf{T}$ be a (possibly) partial function, a be a term, and E, F be declarations. Then, $\overline{\phi_{FE}}(a \updownarrow^E_F) \equiv \overline{\phi_{EF}}(a) \updownarrow^E_F$ holds.*

Proof. We can prove, by induction on a, that the identity

$$\phi_{FE\overline{E}}(\Updownarrow^E_{F\overline{E}}(a)) \equiv \Updownarrow^E_{F\overline{E}}(\phi_{EF\overline{E}}(a))$$

holds for any sequence of declarations $\overline{E}$. In the case a is a variable, we apply the case analysis similar to the one in the proof of Lemma 5.6 and use Lemma 5.3 clause 3. □

The reduction is compatible with substitution.

Theorem 5.8 *If $a \xrightarrow{*}_{\lambda\kappa\varepsilon} b$, then $a[s] \xrightarrow{*}_{\lambda\kappa\varepsilon} b[s]$.*

Proof. By Lemma 5.5 and Lemma 5.7. □

As we have studied in [18], we can internalize the meta-level operation of substitution by means of evaluation terms which are of the form $e[\![a]\!]$. We can show that the meta-level substitution and the internalized substitution coincide, that is, $a[s] =_\varepsilon s[\![a]\!]$ holds.

Theorem 5.9 *Let s be a canonical environment term. Then, for any term a, $a[s] =_\varepsilon s[\![a]\!]$ holds.*

Proof. We define the size $|a|$ of a term a as follows.

1. $|x| := 1$
2. $|\lambda x.\, b| := |b| + 1$
3. $|ba| := |b| + |a|$

4. $|\kappa E.\ a| := |a| + 1$
5. $|a{\bullet}e| := |a| + |e|$
6. $|\{a_1/x_1, \dots, a_n/x_n\}| := |a_1| + \cdots + |a_n| + 1$
7. $|e\llbracket a\rrbracket| := |e| + |a|$

For terms e_1, …, e_n, a, we prove the identity

$$(e_1\llbracket \cdots e_n\llbracket a\rrbracket \cdots\rrbracket)[s] =_\varepsilon s\llbracket e_1\llbracket \cdots e_n\llbracket a\rrbracket \cdots\rrbracket\rrbracket \qquad (\star)$$

by induction on the lexicographic order $(|e_1\llbracket \cdots e_n\llbracket a\rrbracket \cdots\rrbracket|, |a|)$.

1. *a is a variable*: We put $X := \mathrm{TY}(s)$, $E_i := \mathrm{TY}(e_i)$, $y_{n+1} := a$, and $y_i := \Downarrow_{E_i}(y_{i+1})$.

   (a) *$y_{i+1} \in E_i$ for some i*: We put $f := e_1\llbracket \cdots e_{i-1}\llbracket e_i\rrbracket \cdots\rrbracket$. Since

   $$e_1\llbracket \cdots e_n\llbracket a\rrbracket \cdots\rrbracket \overset{*}{\longrightarrow}_\varepsilon e_1\llbracket \cdots e_i\llbracket y_{i+1}\rrbracket \cdots\rrbracket \overset{*}{\longrightarrow}_\varepsilon f\llbracket y_{i+1}\rrbracket$$

   holds, we have

   $$\begin{aligned}\text{LHS of } (\star) &\overset{*}{\longrightarrow}_\varepsilon (f\llbracket y_{i+1}\rrbracket)[s] \equiv f[s]\llbracket y_{i+1} \updownarrow^X_{E_i}[s\uparrow^{E_i}]\rrbracket,\\ \text{RHS of } (\star) &\overset{*}{\longrightarrow}_\varepsilon s\llbracket f\llbracket y_{i+1}\rrbracket\rrbracket \longrightarrow_\varepsilon s\llbracket f\rrbracket\llbracket y_{i+1}\rrbracket.\end{aligned}$$

   Since we have

   $$y_{i+1} \updownarrow^X_{E_i}[s\uparrow^{E_i}] \equiv \Uparrow^X(y_{i+1})[s\uparrow^{E_i}] \equiv \Downarrow_X(\Uparrow^X(y_{i+1})) \equiv y_{i+1},$$

   the identity $(\star)$ holds by the induction hypothesis for e_1, …, e_{i-1}, e_i.

   (b) *Otherwise*: Since $e_1\llbracket \cdots e_n\llbracket a\rrbracket \cdots\rrbracket \overset{*}{\longrightarrow}_\varepsilon y_1$, the identity $(\star)$ holds from $s\llbracket y_1\rrbracket \longrightarrow_\varepsilon y_1[s]$.

2. $a \equiv \lambda y.\ b$: By successively using (fun), we have

   $$e_1\llbracket \cdots e_n\llbracket \lambda y.\ b\rrbracket \cdots\rrbracket \overset{*}{\longrightarrow}_\varepsilon \lambda y.\ e'_1\llbracket \cdots e'_n\llbracket b'\rrbracket \cdots\rrbracket$$

   where e'_1, …, e'_n, b' are obtained by applying push/exchange operations to e_1, …, e_n, b respectively. By Theorem 5.8 and the definition of substitution,

   $$\text{LHS of } (\star) \overset{*}{\longrightarrow}_\varepsilon \lambda y.\ (e''_1\llbracket \cdots e''_n\llbracket b''\rrbracket \cdots\rrbracket[s'])$$

where e''_1, …, e''_n, b'', s' are obtained by applying push/exchange operations to e'_1, …, e'_n, b', s respectively. On the other hand, we have

$$\text{RHS of } (\star) \xrightarrow{*}_{\varepsilon} s[\![\lambda y.\ e'_1[\![\cdots e'_n[\![b']\!]\cdots]\!]]\!] \rightarrow_{\varepsilon} \lambda y.\ s'[\![e''_1[\![\cdots e''_n[\![b'']\!]\cdots]\!]]\!].$$

Since the size of a term does not change by push/exchange operations, we can apply the induction hypothesis.

3. $a \equiv e[\![b]\!]$: $(e_1[\![\cdots e_n[\![e[\![b]\!]]\!]\cdots]\!])[s] =_{\varepsilon} s[\![e_1[\![\cdots e_n[\![e[\![b]\!]]\!]\cdots]\!]]\!]$ by the induction hypothesis for e_1, …, e_n, e, b.

4. *Otherwise*: Similar to the case 2 or trivial. □

We can generalize the internalized version of Substitution Lemma (Lemma 5.6).

Lemma 5.10 *Let e and f be environment terms. Then, for any term a, $e[\![f[\![a]\!]]\!] =_{\varepsilon} e[\![f]\!][\![(e{\uparrow}^{\mathrm{TY}(f)})[\![a{\updownarrow}^{\mathrm{TY}(e)}_{\mathrm{TY}(f)}]\!]]\!]$ holds.*

Proof. (Sketch) We can prove the following identities:

$$\Uparrow^{X}{}_{E}(a) \equiv \Updownarrow^{E}_{X}(\Uparrow^{X}(a)), \quad \Updownarrow^{F}_{X\,E}(\Updownarrow^{E}_{X}(\Updownarrow^{E}_{F\,X}(a))) \equiv \Updownarrow^{E}_{F}(\Updownarrow^{E}_{X\,F}(\Updownarrow^{F}_{X}(a))),$$

where a is a term and E, F, X are declarations. Using these identities, we can prove this lemma by induction of the size of a. □

In the following, we prove several important theorems. Especially, Theorem 5.14, 5.24 and 5.26 will establish the purity of our calculus.

Theorem 5.11 (Closed Normal Term is Canonical) *If $\vdash c : C$ and c is normal, then c is canonical.*

Proof. We omit the proof, because this theorem can be proved in the same way as in [18]. □

Lemma 5.12 *If $\Gamma \vdash a : A$, then $\Gamma{\Uparrow}^{E} \vdash a{\uparrow}^{E} : A$ and $\Gamma{\Updownarrow}^{E}_{F} \vdash a{\updownarrow}^{E}_{F} : A$ for any declarations E, F.*

Proof. (Sketch) In this proof, we use the notation $\phi(\Gamma) := \{\phi(x) \mid x \in \Gamma\}$ for $\phi : \mathsf{V} \to \mathsf{V}$. We can prove that if $\Gamma \vdash a : A$, then

$$\Uparrow^E_{\overline{E}}(\Gamma) \vdash \Uparrow^E_{\overline{E}}(a) : A, \qquad \Updownarrow^E_{F\overline{E}}(\Gamma) \vdash \Updownarrow^E_{F\overline{E}}(a) : A$$

for any sequence of declarations $\overline{E}$, by the induction on the derivation $\Gamma \vdash a : A$. □

Theorem 5.13 (Subject Reduction) *If $\Gamma \vdash a : A$ and $a \to_{\lambda\kappa\varepsilon} b$, then $\Delta \vdash b : A$ for some $\Delta \subseteq \Gamma$.*

Proof. The theorem is proved by the induction on the derivation $\Gamma \vdash a : A$ and the reduction $a \to_{\lambda\kappa\varepsilon} b$. Here we verify only a few key cases.

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash a : A\end{array}}{(\Gamma - E)\Downarrow_E \vdash \kappa E.\ a : A^E} \quad \begin{array}{c}\vdots\\ \Delta \vdash e : E\end{array}}{(\Gamma - E)\Downarrow_E \cup \Delta \vdash (\kappa E.\ a)\bullet e : A} \qquad \mapsto_\kappa$$

$$\dfrac{\begin{array}{c}\vdots\\ \Delta \vdash e : E\end{array} \quad \begin{array}{c}\vdots\\ \Gamma \vdash a : A\end{array}}{\Delta \cup (\Gamma - E)\Downarrow_E \vdash e[\![a]\!] : A.}$$

$$\dfrac{\begin{array}{c}\vdots\\ \Gamma \vdash e : F\end{array} \quad \dfrac{\begin{array}{c}\vdots\\ \Delta \vdash a : A\end{array}}{(\Delta - E)\Downarrow_E \vdash \kappa E.\ a : A^E}}{\Gamma \cup ((\Delta - E)\Downarrow_E - F)\Downarrow_F \vdash e[\![\kappa E.\ a]\!] : A^E} \qquad \mapsto_\varepsilon$$

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma\Uparrow^E \vdash e\uparrow^E : F\end{array} \quad \begin{array}{c}\vdots\\ \Delta\Updownarrow^F_E \vdash a\updownarrow^F_E : A\end{array}}{\Gamma\Uparrow^E \cup (\Delta\Updownarrow^F_E - F)\Downarrow_F \vdash (e\uparrow^E)[\![a\updownarrow^F_E]\!] : A}}{((\Gamma\Uparrow^E \cup (\Delta\Updownarrow^F_E - F)\Downarrow_F) - E)\Downarrow_E \vdash \kappa E.\ (e\uparrow^E)[\![a\updownarrow^F_E]\!] : A^E.}$$

We have $\Gamma \cup ((\Delta - E)\Downarrow_E - F)\Downarrow_F \equiv ((\Gamma\Uparrow^E \cup (\Delta\Updownarrow^F_E - F)\Downarrow_F) - E)\Downarrow_E$ from Lemma 2.3. By Lemma 5.12, we are done. □

Theorem 5.14 (Confluence) *$\to_{\lambda\kappa\varepsilon}$ on $\lambda\kappa\varepsilon$-terms is confluent.*

Proof. The proof is a straightforward extension of that for $\lambda\varepsilon$ [18], and we omit the details here. □

We remark that from the confluence of $\lambda\kappa\varepsilon$, we see that the operations of hole filling and β-reduction always commute, since in $\lambda\kappa\varepsilon$, hole filling is computed by reducing a term of the form $(\lambda X.\ a)(\kappa E.\ b)$.

We next prove that $\lambda\kappa\varepsilon$ is a conservative extension of $\lambda\beta$ and $\lambda\varepsilon$ [18]. We first prove that α-equivalence is preserved by reduction.

Lemma 5.15 *Let* * *be* $^{*\lambda}$ *or* $^{*\varepsilon}$*, a be a $\lambda\kappa\varepsilon$-term, and E, F be declarations. We have the following identities.*

$$\mathrm{FV}(a^*) = \mathrm{FV}(a), \quad \mathrm{TY}(a^*) = \mathrm{TY}(a),$$

$$(a\uparrow^{E})^* \equiv a^*\uparrow^{E}, \quad (a\downarrow_{E})^* \equiv a^*\downarrow_{E}, \quad (a\updownarrow^{E}_{F})^* \equiv a^*\updownarrow^{E}_{F}.$$

Proof. $\mathrm{FV}(a^*) = \mathrm{FV}(a)$ can be proved by induction on a. $\mathrm{TY}(a^*) = \mathrm{TY}(a)$ can be proved easily. For the rest of identities, we give a sketch of the proof of $(a\uparrow^{E})^* \equiv a^*\uparrow^{E}$, because other ones can be proved similarly.

Put $X := \{x_1, \ldots, x_n\}$ and $I := \{\iota_1, \ldots, \iota_n\}$. We can prove the identity:

$$(\Uparrow^{E}{}_{X}(a))[\{\iota_1/x_1, \ldots, \iota_n/x_n\}] \equiv \Uparrow^{E}{}_{I}(a[\{\iota_1/x_1, \ldots, \iota_n/x_n\}]).$$

Using this identity, we can prove that $(\Uparrow^{E}{}_{\overline{F}}(a))^* \equiv \Uparrow^{E}{}_{\overline{F}}(a^*)$ holds for any sequence of declarations $\overline{F}$ by induction on a. □

Lemma 5.16 *Let E and F be declarations such that $\{\natural(x) \mid x \in E\} \cap \{\natural(x) \mid x \in F\} = \emptyset$. Then, we have $a\updownarrow^{E}_{F} \equiv a$.*

Proof. We can prove that $\Updownarrow^{E}_{F\overline{F}}(a) \equiv a$ holds for any sequence of declarations $\overline{F}$ by induction on a and Lemma 2.2. □

Theorem 5.17 *If $a \equiv_\alpha a'$, $a \to_{\lambda\kappa\varepsilon} b$, $a' \to_{\lambda\kappa\varepsilon} b'$, and the position of the redex of $a \to_{\lambda\kappa\varepsilon} b$ and that of the redex of $a' \to_{\lambda\kappa\varepsilon} b'$ are the same, then $b \equiv_\alpha b'$.*

Proof. This theorem is proved by induction on $a \to_{\lambda\kappa\varepsilon} b$. We prove only the base case $e[\![\lambda x.\ b]\!] \to_{\lambda\kappa\varepsilon} \lambda x.\ (e\uparrow^{\{x\}})[\![b\updownarrow^{\mathrm{TY}(e)}_{\{x\}}]\!]$. Other cases are similar or trivial. In this case, we have a term $e'[\![\lambda x'.\ b']\!]$ such that $e[\![\lambda x.\ b]\!] \equiv_\alpha e'[\![\lambda x'.\ b']\!]$ and $e'[\![\lambda x'.\ b']\!] \to_{\lambda\kappa\varepsilon} \lambda x'.\ (e'\uparrow^{\{x'\}})[\![b'\updownarrow^{\mathrm{TY}(e')}_{\{x'\}}]\!]$. So, we will prove

$$\lambda x.\ (e\uparrow^{\{x\}})[\![b\updownarrow^{\mathrm{TY}(e)}_{\{x\}}]\!] \equiv_\alpha \lambda x'.\ (e'\uparrow^{\{x'\}})[\![b'\updownarrow^{\mathrm{TY}(e')}_{\{x'\}}]\!] \qquad (\star)$$

in the following. We have two cases according to the form of e. (In the following, * stands for $^{*\varepsilon}$ and ι for ι_1.)

1. e is not a canonical environment term: In this case, we have $e^* \equiv e'^*$ and $b^*[\{\iota/x\}] \equiv b'^*[\{\iota/x'\}]$, since $e[\![\lambda x.\ b]\!] \equiv_\alpha e'[\![\lambda x'.\ b']\!]$. On the other hand, we have

$$\begin{aligned}
&(\lambda x.\ (e\uparrow^{\{x\}})[\![b\updownarrow^{\mathrm{TY}(e)}_{\{x\}}]\!])^* \\
&\equiv \lambda\iota.\ ((e\uparrow^{\{x\}})[\![b\updownarrow^{\mathrm{TY}(e)}_{\{x\}}]\!])^*[\{\iota/x\}] \\
&\equiv \lambda\iota.\ (e\uparrow^{\{x\}})^*[\{\iota/x\}][\![((b\updownarrow^{\mathrm{TY}(e)}_{\{x\}})^*\updownarrow^{\{x\}}_{\mathrm{TY}(e)})[\{\iota/x\}]]\!] \\
&\equiv \lambda\iota.\ e^*[\![b^*[\{\iota/x\}]]\!]
\end{aligned}$$

by Lemma 5.15 and Lemma 5.3 clause 1,2. We also have

$$(\lambda x'.\ (e'\uparrow^{\{x'\}})[\![b'\updownarrow^{\mathrm{TY}(e')}_{\{x'\}}]\!])^* \equiv \lambda\iota.\ e'^*[\![b'^*[\{\iota/x'\}]]\!]$$

by the same calculation. Therefore, we have ($\star$).

2. $e \equiv \{a_1/y_1, \ldots, a_n/y_n\}$: By putting $\{\overline{\iota}/\overline{y}\} := \{\iota_1/y_{p(1)}, \ldots, \iota_n/y_{p(n)}\}$ where $y_{p(1)} \prec \cdots \prec y_{p(n)}$, we have

$$(e[\![\lambda x.\ b]\!])^* \equiv \{a_{p(1)}{}^*/\iota_1, \ldots, a_{p(n)}{}^*/\iota_n\}[\![(\lambda x.\ b)^*[\{\overline{\iota}/\overline{y}\}]]\!].$$

We also put $Y := \{y_1, \ldots, y_n\}$, $I := \{\iota_1, \ldots, \iota_n\}$, and $\overline{\iota}' := \sharp\iota_1, \iota_2, \ldots, \iota_n$. Then, we have

$$\begin{aligned}
&(\lambda x.\ b)^*[\{\overline{\iota}/\overline{y}\}] \equiv (\lambda\iota.\ b^*[\{\iota/x\}])[\{\overline{\iota}/\overline{y}\}] \\
&\equiv \lambda\iota.\ (b^*[\{\iota/x\}])\updownarrow^{Y}_{\{\iota\}}[\{\overline{\iota}/\overline{y}\}\uparrow^{\{\iota\}}] \equiv \lambda\iota.\ b^*[\{\iota/x\}][\{\overline{\iota}'/\overline{y}\}]
\end{aligned}$$

by Lemma 5.16 for Y and $\{\iota\}$. Since $e[\![\lambda x.\ b]\!] \equiv_\alpha e'[\![\lambda x'.\ b']\!]$, e' should be an environment term such that $e' \equiv \{a'_1/y'_1, \ldots, a'_n/y'_n\}$ and $a_{p(i)}{}^* \equiv a'_{q(i)}{}^*$ where $y'_{q(1)} \prec \cdots \prec y'_{q(n)}$. Putting $\{\overline{\iota}/\overline{y'}\} := \{\iota_1/y'_{q(1)}, \ldots, \iota_n/y'_{q(n)}\}$, we have

$$b^*[\{\iota/x\}][\{\overline{\iota}'/\overline{y}\}] \equiv b'^*[\{\iota/x'\}][\{\overline{\iota}'/\overline{y'}\}].$$

On the other hand, we have

$$\begin{aligned}
&(\lambda x.\ (e\uparrow^{\{x\}})[\![b\updownarrow^{\mathrm{TY}(e)}_{\{x\}}]\!])^* \\
&\equiv \lambda\iota.\ ((e\uparrow^{\{x\}})[\![b\updownarrow^{Y}_{\{x\}}]\!])^*[\{\iota/x\}] \\
&\equiv \lambda\iota.\ \{c_{p(1)}/\iota_1, \ldots, c_{p(n)}/\iota_n\}[\![(b\updownarrow^{Y}_{\{x\}})^*[\{\overline{\iota}/\overline{y}\}]]\!][\{\iota/x\}] \\
&\equiv \lambda\iota.\ \{d_{p(1)}/\iota_1, \ldots, d_{p(n)}/\iota_n\}[\![((b\updownarrow^{Y}_{\{x\}})^*[\{\overline{\iota}/\overline{y}\}])\updownarrow^{\{x\}}_{I}[\{\iota/x\}]]\!]
\end{aligned}$$

where we put $c_i := (a_i \uparrow^{\{x\}})^*$ and $d_i := (a_i \uparrow^{\{x\}})^*[\{\iota/x\}]$. Then, we have

$$\begin{aligned}&((b \updownarrow^{Y}_{\{x\}})^*[\{\overline{\iota}/\overline{y}\}]) \updownarrow^{\{x\}}_{I} [\{\iota/x\}] \equiv (b \updownarrow^{Y}_{\{x\}})^*[\{\overline{\iota}/\overline{y}\}][\{\iota/x\}] \\ &\equiv (b \updownarrow^{Y}_{\{x\}})^* \updownarrow^{\{x\}}_{Y} [\{\iota \uparrow^{I}/x\}][\{\overline{\iota}/\overline{y}\}] \equiv b^*[\{\sharp\iota/x\}][\{\overline{\iota}/\overline{y}\}]\end{aligned}$$

and $d_i \equiv a_i^*$ by Lemma 5.16 for $\{x\}$ and I, Lemma 5.15 and Lemma 5.3 clause 1,2. By the same calculation, we have

$$\begin{aligned}&(\lambda x'.\ (e' \uparrow^{\{x'\}})[\![b' \updownarrow^{\mathrm{TY}(e')}_{\{x'\}}]\!])^* \\ &\equiv \lambda\iota.\ \{{a'_{q(1)}}^*/\iota_1, \ldots, {a'_{q(n)}}^*/\iota_n\}[\![b'^*[\{\sharp\iota/x'\}][\{\overline{\iota}/\overline{y'}\}]]\!].\end{aligned}$$

Therefore, we have $(\star)$. $\square$

Note that α^λ-equivalence is not preserved by reduction. For example, $(\lambda x.\ x)z$ and $(\lambda y.\ y)z$ are α^λ-equivalent, but $\{z/x\}[\![x]\!]$ and $\{z/y\}[\![y]\!]$ are not.

Now we can describe the conservativity of $\lambda\kappa\varepsilon$ over $\lambda\beta$. Let $\lambda\kappa\varepsilon/\alpha$ be the system obtained by identifying α-equivalent terms in $\lambda\kappa\varepsilon$. By Theorem 5.17, the reduction relation in $\lambda\kappa\varepsilon$ becomes the reduction relation in $\lambda\kappa\varepsilon/\alpha$.

Theorem 5.18 (Conservativity) *Let a and b be terms of $\lambda\beta$. Then, $a \overset{*}{\to} b$ in $\lambda\kappa\varepsilon/\alpha$ iff $a \overset{*}{\to} b$ in $\lambda\beta$.*

Proof. This theorem can be proved in the same way as in [18], which uses the translation from terms of $\lambda\varepsilon$ to terms of $\lambda\beta$. $\square$

To describe the conservativity theorem (Theorem 5.18), we have identified α-equivalent terms in $\lambda\kappa\varepsilon$, because α-equivalent terms are identified in $\lambda\beta$. But we can give an elaborated description by using the feature of $\lambda\kappa\varepsilon$ that α-conversion is not necessary to reduce terms. In the following, we will prove the conservativity of $\lambda\kappa\varepsilon$ over $\lambda\varepsilon$ and $\lambda\beta$ described in a different style (Theorem 5.22, 5.24). For this purpose, we embed the terms of $\lambda\varepsilon$ in the $\lambda\kappa\varepsilon$-terms. A *general $\lambda\varepsilon$-term* is a $\lambda\kappa\varepsilon$-term such that its typing derivation uses the (axiom), ($\Rightarrow I$), ($\Rightarrow E$), (env$I$), (env$E$) rules only, and in the derivation, x in ($\Rightarrow I$) is a pure variable, $x_1, \ldots, x_n$ in (env$I$) are pure variables, and E in (env$E$) is a set of pure variables. A term is *pure* if all the variables used in the (axiom) rule in its typing derivation are pure variables. A *$\lambda\varepsilon$-term* is a pure general $\lambda\varepsilon$-term. Next we represent the reduction rules of $\lambda\varepsilon$ in $\lambda\kappa\varepsilon$. The rules of $\lambda\varepsilon$ are (var), (funapp), (env), (eval) of $\lambda\kappa\varepsilon$ and the following two rules.

(gc$'$) $e[\![a]\!] \mapsto_\varepsilon a$, if $\mathrm{TY}(e) \cap \mathrm{FV}(a) = \emptyset$.

(fun$'$) $e[\![\lambda x.\ b]\!] \mapsto_\varepsilon \lambda x.\ e[\![b]\!]$, if $x \notin \mathrm{TY}(e) \cup \mathrm{FV}(e)$.

But unlike $\lambda\kappa\varepsilon$, α^λ-equivalent terms are identified in $\lambda\varepsilon$. Therefore, we represent the reduction rules as follows: Let a and b be $\lambda\varepsilon$-terms. We write $a \xrightarrow{\lambda\varepsilon} b$ if there exist $\lambda\varepsilon$-terms a' and b' such that $a \equiv^\lambda_\alpha a'$, $b \equiv^\lambda_\alpha b'$, and a' is reduced to b' by the above rules. We write $\xrightarrow{\lambda\varepsilon *}$ for the reflexive and transitive closure of $\xrightarrow{\lambda\varepsilon}$.

Lemma 5.19 *Let a be a $\lambda\varepsilon$-term and E, F be sets of pure variables.*
1. $E \cap \mathrm{FV}(a) = \emptyset$ iff $a{\uparrow}^E \equiv a$ iff $a{\uparrow}^E$ is a $\lambda\varepsilon$-term.
2. $E \cap \mathrm{FV}(a) = \emptyset$ iff $a{\downarrow}_E \equiv a$ iff $a{\downarrow}_E$ is a $\lambda\varepsilon$-term.
3. $E \cap F \cap \mathrm{FV}(a) = \emptyset$ iff $a{\updownarrow}^E_F \equiv a$ iff $a{\updownarrow}^E_F$ is a $\lambda\varepsilon$-term.

Proof. Since a is a $\lambda\varepsilon$-term, all the variables in a are of level 0. Therefore, we need to prove only the case a is a variable. But it is easily checked. □

Lemma 5.20 *1. Let $e[\![a]\!]$ be a $\lambda\varepsilon$-term. Then, $a{\downarrow}_{\mathrm{TY}(e)} \equiv a$ iff $\mathrm{TY}(e) \cap \mathrm{FV}(a) = \emptyset$.*
2. Let $e[\![\lambda x.\ b]\!]$ be a $\lambda\varepsilon$-term. Then, $\lambda x.\ (e{\uparrow}^{\{x\}})[\![b{\updownarrow}^{\mathrm{TY}(e)}_{\{x\}}]\!] \equiv \lambda x.\ e[\![b]\!]$ iff $x \notin \mathrm{TY}(e) \cup (\mathrm{FV}(e) \cap \mathrm{FV}(b))$.

Proof. By Lemma 5.19. □

By this lemma, we know that (gc) restricted to $\lambda\varepsilon$-terms is (gc$'$), but (fun) restricted to $\lambda\varepsilon$-terms requires a weaker condition than (fun$'$). However, this difference is not essential as we see in Lemma 5.21.

Lemma 5.21 *If a and b are general $\lambda\varepsilon$-terms, a' is a $\lambda\varepsilon$-term, $a \equiv_\alpha a'$, and $a \to_{\lambda\varepsilon} b$, then there is a $\lambda\varepsilon$-term a'' and b' such that $a' \equiv^\lambda_\alpha a''$, $b' \equiv_\alpha b$, and $a'' \to_{\lambda\varepsilon} b'$. Here, if $a \to_{\lambda\varepsilon} b$ by* (fun)*, then $a'' \to_{\lambda\varepsilon} b'$ by* (fun$'$)*, otherwise, $a'' \to_{\lambda\varepsilon} b'$ by the same rule as $a \to_{\lambda\varepsilon} b$.*

Proof. This theorem is proved by induction on $a \to_{\lambda\varepsilon} b$. We prove only the base case $e[\![\lambda x.\ b]\!] \to_{\lambda\varepsilon} \lambda x.\ (e{\uparrow}^{\{x\}})[\![b{\updownarrow}^{\mathrm{TY}(e)}_{\{x\}}]\!]$. Other cases are similar or trivial. In this case, we have a $\lambda\varepsilon$-term $e'[\![\lambda x'.\ b']\!]$ such that $e[\![\lambda x.\ b]\!] \equiv_\alpha e'[\![\lambda x'.\ b']\!]$. Then, we can find a $\lambda\varepsilon$-term $\lambda x''.\ b''$ such that $\lambda x''.\ b'' \equiv^\lambda_\alpha \lambda x'.\ b'$ and $x'' \notin \mathrm{TY}(e') \cup \mathrm{FV}(e')$. So, we will prove

$$\lambda x.\ (e{\uparrow}^{\{x\}})[\![b{\updownarrow}^{\mathrm{TY}(e)}_{\{x\}}]\!] \equiv_\alpha \lambda x''.\ e'[\![b'']\!] \qquad (\star)$$

in the following. We have two cases according to the form of e. (In the following, * stands for $^{*\varepsilon}$ and ι for ι_1.)

1. $e \not\equiv \{a_1/y_1, \ldots, a_n/y_n\}$: In this case, we have $e^* \equiv e'^*$ and $b^*[\{\iota/x\}] \equiv b''^*[\{\iota/x''\}]$, since $e[\![\lambda x.\ b]\!] \equiv_\alpha e'[\![\lambda x''.\ b'']\!]$. On the other hand, we have

   $$(\lambda x.\ (e{\uparrow}^{\{x\}})[\![b{\updownarrow}_{\{x\}}^{\mathrm{TY}(e)}]\!])^* \equiv \lambda\iota.\ e^*[\![b^*[\{\iota/x\}]]\!]$$

   as we have shown in the proof of Theorem 5.17. By a similar calculation, but using $x'' \notin \mathrm{TY}(e')$ and $x'' \notin \mathrm{FV}(e')$ in this case, we have

   $$(\lambda x''.\ e'[\![b'']\!])^* \equiv \lambda\iota.\ e'^*[\![b''^*[\{\iota/x''\}]]\!].$$

   Therefore, we have $(\star)$.

2. $e \equiv \{a_1/y_1, \ldots, a_n/y_n\}$: As we have shown in the proof of Theorem 5.17, we have

   $${a_{p(i)}}^* \equiv {a'_{q(i)}}^*, \qquad b^*[\{\iota/x\}][\{\overline{\iota}'/\overline{y}\}] \equiv b''^*[\{\iota/x''\}][\{\overline{\iota}'/\overline{y'}\}]$$

   where $y_{p(1)} \prec \cdots \prec y_{p(n)}$, $\{\overline{\iota}/\overline{y}\} := \{\iota_1/y_{p(1)}, \ldots, \iota_n/y_{p(n)}\}$, $e' \equiv \{a'_1/y'_1, \ldots, a'_n/y'_n\}$, $y'_{q(1)} \prec \cdots \prec y'_{q(n)}$, and $\{\overline{\iota}/\overline{y'}\} := \{\iota_1/y'_{q(1)}, \ldots, \iota_n/y'_{q(n)}\}$. We have also shown

   $$(\lambda x.\ (e{\uparrow}^{\{x\}})[\![b{\updownarrow}_{\{x\}}^{\mathrm{TY}(e)}]\!])^* \equiv \lambda\iota.\ \{{a_{p(1)}}^*/\iota_1, \ldots, {a_{p(n)}}^*/\iota_n\}[\![b^*[\{\sharp\iota/x\}][\{\overline{\iota}/\overline{y}\}]]\!].$$

   By a similar calculation, but using $x'' \notin Y$ and $x'' \notin \mathrm{FV}(a'_i)$ in this case, we have

   $$(\lambda x''.\ e'[\![b'']\!])^* \equiv \lambda\iota.\ \{{a'_{q(1)}}^*/\iota_1, \ldots, {a'_{q(n)}}^*/\iota_n\}[\![b''^*[\{\sharp\iota/x''\}][\{\overline{\iota}/\overline{y'}\}]]\!].$$

   Therefore, we have $(\star)$. □

Theorem 5.22 (Conservativity) *Let a and b be $\lambda\varepsilon$-terms.*
1. If $a \xrightarrow{*}_{\lambda\kappa\varepsilon} b$, then there is a $\lambda\varepsilon$-term b' such that $a \overset{\lambda\varepsilon *}{\longrightarrow} b'$ and $b \equiv_\alpha b'$.
2. If $a \overset{\lambda\varepsilon *}{\longrightarrow} b$, then there is a $\lambda\varepsilon$-term b' such that $a \xrightarrow{*}_{\lambda\kappa\varepsilon} b'$ and $b \equiv_\alpha b'$.

Proof. 1. The reduction sequence $a \xrightarrow{*}_{\lambda\varepsilon} b$ is of the form $a_1 \to_{\lambda\varepsilon} \cdots \to_{\lambda\varepsilon} a_n$ where $a \equiv a_1$ and $b \equiv a_n$. (Note that a_i is a general $\lambda\varepsilon$-term.) We define $\lambda\varepsilon$-terms a'_i $(i = 1, \ldots, n)$ and a''_i $(i = 1, \ldots, n-1)$ as follows. First, we put $a'_1 := a_1$. Next, assume we have a $\lambda\varepsilon$-term a'_i such that $a_i \equiv_\alpha a'_i$. By Lemma

5.21, there are $\lambda\varepsilon$-terms a_i'' and a_{i+1}' such that $a_i'' \rightarrow_{\lambda\varepsilon} a_{i+1}'$, $a_i' \equiv_\alpha^\lambda a_i''$, and $a_{i+1} \equiv_\alpha a_{i+1}'$. Since $a_i' \xrightarrow{\lambda\varepsilon} a_{i+1}'$, we have $a_1 \xrightarrow{\lambda\varepsilon *} a_n'$.
2. The reduction sequence $a \xrightarrow{\lambda\varepsilon *} b$ is of the form $a_1 \xrightarrow{\lambda\varepsilon} \cdots \xrightarrow{\lambda\varepsilon} a_n$ where $a \equiv a_1$ and $b \equiv a_n$. We define $\lambda\varepsilon$-terms a_i° $(i = 1, \ldots, n)$ as follows. First, we put $a_1^\circ := a_1$. Next, assume we have a_i° such that $a_1 \overset{*}{\rightarrow}_{\lambda\varepsilon} a_i^\circ$ and $a_i^\circ \equiv_\alpha a_i$. Since $a_i \xrightarrow{\lambda\varepsilon} a_{i+1}$, we have a_i' and a_{i+1}' such that $a_i' \equiv_\alpha^\lambda a_i$, $a_{i+1}' \equiv_\alpha^\lambda a_{i+1}$, and $a_i' \rightarrow_{\lambda\varepsilon} a_{i+1}'$. So we have $a_i' \equiv_\alpha a_i^\circ$. By Lemma 5.21, there is a $\lambda\varepsilon$-term a_{i+1}° such that $a_i^\circ \rightarrow_{\lambda\varepsilon} a_{i+1}^\circ$ and $a_{i+1}^\circ \equiv_\alpha a_{i+1}'$. So, we have $a_1 \overset{*}{\rightarrow}_{\lambda\varepsilon} a_{i+1}^\circ$ and $a_{i+1}^\circ \equiv_\alpha a_{i+1}$. Therefore, we can take a_n° as b'. □

In [18], it is proved that $\lambda\varepsilon$ is a conservative extension of the simply typed lambda calculus $\lambda\beta$. To express the conservativity within $\lambda\kappa\varepsilon$, we embed the $\lambda\beta$-terms in the $\lambda\kappa\varepsilon$-terms similarly to the $\lambda\varepsilon$-terms. A $\lambda\beta$-term is a $\lambda\kappa\varepsilon$-term such that its typing derivation uses the (axiom), $(\Rightarrow I)$, $(\Rightarrow E)$ rules only, and all the variables used in the (axiom) rule are pure variables. We represent the reduction rule of $\lambda\beta$ in $\lambda\kappa\varepsilon$ as follows: Let a and b be $\lambda\beta$-terms. We write $a \xrightarrow{\beta} b$ if there exist $\lambda\beta$-terms a' and b' such that $a \equiv_\alpha a'$, $b \equiv_\alpha b'$, and $a' \rightarrow_\beta b'$, where the conversion $\mapsto_\beta$ is defined by the following rule.

$(\boldsymbol{\beta})$ $(\lambda x.\, b)a \mapsto_\beta b[\{a/x\}]$.

Then, we have the following theorem.

Theorem 5.23 (Theorem 5.8 [18]) *Let a and b be $\lambda\beta$-terms. Then, $a \xrightarrow{\beta *} b$ iff $a \xrightarrow{\lambda\varepsilon *} b$.*

By combining Theorem 5.22 and 5.23, we have the conservativity of $\lambda\kappa\varepsilon$ over the simply typed $\lambda\beta$-calculus.

Theorem 5.24 (Conservativity) *Let a and b be $\lambda\beta$-terms. Then, $a \xrightarrow{\beta *} b$ iff there is a $\lambda\beta$-term b' such that $a \overset{*}{\rightarrow}_{\lambda\kappa\varepsilon} b'$ and $b \equiv_\alpha b'$.*

Proof. The only-if-part is proved by Theorem 5.22 and 5.23. The if-part is proved as follows. By Theorem 5.22, there is a $\lambda\varepsilon$-term c such that $a \xrightarrow{\lambda\varepsilon *} c$ and $b' \equiv_\alpha c$. Let $a_{n-1} \xrightarrow{\lambda\varepsilon} c$ be the last step of the reduction sequence. Then there exist a_{n-1}' and c' such that $a_{n-1}' \rightarrow_{\lambda\varepsilon} c'$, $a_{n-1} \equiv_\alpha^\lambda a_{n-1}'$, and $c \equiv_\alpha^\lambda c'$. Since c is a $\lambda\beta$-term, $b \equiv_\alpha^\lambda c$. So, we have $b \equiv_\alpha^\lambda c'$. Therefore, we have $a_{n-1} \xrightarrow{\lambda\varepsilon} b$, so that $a \xrightarrow{\lambda\varepsilon *} b$. By Theorem 5.23, we have $a \xrightarrow{\beta *} b$. □

Now, we prove Theorem 3.1 which relates α-equivalence and substitution.

Lemma 5.25 *Let a be a $\lambda\kappa\varepsilon$-term and s be a canonical environment term.*

$$(a[s])^{*\lambda} \equiv a^{*\lambda}[s^{*\lambda}], \quad (a[s])^{*\varepsilon} \equiv (a^{*\varepsilon}[s^{*\varepsilon}])^{*\varepsilon}$$

Proof. The first identity can be proved similarly to Lemma 5.15. The second one can also be proved similarly, but we have to take care of the case that a has a subterm $z[\![b]\!]$ and s is of the form $\{\dots, e/z, \dots\}$ where e is a canonical environment term. $\square$

Proof of Theorem 3.1. By Lemma 5.25, we have $(a[s])^{*\lambda} \equiv a^{*\lambda}[s^{*\lambda}] \equiv b^{*\lambda}[s^{*\lambda}] \equiv (b[s])^{*\lambda}$ and $(a[s])^{*\varepsilon} \equiv (a^{*\varepsilon}[s^{*\varepsilon}])^{*\varepsilon} \equiv (b^{*\varepsilon}[s^{*\varepsilon}])^{*\varepsilon} \equiv (b[s])^{*\varepsilon}$. $\square$

Theorem 5.26 (Strong Normalizability) *If $\Gamma \vdash a : A$, then a is strongly normalizable.*

Proof. We can prove this theorem in the same way as the strongly normalizability theorem of $\lambda\varepsilon$ [18], because we can treat the cases of (abs) and (absapp) similarly to the cases of (fun) and (funapp). $\square$

6 Related Works

In this section, we compare our calculus with some related works.

In the formulation of contexts in $\lambda\kappa\varepsilon$, first-class environments (or explicit substitutions) play an essential role. Abadi et al [1] first introduced explicit substitutions, and there are many interesting works on this subject since then. However, to our knowledge, the $\lambda\varepsilon$-calculus [18] is the only language which is pure in our sense. We therefore designed our calculus as an extension of $\lambda\varepsilon$.

The style of the presentation of our paper is very close to that of Hashimoto-Ohori [10]. Both our calculus and the calculus presented in [10] are simply typed calculi which include simply typed $\lambda\beta$-calculus as their subcalculus. The system in [10] enjoys subject reduction property and is confluent. However, neither conservativity over simply typed $\lambda\beta$-calculus nor strong normalizability are shown in the paper. Therefore, it is not known whether their system is pure in our sense[4]. Also, their calculus has severe restrictions in that (i) each context may have at most one hole in it, and (ii) as we have explained in section 1, the application of the β-reduction is allowed only when

[4]Sakurada [15] proved the strong normalizability of Hashimoto-Ohori's calculus by interpreting it in $\lambda\varepsilon$.

the β-redex has no hole in it. Our calculus does not have such restrictions and β-reduction and hole-filling always commute.

Dami's calculus λN [5] is a very simple and powerful calculus with named variables. It is possible to represent both contexts and hole-filling in λN. However, this is done by a translation of $\lambda\beta$ calculus into λN. Therefore, it is hard to read the translated terms as contexts. On the other hand, Mason [12] introduces a system with first-class contexts in which contexts are directly represented as terms in his calculus. However, he defines hole-filling as a meta-level operation. It is therefore not possible to compute hole-filling within his system. Unlike these systems, in $\lambda\kappa\varepsilon$, contexts are directly representable as terms of $\lambda\kappa\varepsilon$, and we can compute hole-filling within $\lambda\kappa\varepsilon$.

Sands [16] uses Pitts' [14] definition of contexts and shows that hole-filling commutes with many relations on terms including α-equivalence. Pitts defines contexts by representing holes by (higher-order) function variables where each function variable has a fixed arity, and by representing hole-filling by substitution of a meta-abstraction for a function variable. For example, the term

$$\lambda x.\ (\lambda y.\ x + X{\bullet}\{x/x, y/y\})3$$

in $\lambda\kappa\varepsilon$ can be expressed by

$$\lambda x.\ (\lambda y.\ x + \xi(x, y))3$$

where ξ is a binary function variable, and the substitution

$$\{\kappa\{x, y\}.\ x + y/X\}$$

in $\lambda\kappa\varepsilon$ can be expressed by

$$[(x, y)(x + y)/\xi].$$

As can be seen by this example, Pitts' representation of contexts is structurally similar to ours, but the statuses of contexts are quite different. That is, Pitts' contexts are meta-level objects outside the object language (λ calculus in case of the above example) and our contexts are internal objects of our language $\lambda\kappa\varepsilon$. Because of this meta-level status of Pitts' contexts, Sands [16] could successfully attach contexts to many languages and could prove that hole-filling commutes with many rules in a uniform way. In contrast to this, we have been interested in internalizing such meta-level objects as contexts and environments so that we can enrich λ-calculus to a more powerful programming language.

7 Conclusion

We have introduced a simply typed λ-calculus which has both contexts and environments as first-class values. We have shown that our calculus, $\lambda\kappa\varepsilon$, is a conservative extension of the simply typed $\lambda\beta$-calculus, enjoys subject reduction property, is confluent and strongly normalizing. Thus we have shown that our language is pure in the sense of [18] and also we have realized our hope, which we stated in the conclusion of [18], to design a pure language that has both contexts and environments as first-class values. To the best of our knowledge, $\lambda\kappa\varepsilon$ is the first such language.

We have also introduced a new method of defining substitution which is conceptually simpler than traditional methods. We think that our method is also suitable for representing terms and computing substitutions on the computer.

References

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Levy. Explicit Substitutions. *Journal of Functional Programming*, 1:375–416, 1991.

[2] H. P. Barendregt. *The Lambda Calculus, Its Syntax and Semantics.* North-Holland, 1981.

[3] R. Bloo and K.H. Rose. Preservation of Strong Normalization in Named Lambda Calculi with Explicit Substitution and Garbage Collection. In van Vliet J. C., editor, *Proc. CSN'95 (Computer Science in Netherlands)*, 1995. (`ftp://ftp.diku.dk/diku/semantics/papers/D-246.ps`).

[4] M. Bognar and R. de Vrijer. A calculus of lambda calculus contexts. *J. Automated Reasoning*, 27:29–59, 2001.

[5] L. Dami. A Lambda-Calculus for Dynamic Binding. *Theoretical Computer Science*, 192:201–231, 1998.

[6] D. G. de Bruijn. Lambda Calculus Notation with Nameless Dummies, a Tool for Automatic Formula Manipulation, with Application to the Church-Rosser Theorem. *Indag. Math.*, 34:381–392, 1972.

[7] M. Fiore, G. Plotkin, and D. Turi. Abstract Syntax and Variable Binding (Extended Abstract). In *Proc. 14th Symposium on Logic in Computer Science*, pages 193–202, 1999.

[8] J. Garrigue and H. Aït-Kaci. The Typed Polymorphic Label-Selective λ-Calculus. In *Proc. 21st Annual ACM Symposium on Principles of Programming Languages*, pages 35–48, 1994.

[9] C. Gunter. *Semantics of Programming Languages.* The MIT Press, 1992.

[10] M. Hashimoto and A. Ohori. *A Typed Context Calculus.* Preprint RIMS-1098. Research Institute for Mathematical Sciences, Kyoto University, 1996. (Journal version is to appear in *Theoretical Computer Science*.).

[11] S.-R. Lee and D. P. Friedman. Enriching the Lambda Calculus with Contexts: Toward a Theory of Incremental Program Construction. In *ACM SIGPLAN Notices, Proc. International Conference on Functional Programming*, pages 239–250, 1996.

[12] I. Mason. Computing with Contexts. *Higher-Order and Symbolic Computation*, 12:171–201, 1999.

[13] P.-A. Melliès. Typed λ-calculi with explicit substitutions may not terminate. In *Proc. Second International Conference on Typed Lambda Calculi and Applications*, LNCS 902, pages 328–349, 1995.

[14] A. M. Pitts. Some Notes on Inductive and Co-Inductive Techniques in the Semantics of Functional Programs. Notes Series BRICS-NS-94-5, Department of Computer Science, University of Aarhus, 1994.

[15] H. Sakurada. An Interpretation of a Context Calculus in an Environment Calculus. *Transactions of Information Processing Society of Japan: Programming*, 41(SIG 9 (PRO 8)):1–7, 2000.

[16] D. Sands. Computing with Contexts - a simple approach. *Electronic Notes in Theoretical Computer Science*, 10, 1998.

[17] M. Sato. Theory of Symbolic Expressions, II. *Publ. of Res. Inst. for Math. Sci., Kyoto Univ.*, 21:455–540, 1985.

[18] M. Sato, T. Sakurai, and R. Burstall. Explicit Environments. *Fundamenta Informaticae*, 45(1-2):79–115, 2001.

[19] C. Talcott. A Theory of binding structures and applications to rewriting. *Theoretical Computer Science*, 112(1):99–143, 1993.