A Second-Order Context Calculus

Azza A. Taha, Masahiko Sato, and Yukiyoshi Kameyama
Graduate School of Informatics, Kyoto University
E-mail: {azza,masahiko,kameyama}@kuis.kyoto-u.ac.jp

Abstract

We introduce a second-order context calculus λ_{2xc} which has contexts as first-class objects and hole-filling as an explicit operation. In the representation of contexts both term variables and type variables can get bound after filling in the hole.

We show that λ_{2xc} is confluent, conservative over the second order lambda calculus, satisfying the strong normalization property and the type system has the subject reduction property.

In order to avoid the unwanted capture of variables during substitution, the push and pull techniques are used to rename free variables instead of the usual α-conversion which renames bound variables. We also introduce a new definition for the push operator which is simpler than the ordinary one and general, therefore it can be used in any calculi with different terms.

1 Introduction

In recent years, there have been several attempts to formalize and compute with contexts–terms with holes in them– see, for example, Dami [3], Hashimoto-Ohori [5], Lee-Friedman [7], Mason [8], Sato-Sakurai-Kameyama [10] and Sands [9]. In all these calculi, contexts are only able to provide bindings for the term variables. In this paper, we introduce a second-order context calculus λ_{2xc} in which both abstraction on types and representation of contexts are possible. So, in the representation of contexts here both term variables and type variables can get bound after filling in the hole. Also, most of the existing calculi of contexts are first-order and cannot define useful data types internally. But, in our calculus, we can define useful data types such as boolean, integers, list etc. This means that our calculus is closer to conventional functional programming language than existing calculi of contexts.

We present our calculus λ_{2xc} as an extension of the calculus λ_{xc} [11] which is a type-free calculus that has contexts as first-class objects and hole-filling as an explicit operation. In our calculus holes are represented by ordinary variables of appropriate types and hole-filling is represented by ordinary application together with a new abstraction mechanism which represents the variables intended to be bound after filling in the hole.

The operation for a context is to fill its hole with a term where in performing this filling operation free variables can get captured. For example, consider the context $\Lambda\alpha.\lambda x^{\alpha}.[.]$, where [.] represents a hole. If the term x^{α} is filled in this hole we get $\Lambda\alpha.\lambda x^{\alpha}.x^{\alpha}$ in which both x and α get bound after filling in this hole, and this is the difference between hole-filling and capturing-avoiding substitution in the lambda calculus.

As in the first order case, hole-filling is at odds with the ordinary β-substitution. To see this problem, consider the context $\Lambda\alpha.\lambda x^{\alpha}.[.]$. If we fill the hole in this term with x^{α} and then substitute A for α we get $\lambda x^{A}.x^{A}$, while if we substitute A for α first and then fill the hole with x^{α}, we get $\lambda x^{A}.x^{\alpha}$, which are not the same.

Also, α-conversion and hole-filling are at odds. For example, in the term $\Lambda\alpha.\lambda x^{\alpha}.[.]$, if we fill this hole with x^{α} we get $\Lambda\alpha.\lambda x^{\alpha}.x^{\alpha}$ which is α-equivalent to $\Lambda\beta.\lambda x^{\beta}.x^{\beta}$, while if we perform the α-equivalence rule first and then the hole filling we get $\Lambda\beta.\lambda x^{\beta}.x^{\alpha}$, which are not α-equivalent.

To solve this, we represent the term $\Lambda\alpha.\lambda x^{\alpha}.[.]$ as:

$$C \equiv \Lambda\alpha.\lambda x^{\alpha}.(X \bullet [\langle \alpha := \alpha \rangle] \bullet [\langle x^{\alpha} := x^{\alpha} \rangle])$$

where X represents the hole and its type is of the form $\alpha^{\{x^{\alpha}\}\{\alpha\}}$. So, when we fill the hole X with x^{α}, we substitute $(\{\alpha\}.\{x\}.x^{\alpha})$– this abstraction dictates that the variable x and its type α are intended to be bound– for it in C and we get:

$$\Lambda\alpha.\lambda x^{\alpha}.(\{\alpha\}.\{x\}.x^{\alpha}) \bullet [\langle \alpha := \alpha \rangle] \bullet [\langle x := x \rangle]$$

which is reduced to $\Lambda\alpha.\lambda x^{\alpha}.x^{\alpha}$. Then if A is applied to this term we get $\lambda x^{A}.x^{A}$. While if A is applied to the term C and then the hole is filled with x^{α}, we get the same result (note that, the explicit substitution is used to keep the value of α.)

Also, filling the term C with x^{α} first and then α-conversion rule is used we get $\Lambda\beta.\lambda x^{\beta}.x^{\beta}$, and doing the α-conversion rule first we get:

$$\Lambda\beta.\lambda x^{\beta}.(X \bullet [\langle \alpha := \beta \rangle] \bullet [\langle x^{\beta} := x^{\beta} \rangle])$$

which is reduced to the same result $\Lambda\beta.\lambda x^{\beta}.x^{\beta}$ after filling the hole X with x^{α}.

In our calculus, the usual α-conversion cannot be used to rename the variables bound by the $\{x\}$ and $\{\gamma\}$ binders for the following reason:

Consider the following term:

$$((\{\gamma\}.\{x\}.xy\gamma\beta)\langle \beta := \gamma \rangle\langle y := x \rangle) \bullet [\langle \gamma := A \rangle] \bullet [\langle x := z \rangle]$$

It is known that before performing the substitution $\langle \beta := \gamma \rangle$ the α-conversion rule must be used to rename the variable γ to another new variable to avoid the unwanted capture of the γ in this substitution by that in the binder $\{\gamma\}$, so the correct result should be:

$(\{\gamma_1\}.\{x\}.xy\gamma_1\gamma)\langle y := x\rangle) \bullet [\langle \gamma := A\rangle] \bullet [\langle x := z\rangle]$

which cannot be further reduced in our calculus since the type variable γ in the substitution $\langle \gamma := A\rangle$ must match the binder $\{\gamma_1\}$. So, another method is needed to prevent the unwanted capture of variables during substitution while keeping bound variable names. Sato et al [10] defined a method to rename free variables (if necessary) systematically instead of the bound variables to avoid the unwanted capture of variables. The same method will be used here, and in the last example the type variable γ in $\langle \beta := \gamma\rangle$ is renamed to $\#\gamma$. We say that the variable γ in the explicit substitution is *pushed through* the binding variable γ, written as $\gamma \uparrow \gamma$, and we get $\#\gamma$ as the result.

So, the above term is calculated as follows:

$(\{\gamma\}.\{x\}.xy\gamma\#\gamma)\langle y := x\rangle \bullet [\langle \gamma := A\rangle] \bullet [\langle x := z\rangle]$

which can be reduced to :

$(\{x\}.xyA\gamma)\langle y := x\rangle \bullet [\langle x := z\rangle]$

Note that the type variable $\#\gamma$ is *pulled* from γ to get γ since it is no longer in the scope of the binder $\{\gamma\}$.

For the $\{x\}$ binder, the situation is the same as in the case of type variable. We also rename the free term variable to avoid the unwanted capture of free variables and the above term reduces to:

$$(\{x\}.x\#xA\gamma) \bullet [\langle x := z\rangle]$$

which is finally reduced to $zxA\gamma$.

The rest of the paper is organized as follows. In Section 2 we give the type system, and the reduction rules are given in Section 3. In Section 4 we show that our calculus has some desired properties such as subject reduction, confluence, conservativity over the second order lambda calculus and has the strong normalization property. In Section 5, we give concluding remarks.

2 The Type System

Notations $A\langle \overline{\nu} := \overline{B}\rangle$ will be used to abbreviate $A\langle \nu_1, ..., \nu_n := B_1, ..., B_n\rangle$, $\overline{\alpha}$ will be used to abbreviate $\alpha_1, ..., \alpha_n$, $\overline{x^B}$ to abbreviate $x_1^{B_1}, ..., x_n^{B_n}$, $M\langle \overline{x^B} := \overline{N}\rangle$ to abbreviate $M\langle x_1^{B_1}, ..., x_n^{B_n} := N_1, ..., N_n\rangle$, and $\overline{\Delta}$ to abbreviate $\Delta_1, ..., \Delta_n$.

2.1 Types

Assume that PT is the set of *pure type variables* $\alpha, \beta, \gamma ...$ and TV is the set of *type variables* given by $TV ::= PT \mid \#TV$. We will use $\nu, \mu, \zeta ...$ as meta variables for type variables.

Types are defined as:

$$A, B ::= K \mid TV \mid A \Rightarrow B \mid \forall\alpha.A \mid A\langle \overline{\nu} := \overline{B}\rangle \mid A^{\{\overline{\alpha}\}} \mid A^{\{\overline{x^B}\}}$$

where K ranges over atomic types, $\{\overline{\alpha}\}$ over finite sets of type variables and $\{\overline{x^B}\}$ over finite set of term variables. $A^{\{\overline{\alpha}\}}$ and $A^{\{\overline{x^B}\}}$ are the types of the hole, where $\{\overline{\alpha}\}$ ($\{\overline{x^B}\}$) describes the set of type variables (term variables, respectively) being captured when they are filled in the hole, canonical elements of these two types are abstracts of the form $\{\overline{\alpha}\}.M$ and $\{\overline{x^B}\}.M$ respectively where M is of type A. A term of the form $(\{\overline{\alpha}\}.\{\overline{x^B}\}.M)$ must be reduced in two stages, instead of one, when it is applied to the explicit substitutions using the application $\bullet$, for the types $\overline{B}$ of $\overline{x}$ is not necessary match its type in the explicit substitution. As for example in the term, $(\{\overline{\alpha}\}.\{\overline{x^\alpha}\}.M) \bullet [\langle \overline{\alpha} := \overline{C}\rangle] \bullet [\langle \overline{x^C} := \overline{N}\rangle]$. This is the reason that we have two abstraction sets instead of one and consequently the reason that we have two types for the hole.

Note that, we have the following notions of boundness in types: the type variables $\{\overline{\alpha}\}$, α, $\overline{\nu}$ are considered bound in A in the types $A^{\{\overline{\alpha}\}}$, $\forall\alpha.A$ and $A\langle \overline{\nu} := \overline{B}\rangle$ respectively. The explicit substitution $A\langle \overline{\nu} := \overline{B}\rangle$ stands for substituting B_1 for ν_1, B_2 for ν_2,..., B_m for ν_m in A simultaneously. For technical reasons, $\texttt{FTV}(A)$ is defined as a sequence of *free type variables* occurring in the type A, rather than a set of free variables. Its formal definition can be deduced from the general definition of free variables in Section 2.3.

2.2 Terms

Assume that P is the set of *pure term variables* $x^A, y^B, z^C, ...$ and V is the set of *term variables* given by $V ::= P \mid \#V$. We will use $v^A, w^B, ...$ as meta variables for term variables. Assume that $pure_v$ is a function which takes a term variable and returns its pure version, e.g. $pure_v(x^A) = x^A$ and $pure_v(\#y^B) = y^B$. *Terms* are defined as:

$$\begin{aligned} M, N \quad ::= \quad & V^A \\ & \mid \lambda x^A.M \mid MN \\ & \mid \Lambda\alpha.M \mid MA \\ & \mid M\langle \overline{v^B} := \overline{N}\rangle \\ & \mid \{\overline{\alpha}\}.M \mid M \bullet [\langle \overline{\alpha} := \overline{C}\rangle] \\ & \mid \{\overline{x^B}\}.M \mid M \bullet [\langle \overline{x^B} := \overline{N}\rangle] \\ & \mid M\langle \overline{\nu} := \overline{B}\rangle \end{aligned}$$

Note that we sometimes write $v, u, ...$ for $v^A, u^B, ...$ for the sake of notational simplicity.

We have the following notion of boundness in terms: the term variables $x, \overline{v}, \overline{x}$ are considered bound in M in the terms $\lambda x.M$, $M\langle \overline{v} := \overline{N}\rangle$ and $\{\overline{x^B}\}.M$ respectively. The type variables $\overline{\alpha}$, α, and $\overline{\nu}$ are considered bound in M in the terms $\{\overline{\alpha}\}.M, \Lambda\alpha.M$ and $M\langle \overline{\nu} := \overline{B}\rangle$ respectively.

The explicit substitution $M\langle \overline{\nu} := \overline{B}\rangle$ ($M\langle \overline{v^B} := \overline{N}\rangle$) stands for substituting B_1 for ν_1,..., B_n for ν_n (N_1 for $v_1^{B_1}, ..., N_n$ for $v_n^{B_n}$, respectively) in M simultaneously.

We will restrict all the bound term variables (type variable) – except those bound by the explicit substitution– in our calculus to pure term variables (pure type variable, respectively), which makes our calculus a little simpler and

it does not affect our intended motivation about context.

The expression $\texttt{FV}(M)$ stands for the sequence of *free term variables* occurring in the term M. The expression $\texttt{FTV}(M)$ stands for the sequence of *free type variable* occurring in the term M. The formal definition can be deduced from the general definition of free variables in Section 2.3.

2.3 A New Definition for the Push Operator

In the ordinary $\lambda\beta$-calculus, it is known that before reducing any term we have to check whether the α-conversion rule is necessary or not to avoid problems with variable capture. For example, before reducing the term $(\lambda y.\lambda x.y + x)x$, the α-conversion rule must be used to rename the inner variable x to another new name to avoid capturing the outer free variable with the inner λx.

For the reason explained in the introduction, the α-conversion rule is no longer appropriate for our calculus. Instead the push operator is used to rename the free variable that would otherwise be captured. Therefore, in the above term the variable x is renamed to $\#x$ (we say that the free variable x is pushed through the bound variable x, written as $x \uparrow x$, and we get $\#x$ as a result.) So, the correct result of reducing this term is $\lambda x.\#x+x$. It should be clear that in this term $\#x$ is different from x, and outside the scope of λx, $\#x$ refers to x. Suppose now that this term is applied to z, therefore $\#x$ will no longer be in the scope of λx and we have to remove the # from $\#x$ and we get $x + z$ as the result of this reduction.

In what follows we give a new and general definition for the push operator from which a definition of pushing a $\lambda_{2\text{xc}}$-terms through term variables or type variables and pushing a $\lambda_{2\text{xc}}$-types through type variables can be deduced. For that purpose, we assume that we have a general syntax for a term in which a term is either an abstraction, a structured term, a variables or a constant. Then the notion of pushing this term through variables will be defined. The $\lambda_{2\text{xc}}$-terms and types can easily be encoded in the general syntax of a term and the definitions of pushing a $\lambda_{2\text{xc}}$ type through a type variable and pushing a $\lambda_{2\text{xc}}$-term through a term variable or a type variable can be deduced. The encoding of the $\lambda_{2\text{xc}}$-types and terms in the general syntax for a term will be given in the appendix.

Assume that *Variable* is the set of pure variables $a, b, ...$, and *Var* is the set of variables given by Var ::= Variable | #Var. Assume that *pure* is a function which takes a variable and returns its pure version e.g. $pure(\#^3 a) = a$. Terms are defined by the following grammar:

E	::= Var | Const
	| StructuredTerm
	| Abstraction
Const	::= c | c_1 | c_2 |...
StructuredTerm	::= $(E_1, ..., E_n)$
Abstraction	::= {Var, ..., Var}[E]

As a shorthand we will write a for $\#^0 a$ and $\#^{(n+1)}\ a$ for $\#(\#^n\ a)$.

The *order relation* $\leq$ on variables is defined as:
$\#^n a \leq \#^m b$ iff $n \leq m$ and $a = b$ (where $n, m \in N$) or $a \prec_{lex} b$, where $\prec_{lex}$ is the lexicographic order on variable names.

Let H be a finite set of variables, then $\#^l b$ is called the *smallest element in the set* H if: $\#^l b \in H$ and $\forall \#^m a_i \in H$ we have $\#^l b \leq \#^m a_i$.
The largest element in the set is defined similarly.

The *minus operator* subtracts a variable from a sequence of variables and the result is a sequence of variables of the form (Var,...,Var). We will use $\circ$ to denote the append function on sequences.

1. $(\#^{q_1} a_1, ..., \#^{q_n} a_n) - (\#^l\ b)$, is defined as:
   (a) If $n = 0$ then ().
   (b) If $a_n = b$ then
      i. If $q_n = l$ then $(\#^{q_1} a_1, ..., \#^{q_{(n-1)}} a_{(n-1)}) - (\#^l\ b)$.
      ii. If $q_n > l$ then $((\#^{q_1} a_1, ..., \#^{q_{(n-1)}} a_{(n-1)}) - (\#^l\ b)) \circ (\#^{(q_n)-1} a_n)$.
      iii. If $l > q_n$ then $((\#^{q_1} a_1, ..., \#^{q_{(n-1)}} a_{(n-1)}) - (\#^l\ b)) \circ (\#^{q_n} a_n)$.
   (c) If $a_n \neq b$ then $((\#^{q_1} a_1, ..., \#^{q_{(n-1)}} a_{(n-1)}) - (\#^l b)) \circ (\#^{q_n} a_n)$.

   Example 1
   $(\#^2 b, \#^5 a, \#^3 a, \#^2 a) - (\#^3 a) = (\#^2 b, \#^4 a, \#^2 a)$.

2. Let H be a finite set of variables, then $(\#^{q_1} a_1, ..., \#^{q_n} a_n) - H$ is defined inductively on the number of elements in H as follows:
   (a) If $H = \emptyset$, then $(\#^{q_1} a_1, ..., \#^{q_n} a_n)$.
   (b) Otherwise, $\exists\ H_1\ \exists\ \#^l\ b_{k_1}$ s.t. $H = H_1 \cup \{\#^l b_{k_1}\}$, where $\#^l\ b_{k_1}$ is the smallest element in the set H, and $(\#^{q_1} a_1, ..., \#^{q_n} a_n) - H = ((\#^{q_1} a_1, ..., \#^{q_n} a_n) - H_1) - (\#^l b_{k_1})$.

   i.e. in this definition we subtract the greatest element in the set first, for example, $(\#^2 a) - \{a, \#^2 a\} = \emptyset$, and $\#^3 a - \{\#^2 a, \#a, a, b\} = a$.

Let Γ be a sequence of pure and non pure variables in which the order is important. The following rules are used to derive judgments of the form $\Gamma\ \texttt{exp}\ \#^n a$:

$$\frac{}{\Gamma, \#^n a\ \texttt{exp}\ \#^n a}$$

$$\frac{\Gamma\ \texttt{exp}\ (\#^n a) - (\#^m a)}{\Gamma, \#^m a\ \texttt{exp}\ \#^n a}\ (m \neq n)$$

$$\frac{\Gamma\ \texttt{exp}\ \#^n a}{\Gamma, \#^m b\ \texttt{exp}\ \#^n a}\ (b \neq a)$$

We will say that $\#^n a$ is bound in Γ iff Γ $\texttt{exp}$ $\#^n a$ is derivable. If Γ $\texttt{exp}$ $\#^n a$ is not derivable then $\#^n a$ is said to be free in Γ.

Example 2

1. The variable $\#^2 a$ is bound in $\#a, a$. The variable $\#a$ is bound in a, a. Also the variable $\#a$ is bound in $\#a$, and a is bound in $a, \#a$.
2. The variable $\#a$ is free in $\#a, a$, and $\#^2 a$ is free in $a, \#a$.

The push operator is defined as:

1. Push $\#^n b$ through $\#^m a$, written as $\#^n b \uparrow_\Gamma \#^m a$, is defined in figure 1.
2. Let H be a finite set of variables, then $\#^n b \uparrow_\Gamma H$ is defined inductively on the number of elements in H as follows:
   (a) If $H = \emptyset$ then $\#^n b \uparrow_\Gamma H = \#^n b$.
   (b) Otherwise, $\exists H_1 \exists \#^m a$ s.t. $H = H_1 \cup \{\#^m a\}$, where $\#^m a$ is the largest element in the set H and $\#^n b \uparrow_\Gamma H = (\#^n b \uparrow_\Gamma H_1) \uparrow_\Gamma \#^m a$.

   i.e. in this definition we push the smallest element in the set first, for example $a \uparrow \{a, \#a, b\} = \#^2 a$ and $a \uparrow \{\#a, b\} = a$.
3. Let G be a finite set of variables $\{\#^{l_1} a_1, ..., \#^{l_n} a_n\}$, then $G \uparrow_\Gamma H = \{\#^{l_1} a_1 \uparrow_\Gamma H, ..., \#^{l_n} a_n \uparrow_\Gamma H\}$.
4. Push E through H, $E \uparrow_\Gamma H$, is defined inductively as:
   (a) When E is a variable, it is defined as above.
   (b) $c \uparrow_\Gamma H = c$.
   (c) $(E_1, ..., E_n) \uparrow_\Gamma H = (E_1 \uparrow_\Gamma H, ..., E_n \uparrow_\Gamma H)$.
   (d) $(\{\#^{q_1} b_1, ..., \#^{q_n} b_n\}[E]) \uparrow_\Gamma H =$
   $\{\#^{q_1} b_1, ..., \#^{q_n} b_n\}[E \uparrow_{\Gamma, \Gamma_1} (H$
   $\uparrow \{\#^{q_1} b_1, ..., \#^{q_n} b_n\})]$.

   Where Γ_1 is a sequence consisting of the elements of the set $\{\#^{q_1} b_1, ..., \#^{q_n} b_n\}$ and the order of putting these elements in Γ_1 is as follows:
   $\Gamma_1 = \#^{l_1} b_{k_1}, \#^{l_2} b_{k_2}, ..., \#^{l_n} b_{k_n}$ s.t.
   Γ_1 (considered as a set) =
   $= \{\#^{q_1} b_1, ..., \#^{q_n} b_n\}$ and $\#^{l_1} b_{k_1} \leq \#^{l_2} b_{k_2} ... \leq \#^{l_n} b_{k_n}$
   i.e. we have to put the smallest element in the set first followed by its greater element and so on until the greatest one be in the last. For Example,
   $(\{\#^2 a, a\}[\{\#a, \#^2 a\}[\#^4 a]]) \uparrow a$
   $= \{\#^2 a, a\}[(\{\#a, \#^2 a\}[\#^4 a]) \uparrow_{a, \#^2 a} \#a]$
   $= \{\#^2 a, a\}[\{\#a, \#^2 a\}[\#^4 a \uparrow_{a, \#^2 a, \#a, \#^2 a} \#^3 a]]$
   $= \{\#^2 a, a\}[\{\#a, \#^2 a\}[\#^4 a]]$

Note that, when Γ is an empty sequence we write $E \uparrow H$ for $E \uparrow_\Gamma H$.

Example 3

1. $(\{a\}[\#a]) \uparrow \#a = \{a\}[\#a]$.
2. $(\{a\}[\{a\}[\#a]]) \uparrow a = \{a\}[\{a\}[\#a]]$.
3. $(\{\#a\}[\{a\}[\#a]]) \uparrow a = \{\#a\}[\{a\}[\#^3 a]]$.

The sequence of *Free Variables* occurred in E, $\texttt{FV}(E)$, is defined as:

1. $\texttt{FV}(\#^n a) = (\#^n a)$.
2. $\texttt{FV}(c) = ()$.
3. $\texttt{FV}(E_1, ..., E_n) = \texttt{FV}(E_1) \circ ... \circ \texttt{FV}(E_n)$.
4. $\texttt{FV}(\{\#^{q_1} b_1, ..., \#^{q_n} b_n\}[E]) =$
   $\texttt{FV}(E) - \{\#^{q_1} b_1, ..., \#^{q_n} b_n\}$.

For example, $\texttt{FV}(\{a, \#a\}[\#^2 a]) = a$.

The *pull operator* $\downarrow$ is defined as an inverse operator of $\uparrow$. Pull E from H, written as $E \downarrow H$, is defined only when $H \cap \texttt{FV}(E) = \emptyset$. (Note that we sometimes regard a sequence as a set consisting of its elements).

Example 4

1. $\#^2 a \downarrow a = \#a$, and $\#a \downarrow \#^2 a = \#a$.
2. $\#^2 a \downarrow b = \#^2 a$, and $\#^2 a \downarrow \#^2 a$ is undefined.
3. $\#^2 a \downarrow \{a, \#a\} = a$. (Note that, we pull $\#^2 a$ from the largest element in the set first.)
4. $(\{\#a\}[\#^2 a]) \downarrow a = \{\#a\}[\#^2 a \downarrow_{\#a} (a \uparrow \#a)]$
   $= \{\#a\}[\#^2 a \downarrow_{\#a} a] = \{\#a\}[a]$.
5. $(\{\#^2 a\}[\#^3 a]) \downarrow \#^2 a = \{\#^2 a\}[\#^3 a \downarrow_{\#^2 a} (\#^2 a \uparrow \#^2 a)]$
   $= \{\#^2 a\}[\#^3 a \downarrow_{\#^2 a} \#^3 a]$ which is undefined.

We recently noticed that the new definition of the push operator defined above has the same structure as the ϕ function defined in Section 6.2 [4].

2.4 The Typing Rules

Since we apply explicit substitutions on types, we sometimes need to reduce types when we assign them to terms. So we give the type reduction rules before the typing rules. The Type Reduction Rules $\rightarrow_T$ are defined as:

$$\#^n b \uparrow_\Gamma \#^m a = \begin{cases} \#^n b & \text{if } \#^n b \text{ is bound in } \Gamma, \text{or } n < m \text{ or } b \neq a \\ \#^{(n+i)} a & \text{if } b = a \text{ and } \#^n b \text{ is free in } \Gamma, \text{ and } n \geq m \\ & \text{where } i \text{ is the least natural number } (> 0) \\ & \text{satisfying : } \#^{(n+i)} a \text{ is free in } \Gamma. \end{cases}$$

Figure 1: The definition of $\#^n b \uparrow_\Gamma \#^m a$

(Tvar) $\nu_i\langle\overline{\nu} := \overline{A}\rangle \rightarrow_T A_i$.

(Tgc) $B\langle\overline{\nu} := \overline{A}\rangle \rightarrow_T B \downarrow \{\overline{\nu}\}$, if $\{\overline{\nu}\} \cap \mathtt{FTV}(B) = \emptyset$.

(Tfun) $(B \Rightarrow C)\langle\overline{\nu} := \overline{A}\rangle \rightarrow_T B\langle\overline{\nu} := \overline{A}\rangle \Rightarrow C\langle\overline{\nu} := \overline{A}\rangle$.

(Tabs) $(\forall\alpha.B)\langle\overline{\nu} := \overline{A}\rangle \rightarrow_T \forall\alpha.B\langle\overline{\nu} \uparrow \alpha := \overline{A} \uparrow \alpha\rangle$.

(Thvar$_1$) $(A^{\{\overline{\alpha}\}})\langle\overline{\nu} := \overline{C}\rangle \rightarrow_T A\langle\overline{\nu} \uparrow \{\overline{\alpha}\} := \overline{C} \uparrow \{\overline{\alpha}\}\rangle^{\{\overline{\alpha}\}}$.

(Thvar$_2$) $(A^{\{\overline{x^B}\}})\langle\overline{\nu} := \overline{C}\rangle \rightarrow_T A\langle\overline{\nu} := \overline{C}\rangle^{\{\overline{x^B}\langle\overline{\nu}:=\overline{C}\rangle\}}$.

We write $A \rightarrow_T B$ if B is obtained from A by replacing a subtype A' in A by B' s.t. $A' \rightarrow_T B'$. The reflexive and transitive closure of this reduction is denoted by $\stackrel{*}{\rightarrow}_T$. A $\lambda_{2\mathrm{xc}}$-type A is said to be in T-normal form if $A \rightarrow_T B$ holds for no B.

Lemma 1 $\rightarrow_T$ on $\lambda_{2\mathrm{xc}}$ types is noetherian and confluent.

Proof Easy. □

A *declaration* is an expression of the form v^A where v is a variable and A is a type. A *Type Assignment* is a sequence of declarations. We will use $\Gamma, \Delta, \cdots$ as meta variables for type assignments. Γ, Δ will be used for the concatenation of the type assignments Γ and Δ.

A *typing judgment* is an expression of the form $\Gamma \vdash M : A$ where Γ is a type assignment, M is a (typed) term and A is a type. The *typing rules* are defined as follows that are used to derive typing judgements, where rules whose names end with 'I'('E') introduce (eliminate, respectively) the types mentioned in the rule names.

Recall that $\mathrm{FV}(M)$ is the sequence of free variables in M. It can easily be verified that if $\Gamma \vdash M : A$ is derivable, then Γ is equal to $FV(M)$. And $FV(M) = \emptyset$ iff $\vdash M : A$ is derivable.

Note that, if $\Gamma = \{x_1^{A_1}, ..., x_n^{A_n}\}$, then $\mathtt{FTV}(\Gamma) = \mathtt{FTV}(A_1) \circ ... \circ \mathtt{FTV}(A_n)$, where $\circ$ is the append function on sequences. And if $A_i \stackrel{*}{\rightarrow}_T B_i$ for $1 \leq i \leq n$, then $\Gamma \stackrel{*}{\rightarrow}_T \Gamma'$ where $\Gamma' = \{x_1^{B_1}, ..., x_n^{B_n}\}$. If we have $\Gamma\langle\alpha := A\rangle$ and $A \stackrel{*}{\rightarrow}_T B$ then $\Gamma\langle\alpha := A\rangle \rightarrow_T \Gamma\langle\alpha := B\rangle$
Also, $\Gamma\langle\alpha := B\rangle \rightarrow_T \{x_1^{A_1\{\alpha:=B\}}, ..., x_n^{A_n\{\alpha:=B\}}\}$ And $\Gamma \downarrow \alpha = \{x_1^{A_1\downarrow\alpha}, ..., x_n^{A_n\downarrow\alpha}\}$.

The *Typing Rules* are defined as:

$$\frac{}{v^A \vdash v^A : A}\ (\text{assume})$$

$$\frac{\Gamma \vdash M : B}{\Gamma - \{x^A\} \vdash \lambda x^A.M : A \Rightarrow B}\ (\Rightarrow I)$$

$$\frac{\Gamma \vdash M : A \Rightarrow B \quad \Delta \vdash N : A}{\Gamma, \Delta \vdash MN : B}\ (\Rightarrow E)$$

$$\frac{\Gamma \vdash M : A}{\Gamma \downarrow \alpha \vdash \Lambda\alpha.M : \forall\alpha.A}\ (\forall\ \ I)\quad (\alpha \notin \mathtt{FTV}(\Gamma))$$

$$\frac{\Gamma \vdash M : \forall\alpha.B}{\Gamma \vdash MA : B\langle\alpha := A\rangle}\ (\forall\ \ E)\quad (A \text{ is a type})$$

$$\frac{\Delta \vdash M : A \qquad \Gamma_i \vdash N_i : B_i}{\Delta - \{\overline{v^B}\}, \overline{\Gamma} \vdash M\langle\overline{v^B} := \overline{N}\rangle : A}\ (\mathrm{sub}_1)$$

$$\frac{\Gamma \vdash M : A}{\Gamma \downarrow \{\overline{\alpha}\} \vdash \{\overline{\alpha}\}.M : A^{\{\overline{\alpha}\}}}\ (\mathrm{abs\ I}_1)\quad (\{\overline{\alpha}\} \cap \mathtt{FTV}(\Gamma) = \emptyset)$$

$$\frac{\Gamma \vdash M : A^{\{\overline{\alpha}\}}}{\Gamma \vdash M \bullet [\langle\overline{\alpha} := \overline{C}\rangle] : A\langle\overline{\alpha} := \overline{C}\rangle}\ (\mathrm{abs\ E}_1)$$

$$\frac{\Gamma \vdash M : A}{\Gamma - \{\overline{x^B}\} \vdash \{\overline{x^B}\}.M : A^{\{\overline{x^B}\}}}\ (\mathrm{abs\ \ I}_2)$$

$$\frac{\Gamma \vdash M : A^{\{\overline{x^B}\}} \qquad \Delta_i \vdash N_i : B_i}{\Gamma, \overline{\Delta} \vdash M \bullet [\langle\overline{x^B} := \overline{N}\rangle] : A}\ (\mathrm{abs\ \ E}_2)$$

$$\frac{\Gamma \vdash M : A}{\Gamma\langle\overline{\nu} := \overline{B}\rangle \vdash M\langle\overline{\nu} := \overline{B}\rangle : A\langle\overline{\nu} := \overline{B}\rangle}\ (\mathrm{sub}_2)$$

$$\frac{\Gamma \vdash M : A \quad A \stackrel{*}{\rightarrow}_T B}{\Gamma \vdash M : B}\ (\mathrm{conv}_1)$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \stackrel{*}{\rightarrow}_T \Delta}{\Delta \vdash M : A}\ (\mathrm{conv}_2)$$

Note that in the (sub_2) rule, the explicit substitution $\langle\overline{\nu} := \overline{B}\rangle$ works not only on terms but also on types and on type assignments, and the rules (conv_1) and (conv_2) are used to reduce $A\langle\overline{\nu} := \overline{B}\rangle$ and $\Gamma\langle\overline{\nu} := \overline{B}\rangle$ to their T-normal forms. In the (abs E_2) rule, the type B_i of x_i in the explicit substitution should exactly match its type in M. This is one of the reasons that we need to attach information about the type of the variable to each of its occurrence.

Figure 2 shows a simple example of a derivation, for the sake of space limitation the inference rule names will be omitted.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{x^\alpha \vdash x^\alpha : \alpha}}{\vdash \{x^\alpha\}.x^\alpha : \alpha^{\{x^\alpha\}}}}{\vdash \{\alpha\}.\{x^\alpha\}.x^\alpha : \alpha^{\{x^\alpha\}\{\alpha\}}}}{\vdash (\{\alpha\}.\{x^\alpha\}.x^\alpha) \bullet [\langle \alpha := A\rangle] : (\alpha^{\{x^\alpha\}})\langle \alpha := A\rangle} \quad (\alpha^{\{x^\alpha\}})\langle \alpha := A\rangle \xrightarrow{*}_T A^{\{x^A\}}}{\vdash (\{\alpha\}.\{x^\alpha\}.x^\alpha) \bullet [\langle \alpha := A\rangle] : A^{\{x^A\}}} \quad \overline{z^A \vdash z^A : A}$$
$$z^A \vdash (\{\alpha\}.\{x^\alpha\}.x^\alpha) \bullet [\langle \alpha := A\rangle] \bullet [\langle x^A := z^A\rangle] : A$$

Figure 2: A simple example of derivation

3 The Reduction Rules

The $\lambda_{2\mathrm{xc}}$ *Reduction Rules* are the union of the following reduction rules:

1. The relation $\rightarrow_\lambda$ is defined by the following rule:
   (λ) $(\lambda x.M)N \rightarrow_\lambda M\langle x := N\rangle$.

2. The relation $\rightarrow_\Lambda$ is defined by the following rule:
   (Λ) $(\Lambda\alpha.M)A \rightarrow_\Lambda M\langle \alpha := A\rangle$.

3. The relation $\rightarrow_{\mathrm{c}}$ is defined by the following two rules:
   (c_1) $(\{\overline{\alpha}\}.M) \bullet [\langle \overline{\alpha} := \overline{C}\rangle] \rightarrow_{\mathrm{c}_1} M\langle \overline{\alpha} := \overline{C}\rangle$.
   (c_2) $(\{\overline{x^B}\}.M) \bullet [\langle \overline{x^B} := \overline{N}\rangle] \rightarrow_{\mathrm{c}_2} M\langle \overline{x^B} := \overline{N}\rangle$.

4. The relation $\rightarrow_{\mathrm{x}}$ (e*X*plicit substitution), which is necessary to evaluate terms of the form $M\langle \overline{v} := \overline{N}\rangle$, is defined by the following rules:

   (xvar) $v_i\langle \overline{v} := \overline{N}\rangle \rightarrow_{\mathrm{x}} N_i$.
   (gc) $M\langle \overline{v} := \overline{N}\rangle \rightarrow_{\mathrm{x}} M \downarrow \{\overline{v}\}$, (if $\{\overline{v}\} \cap \mathtt{FV}(M) = \emptyset$).
   (xabs_1) $(\lambda x.M)\langle \overline{v} := \overline{N}\rangle \rightarrow_{\mathrm{x}} \lambda x.M\langle \overline{v} \uparrow x := \overline{N} \uparrow \{x\}\rangle$.
   (xabs_2) $(\Lambda\alpha.M)\langle \overline{v^A} := \overline{N}\rangle \rightarrow_{\mathrm{x}} \Lambda\alpha.M\langle \overline{v^{(A\uparrow\{\alpha\})}} := \overline{N} \uparrow \{\alpha\}\rangle$.
   (xapp_1) $(M_1M_2)\langle \overline{v} := \overline{N}\rangle \rightarrow_{\mathrm{x}} (M_1\langle \overline{v} := \overline{N}\rangle)(M_2\langle \overline{v} := \overline{N}\rangle)$.
   (xapp_2) $(MA)\langle \overline{v} := \overline{N}\rangle \rightarrow_{\mathrm{x}} (M\langle \overline{v} := \overline{N}\rangle)A$.
   (xabs_3) $(\{\overline{\alpha}\}.M)\langle \overline{v^C} := \overline{N}\rangle \rightarrow_{\mathrm{x}} \{\overline{\alpha}\}.M\langle \overline{v^C} \uparrow \{\overline{\alpha}\} := \overline{N} \uparrow \{\overline{\alpha}\}\rangle$.
   (xabs_4) $(\{\overline{x^B}\}.M)\langle \overline{v^C} := \overline{N}\rangle \rightarrow_{\mathrm{x}} \{\overline{x^B}\}.(M\langle \overline{v^C} \uparrow \{\overline{x^B}\} := \overline{N} \uparrow \{\overline{x^B}\}\rangle)$.
   (xapp_3) $(M \bullet [\langle \overline{\alpha} := \overline{C}\rangle])\langle \overline{v^D} := \overline{P}\rangle \rightarrow_{\mathrm{x}} M\langle \overline{v^D} := \overline{P}\rangle \bullet [\langle \overline{\alpha} := \overline{C}\rangle]$.
   (xapp_4) $(M \bullet [\langle \overline{x} := \overline{N}\rangle])\langle \overline{v^C} := \overline{P}\rangle \rightarrow_{\mathrm{x}} M\langle \overline{v^C} := \overline{P}\rangle \bullet [\langle \overline{x} := \overline{N}\langle \overline{v^C} := \overline{P}\rangle\rangle]$.

5. The relation $\rightarrow_{\mathrm{xt}}$ (e*X*plicit *T*ype substitution), which is necessary to evaluate terms of the form $M\langle \overline{\nu} := \overline{B}\rangle$, is defined as:

   (xtvar) $v^A\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} v^{A\langle \overline{\nu}:=\overline{B}\rangle}$.
   (xtgc) $M\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} M \downarrow \{\overline{\nu}\}$, if $\{\overline{\nu}\} \cap \mathtt{FTV}(M) = \emptyset$.
   (xtabs_1) $(\lambda x^A.M)\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} \lambda x^{A\langle \overline{\nu}:=\overline{B}\rangle}.M\langle \overline{\nu} := \overline{B}\rangle$.
   (xtabs_2) $(\Lambda\beta.M)\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} \Lambda\beta.M\langle \overline{\nu} \uparrow \beta := \overline{B} \uparrow \{\beta\}\rangle$.
   (xtapp_1) $(MN)\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} (M\langle \overline{\nu} := \overline{B}\rangle)(N\langle \overline{\nu} := \overline{B}\rangle)$.
   (xtapp_2) $(MA)\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} (M\langle \overline{\nu} := \overline{B}\rangle)(A\langle \overline{\nu} := \overline{B}\rangle)$.
   (xtabs_3) $(\{\overline{\alpha}\}.M)\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} \{\overline{\alpha}\}.M\langle \overline{\nu} \uparrow \{\overline{\alpha}\} := \overline{B} \uparrow \{\overline{\alpha}\}\rangle$.
   (xtabs_4) $(\{\overline{x^A}\}.M)\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} \{\overline{x^{A\langle \overline{\nu}:=\overline{B}\rangle}}\}.(M\langle \overline{\nu} := \overline{B}\rangle)$.
   (xtapp_3) $(M \bullet [\langle \overline{\gamma} := \overline{C}\rangle])\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} M\langle \overline{\nu} := \overline{B}\rangle \bullet [\langle \overline{\gamma} := \overline{C}\langle \overline{\nu} := \overline{B}\rangle\rangle]$.
   (xtapp_4) $(M \bullet [\langle \overline{x^A} := \overline{N}\rangle])\langle \overline{\nu} := \overline{B}\rangle \rightarrow_{\mathrm{xt}} M\langle \overline{\nu} := \overline{B}\rangle \bullet [\langle \overline{x^{A\langle \overline{\nu}:=\overline{B}\rangle}} := \overline{N}\langle \overline{\nu} := \overline{B}\rangle\rangle]$.

6. The relation $\rightarrow_{\mathrm{tt}}$ (*T*erm-*T*ype reduction) is defined as:

   (*ttvar*) $v^A \rightarrow_{\mathrm{tt}} v^B$, if $A \rightarrow_T B$.
   (ttabs_1) $\lambda x^A.M \rightarrow_{\mathrm{tt}} \lambda x^B.M$, if $A \rightarrow_T B$.
   (ttsub_1) $M\langle \overline{v^A} := \overline{N}\rangle \rightarrow_{\mathrm{tt}} M\langle \overline{v^B} := \overline{N}\rangle$, if $\overline{A} \rightarrow_T \overline{B}$.
   (ttabs_2) $\{\overline{x^A}\}.M \rightarrow_{\mathrm{tt}} \{\overline{x^B}\}.M$, if $\overline{A} \rightarrow_T \overline{B}$.
   (ttapp_1) $M \bullet [\langle \overline{x^A} := \overline{N}\rangle] \rightarrow_{\mathrm{tt}} M \bullet [\langle \overline{x^B} := \overline{N}\rangle]$, if $\overline{A} \rightarrow_T \overline{B}$.

(ttapp_2) $MA \rightarrow_{\text{tt}} MB$, if $A \rightarrow_T B$.

(ttsub_2) $M\langle\overline{\alpha} := \overline{A}\rangle \rightarrow_{\text{tt}} M\langle\overline{\alpha} := \overline{B}\rangle$, if $\overline{A} \rightarrow_T \overline{B}$.

We write $M \rightarrow_\lambda N$ if N is obtained from M by replacing a subterm M' in M by N' s.t. $M' \rightarrow_\lambda N'$. Similarly the other reductions, $M \rightarrow_{\lambda_{2\text{xc}}} N$, $M \rightarrow_\Lambda N$, ... are defined. The refelxive and transitive closure of the reduction $\rightarrow$ is denoted by $\xrightarrow{*}$ and the transitive closure of the reduction $\rightarrow$ is denoted by $\xrightarrow{+}$. The reduction $\rightarrow_{\lambda,\Lambda,\text{c}}$ is the union of the 3-reduction relations $\rightarrow_\lambda$, $\rightarrow_\Lambda$ and $\rightarrow_{\text{c}}$.

In what follows we give the reason of restricting the term variables (type variables) bound by λ and by the set $\{\overline{x^B}\}$ (bound by Λ and by the set $\{\overline{\alpha}\}$, respectively) to pure variables.

Consider, for example, the term

$$(\lambda\#x^A.x + \#x)\langle x := y\rangle$$

in which the variables bound by λ is not pure, in this term the variables $\#x$ is bound by $\lambda\#x$ while x is bound by the variable in the explicit substitution.

We reduce this term as follows:

$\rightarrow_{\text{xabs}_1} \lambda\#x^A.(x + \#x)\langle x \uparrow \#x := y \uparrow \#x\rangle$

$\equiv \lambda\#x^A.(x + \#x)\langle x := y\rangle$

$\xrightarrow{*}_{\text{x}} \lambda\#x^A.y + x$

after this reduction the variable x is no longer bound by $\lambda\#x$ which is not correct. Another operator must be used in the rule (xabs_1) to *keep* the free occurrences of the variables after reduction. This operator–called the exchange operator– is defined in [10]. The situation is the same for the other binders and some other rules. We believe that restricting these variables to pure variables simplify the calculus and it does not affect our intended motivation about context.

Example 5 *Consider the following example:*

$(\lambda X^{\alpha^{\{x^\alpha\}\{\alpha\}}}.(\Lambda\alpha.(\lambda x^\alpha.(X \bullet [\langle\alpha := \alpha\rangle] \bullet [\langle x := x\rangle]))) Ky^K)(\{\alpha\}.\{x^\alpha\}.x^\alpha)$

The computation of this example is shown in figure 3.

4 Properties of $\lambda_{2\text{xc}}$

In this Section we show that our calculus has some desired properties. First, we show the subject reduction.

4.1 Subject Reduction

The following Lemma is necessary to prove Proposition 1.

Lemma 2 *If $A \xrightarrow{*}_T B$, then* $\texttt{FTV}(A) \supseteq \texttt{FTV}(B)$

Proof By induction on the derivation of $A \rightarrow_T B$. □

The following Proposition is necessary to prove the subject reduction property.

Proposition 1 Let $\Gamma \vdash N : B$, then:

1. $\texttt{FTV}(N) \supseteq \texttt{FTV}(\Gamma)$
2. $\texttt{FTV}(N) \supseteq \texttt{FTV}(B)$

Proof By induction on the derivation of $\Gamma \vdash N : B$ using lemma 2. □

Lemma 3 *(Type Substitution Lemma)*

If β is a pure type variable, then:

$C\langle\beta := A\rangle\langle\overline{\nu} := \overline{B}\rangle = C\langle\overline{\nu} \uparrow \beta := \overline{B} \uparrow \{\beta\}\rangle\langle\beta := A\langle\overline{\nu} := \overline{B}\rangle\rangle$

Where $A, \overline{B}, C$ are types, and $\beta, \overline{\nu}$ are type variables.

Proof

To prove this lemma, the following general lemma will be proved and the above one can be concluded by restricting μ to pure variables (if μ is a pure type variable then $(\mu \downarrow (\overline{\nu} \uparrow \mu)) = \mu$)

$C\langle\mu := A\rangle\langle\overline{\nu} := \overline{B}\rangle \equiv C\langle\overline{\nu} \uparrow \mu := \overline{B} \uparrow (\mu \downarrow (\overline{\nu} \uparrow \mu))\rangle\langle\mu \downarrow (\overline{\nu} \uparrow \mu) := A\langle\overline{\nu} := \overline{B}\rangle\rangle$

To prove this it is enough to prove the following property:

$C\langle\mu_1 := A_1\rangle...\langle\mu_n := A_n\rangle\langle\overline{\nu} := \overline{B}\rangle \equiv C\langle\overline{\nu} \uparrow \{\overline{\mu}\} := \overline{B} \uparrow (\mu_n \downarrow (\overline{\nu} \uparrow \mu_n))... \uparrow (\mu_1 \downarrow (\overline{\nu} \uparrow \overline{\mu}))\rangle\langle\mu_1 \downarrow (\overline{\nu} \uparrow \overline{\mu}) := A_1\langle\overline{\nu} \uparrow \overline{\mu_{n,2}} := \overline{B} \uparrow (\mu_n \downarrow (\overline{\nu} \uparrow \mu_n))... \uparrow (\mu_2 \downarrow (\overline{\nu} \uparrow \overline{\mu_{n,2}}))\rangle\rangle...\langle\mu_n \downarrow (\overline{\nu} \uparrow (\mu_n) := A_n\langle\overline{\nu} := \overline{B}\rangle\rangle$

for all types $A_1, ..., A_n, \overline{B}$ and all types C that do not end with a substitution. (So, one can take C as small as possible, i.e. C is a type variable, $C_1 \Rightarrow C_2, \forall\alpha.C_1, C_1^{\{\overline{\alpha}\}}$ or $C_1^{\{\overline{x^B}\}}$)

This can be proved by induction on the number of symbols in the sequence $A_1, ..., A_n$. □

Note that, when β is not pure type variable the last lemma does not hold. Consider the following counter example:

Let $C = \#\beta$, $n = 1$, $\nu_1 = \beta$, then:

LHS $\equiv \#\beta\langle\#\beta := A\rangle\langle\beta := B_1\rangle \equiv A\langle\beta := B_1\rangle$, while

RHS $\equiv \#\beta\langle\beta := B_1 \uparrow \{\#\beta\}\rangle\langle\#\beta := A\langle\beta := B_1\rangle\rangle \equiv \beta$.

Also, we cannot prove this lemma directly when β is a pure type variable, since in some types and during the computation β may be pushed through other type variables and it will no longer be pure and then the induction hypothesis cannot be used. This is the reason that we have to prove the general form of the lemma given above.

To show that the type is preserved by $\lambda_{2\text{xc}}$-reduction and the reductions does not introduce new free variables, we have the following result.

Theorem 1 (Subject Reduction)

If $\Gamma \vdash M : A$ and $M \xrightarrow{*}_{\lambda_{2\text{xc}}} N$, then for some Δ and for some B we have $\Delta \vdash N : B$ and $\Delta \subseteq \Gamma$, $A \xrightarrow{*}_T B$.

Proof By induction on the reduction $M \rightarrow_{\lambda_{2\text{xc}}} N$, using Propostion 1 and the type substitution lemma. □

$$
\begin{array}{ll}
\rightarrow_{\Lambda} & ((\lambda X.(\lambda x^{\alpha}.(X \bullet [\langle \alpha := \alpha \rangle] \bullet [\langle x := x \rangle]))\langle \alpha := K \rangle)y)(\{\alpha\}.\{x^{\alpha}\}.x) \\
\stackrel{*}{\rightarrow}_{\mathrm{xt,tt}} & ((\lambda X^{\alpha\{x^{\alpha}\}\{\alpha\}}.(\lambda x^{K}.X \bullet [\langle \alpha := K \rangle] \bullet [\langle x := x \rangle]))y^{K}(\{\alpha\}.\{x^{\alpha}\}.x^{\alpha})) \\
\rightarrow_{\lambda,\mathrm{x}} & (\lambda X.(X \bullet [\langle \alpha := K \rangle] \bullet [\langle x^{K} := y \rangle]))(\{\alpha\}.\{x^{\alpha}\}.x) \\
\rightarrow_{\lambda} & (X \bullet [\langle \alpha := K \rangle] \bullet [\langle x^{K} := y \rangle])\langle X := \{\alpha\}.\{x^{\alpha}\}.x \rangle \\
\stackrel{*}{\rightarrow}_{\mathrm{x}} & (\{\alpha\}.\{x^{\alpha}\}.x) \bullet [\langle \alpha := K \rangle] \bullet [\langle x^{K} := y \rangle] \\
\rightarrow_{c_1} & (\{x^{\alpha}\}.x^{\alpha})\langle \alpha := K \rangle \bullet [\langle x := y^{K} \rangle] \\
\stackrel{*}{\rightarrow}_{\mathrm{x}} & (\{x^{K}\}.x) \bullet [\langle x^{K} := y^{K} \rangle] \\
\rightarrow_{c2} & x\langle x := y \rangle \\
\rightarrow_{\mathrm{x}} & y^{K}
\end{array}
$$

Figure 3: A computation of example 4 in $\lambda_{2\mathrm{xc}}$

4.2 Confluence

The following two lemmas are important to prove the confluence property.

Lemma 4

The reductions $\rightarrow_{\mathrm{x}} \cup \rightarrow_{\mathrm{xt}} \cup \rightarrow_{tt}$ on $\lambda_{2\mathrm{xc}}$-terms are noetherian and confluent.

Proof Easy. □

A term M is said to be e-normal[2] if $M \rightarrow_{\mathsf{xUxtUtt}} N$ holds for no N. By the above lemma, it is clear that for any term M, there uniquely exists an e-normal term N s.t. $M \stackrel{*}{\rightarrow}_{\mathsf{xUxtUtt}} N$. We will write $\mathrm{e}(M)$ for this N.

Lemma 5 *(Term Substitution Lemma)*

1. If M is an e-normal $\lambda_{2\mathrm{xc}}$term and β is a pure type variable, then

   $\mathrm{e}(M\langle \overline{\beta} := \overline{A} \rangle \langle \overline{\alpha} := \overline{B} \rangle) \equiv \mathrm{e}(M\langle \overline{\alpha} \uparrow \{\overline{\beta}\} := \overline{B} \uparrow \{\overline{\beta}\} \rangle \langle \overline{\beta} := \overline{A}\langle \overline{\alpha} := \overline{B} \rangle \rangle)$

2. If M is an e-normal $\lambda_{2\mathrm{xc}}$term and y is a pure term variable then:

   $\mathrm{e}(M\langle \overline{y} := \overline{N} \rangle \langle \overline{w} := \overline{O} \rangle) \equiv \mathrm{e}(M\langle \overline{w} \uparrow \{\overline{y}\} := \overline{O} \uparrow \{\overline{y}\} \rangle \langle \overline{y} := \overline{N}\langle \overline{w} := \overline{O} \rangle \rangle)$

Proof By induction on the construction of M, using the same method defined in [11]. □

To show that our calculus is computationally well-behaved, we show the following result.

Theorem 2 (Confluence) For any well typed term M, if $M \stackrel{*}{\rightarrow}_{\lambda_{2\mathrm{xc}}} M_1$ and $M \stackrel{*}{\rightarrow}_{\lambda_{2\mathrm{xc}}} M_2$, then there is an M_3 such that $M_1 \stackrel{*}{\rightarrow}_{\lambda_{2\mathrm{xc}}} M_3$ and $M_2 \stackrel{*}{\rightarrow}_{\lambda_{2\mathrm{xc}}} M_3$.

Proof By using the technique of parallel reduction and lemma 4 and lemma 5. □

[2] e is for explicit substitution

4.3 Conservativity

In proving that $\lambda_{2\mathrm{xc}}$ is a conservative extension over the second order λ-calculus we need the following definitions:

Two terms are α-equivalent, written as $\equiv_{\alpha}$, if they are identical except for renaming of bound variables (type and term variables) bound only by λ, Λ and by the $\overline{v^A}$ and $\overline{\nu}$ in the terms $N\langle \overline{v^A} := \overline{M} \rangle$ and $M\langle \overline{\nu} := \overline{B} \rangle$ respectively .

If M, N are $\lambda_{2\mathrm{xc}}$-terms defined by the following grammar:

$$
\begin{array}{c}
A, B ::= K \mid TV \mid A \Rightarrow B \mid \forall \alpha.A \\
M, N ::= V^A \mid \lambda x^A.M \mid MN \mid \Lambda \alpha.M \mid MA
\end{array}
$$

then they can be regarded as λ_2-terms.

Theorem 3 (Conservativity over λ_2-Calculus)
If M, N are λ_2-terms, then $M \stackrel{*}{\rightarrow}_{\beta} N$ iff $M \stackrel{*}{\rightarrow}_{\lambda_{2\mathrm{xc}}} N'$ and $N \equiv_{\alpha} N'$ for some term N' in $\lambda_{2\mathrm{xc}}$.

Proof By induction on the construction of M. □

4.4 Strong Normalizability

A $\lambda_{2\mathrm{xc}}$-term M is called strongly normalizable iff all reduction sequences starting from M terminate.

Let $\lambda_{2\uparrow}$ be the second-order lambda calculus with the push and pull operators. Let λ_{2C} be the second-order lambda calculus with the push and pull operators and contexts (with meta-level substitution).

We need the following lemma to prove the strong normalizability of $\lambda_{2\mathrm{xc}}$.

Lemma 6

Let M, N be $\lambda_{2\mathrm{xc}}$-terms, then
if $M \rightarrow_{\lambda,\Lambda,\mathrm{c}} N$, then $\mathrm{e}(M) \stackrel{+}{\rightarrow}_{\lambda_{2C}} \mathrm{e}(N)$.
Proof By induction on the reduction $M \rightarrow_{\lambda,\Lambda,\mathrm{c}} N$. □

Next, with each $\lambda_{2\mathrm{xc}}$-type and term define the following association:

1. $\mathtt{Ass}(K) = K$.
2. $\mathtt{Ass}(\nu) = \nu$.

3. $\mathtt{Ass}(A \Rightarrow B) = \mathtt{Ass}(A) \Rightarrow \mathtt{Ass}(B)$.

4. $\mathtt{Ass}(B^{\{\overline{x^A}\}}) = \mathtt{Ass}(C_1) \Rightarrow \mathtt{Ass}(C_2) \Rightarrow ... \Rightarrow \mathtt{Ass}(C_n) \Rightarrow \mathtt{Ass}(B)$,
   where $y_1^{C_1}, ..., y_n^{C_n}$ are the total order permutation of $x_1^{A_1}, ..., x_n^{A_n}$ s.t. $y_1^{C_1} < ... < y_n^{C_n}$.

5. $\mathtt{Ass}(A^{\{\overline{\alpha}\}}) = \forall\beta_1, ..., \forall\beta_n.\mathtt{Ass}(A)$,
   where $\beta_1, ..., \beta_n$ are the permutation of $\alpha_1, ..., \alpha_n$ s.t. $\beta_1 < ... < \beta_n$.

6. $\mathtt{Ass}(\forall\alpha.A) = \forall\alpha.\mathtt{Ass}(A)$.

7. $\mathtt{Ass}(v^A) = v^{\mathtt{Ass}(A)}$.

8. $\mathtt{Ass}(\lambda x^A.M) = \lambda x^{\mathtt{Ass}(A)}.\mathtt{Ass}(M)$.

9. $\mathtt{Ass}(MN) = \mathtt{Ass}(M)\mathtt{Ass}(N)$.

10. $\mathtt{Ass}(\Lambda\alpha.M) = \Lambda\alpha.\mathtt{Ass}(M)$.

11. $\mathtt{Ass}(MA) = \mathtt{Ass}(M)\mathtt{Ass}(A)$.

12. $\mathtt{Ass}(\{\alpha_1, ..., \alpha_n\}.M) = \Lambda\beta_1...\Lambda\beta_n.\mathtt{Ass}(M)$,
   where $\beta_1, ..., \beta_n$ are the permutation of $\alpha_1...\alpha_n$ and $\beta_1 < ... < \beta_n$.

13. $\mathtt{Ass}(M \bullet [\langle\overline{\alpha} := \overline{C}\rangle]) = \mathtt{Ass}(M)\mathtt{Ass}(A_1)...\ \mathtt{Ass}(A_n)$,
   where $\beta_1 := A_1, ..., \beta_n := A_n$ are the permutation of $\alpha_1 := C_1, ..., \alpha_n := C_n$ s.t. $\beta_1 < ... < \beta_n$.

14. $\mathtt{Ass}(\{x_1^{A_1}, ..., x_n^{A_n}\}.M) = \lambda y_1^{\mathtt{Ass}(C_1)}...\lambda y_n^{\mathtt{Ass}(C_n)}.\mathtt{Ass}(M)$,
   where $y_1^{C_1}, ..., y_n^{C_n}$ are the total order permutation of $x_1, ..., x_n$ s.t. $y_1^{C_1} < ... < y_n^{C_n}$.

15. $\mathtt{Ass}(M \bullet [\langle x_1, ..., x_n := N_1, ..., N_n\rangle]) = \mathtt{Ass}(M)\ \mathtt{Ass}(L_1)...\mathtt{Ass}(L_n)$,
   where $y_1^{C_1} := L_1, ..., y_n^{C_n} := L_n$ are the total order permutation of $x_1, ..., x_n$ s.t. $y_1^{C_1} < ... < y_n^{C_n}$

The following lemma in needed to prove lemma 9.

Lemma 7

Let $\overline{\alpha}$ be distinct pure variables, $A, \overline{C}$ be $\lambda_{2\mathrm{xc}}$-types then:

$\mathtt{Ass}(A)[\overline{\alpha} := \mathtt{Ass}(\overline{C})] \equiv \mathtt{Ass}(A)[\beta_1 := \mathtt{Ass}(B_1) \uparrow \beta_2... \uparrow \beta_n]...[\beta_n := \mathtt{Ass}(B_n)]$,

where $\beta_1 := \mathtt{Ass}(B_1)...\beta_n := \mathtt{Ass}(B_n)$ are the total order permutation of $\alpha_1 := \mathtt{Ass}(C_1)...\alpha_n := \mathtt{Ass}(C_n)$ s.t. $\beta_1 < ... < \beta_n$.

Proof

In order to prove this lemma, we prove the following more general one:

$\mathtt{Ass}(A)[\overline{\nu} := \mathtt{Ass}(\overline{C})] \equiv \mathtt{Ass}(A)[\mu_1 := \mathtt{Ass}(B_1) \uparrow (\mu_2 \downarrow \mu_1)... \uparrow (\mu_n \downarrow \mu_{n-1} \downarrow ... \downarrow \mu_1)]...[\mu_n \downarrow \mu_{n-1} \downarrow ... \downarrow \mu_1 := \mathtt{Ass}(B_n)]$

where $\mu_1 := \mathtt{Ass}(B_1), ..., \mu_n := \mathtt{Ass}(B_n)$ are the total order permutation of $\alpha_1 := \mathtt{Ass}(C_1), ..., \alpha_n := \mathtt{Ass}(C_n)$ s.t. $\mu_1 < \mu_2... < \mu_n$.
by induction on the construction of A.

The needed formula can be deduced by observing that: if $\overline{\mu}$ are distinct pure varaibles, then:
$\mu_i \downarrow \mu_{i-1} = \mu_i$, for $i \in \{1, ..., n\}$. □

The following lemma is needed in the proof of lemma 10.

Lemma 8

1. Let M be a $\lambda_{2\mathrm{xc}}$ term, $\overline{C}$ be $\lambda_{2\mathrm{xc}}$-types and $\overline{\alpha}$ be distinct pure variables, then:
   $\mathtt{Ass}(M)[\overline{\alpha} := \mathtt{Ass}(\overline{C})] \equiv \mathtt{Ass}(M)[\beta_1 := \mathtt{Ass}(A_1) \uparrow \beta_2... \uparrow \beta_n]...[\beta_n := \mathtt{Ass}(A_n)]$
   where $\beta_1 := A_1, ..., \beta_n := A_n$ are the total order permutation of $\alpha_1 := C_1, ..., \alpha_n := C_n$ s.t. $\beta_1 < ... < \beta_n$.

2. Let $M, \overline{N}$ be $\lambda_{2\mathrm{xc}}$-terms and $\overline{x}$ be distint pure variables, then:
   $\mathtt{Ass}(M)[\overline{x} := \mathtt{Ass}(\overline{N})] \equiv \mathtt{Ass}(M)[y_1^{\mathtt{Ass}(C_1)} := \mathtt{Ass}(L_1) \uparrow y_2... \uparrow y_n]...[y_n^{\mathtt{Ass}(C_n)} := \mathtt{Ass}(A_n)]$
   where $y_1 := L_1, ..., y_n := L_n$ are the total order permutation of $x_1 := N_1...x_n := N_n$ s.t. $y_1 < ... < y_n$.

proof Similar to lemma 7. □

To show that the association defined above preserves typability, we have the following lemma.

Lemma 9

If $\Gamma \vdash_{\lambda_{2C}} M : A$ then $\mathtt{Ass}(\Gamma) \vdash_{\lambda_{2\uparrow}} \mathtt{Ass}(M) : \mathtt{Ass}(A)$.

Proof By induction on the derivation $\Gamma \vdash_{\lambda_{2C}} M : A$ using lemma 7. □

The following lemma shows that there is a correspondence between the reductions in λ_{2C} and that of the associated terms in $\lambda_{2\uparrow}$.

Lemma 10

If $M \rightarrow_{\lambda_{2C}} N$, then $\mathtt{Ass}(M) \rightarrow_{\lambda_{2\uparrow}} \mathtt{Ass}(N)$.
Proof By induction on the reduction $M \rightarrow_{\lambda_{2C}} N$ using lemma 8. □

From lemma 9 and lemma 10 and since $\lambda_{2\uparrow}$ is strongly normalizable, we have the strong normalizability of λ_{2C}.

Corollary 1 (Strong Normalizability)
If $\Gamma \vdash M : A$, then M is strongly normalizable.
Proof
Assume there is an infinite reduction sequence starting from M using $\lambda_{2\mathrm{xc}}$-reductions.

Since the reduction $\rightarrow_{\mathrm{x}} \cup \rightarrow_{\mathrm{xt}} \cup \rightarrow_{\mathrm{tt}}$ are noetherian, then there is an infinite reduction sequence using $\rightarrow_{\lambda,\Lambda,\mathrm{c}}$ reductions. Then there is M_i s.t. $M \overset{*}{\rightarrow}_{\lambda_{2\mathrm{xc}}} M_i$ and

$M_i \rightarrow_{\lambda,\Lambda,c} M_{i+1} \ldots$

From lemma 6, there is also an infinite reduction sequence starting from $e(M_i)$ using λ_{2C} reductions with meta-level substitution, which is a contradiction since the λ_{2C} reduction sequence is strongly normalizable. Then M must also be strongly normalizable. □

5 Conclusion

We have introduced a second-order context calculus which has contexts as first-class values and hole-filling as an explicit operation. In our calculus, the usual data types can be represented and we can define a generic function with holes, in which both the term variable and the type variable can get bound after filling in the hole.

We have shown that our calculus is confluent, conservative over λ_2-calculus, satisfying the strong normalization property and enjoys the subject reduction property.

We have also introduced a new definition for the push operator, which is more natural and general than the ordinary one, and then it can be used in any calculus with different terms.

We conclude the paper by making the following comparison:

The way we represent contexts in our calculus is very close to that of Hashimoto-Ohori [5] in that the context in both calculi is a first-class object and hole-filling is an explicit operation. However, there are two points which we differ from Hashimoto-Ohori.

1. While they cannot reduce a β-redex which contains a hole, we have no such restriction, since we accompany the hole variable with an explicit substitution which is used to keep the value of the variable that would otherwise be lost.

2. To tell which variables should be bound after filling in the hole, they annotate both the hole variable and the fill operator with variable renamer, while in our calculus we introduce the term constructor $\{\overline{x^B}\}.M$ which tells that the set of variables $\{\overline{x^B}\}$ should be bound in M, and instead of renaming bound variables we renamed free variables by using #.

References

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

[2] R. Bloo, Preservation of Termination for Explicit Substitution, Ph.D thesis, Eindhoven University of Technology, 1997.

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

[4] J. Garrigue, H. Ait-Kaci, The Typed Polymorphic Label-Selective Lambda-Calculus, In *Proceeding of the 21th ACM Symposium on Principles of Programming Languages*, pp. 35-47, 1994.

[5] M. Hashimoto, A. Ohori , A Typed Context Calculus, to appear in *Theoretical Computer Science*

[6] S. Ikeda, An Extension of the Second-Order Type System F with Explicit Environments, Master thesis, Dept. of Information science, Kyoto University, 1999 (in Japanese).

[7] S. Lee and D. Friedman, Enriching the Lambda Calculus with Contexts: Towards a Theory of Incremental Program Construction. In *Proceedings of International Conference on Functional Programming*, ACM SIGPLAN notices, pp. 239:250, 1996.

[8] I. Mason, Computing with Contexts, *Higher-Order and Symbolic Computation* 12(2):171-201, September 1999.

[9] D. Sands, Computing with Contexts-a Simple Approach, *Electronic Notes in Theoretical Computer Science* 10, 1998.

[10] M. Sato, T. Sakurai and Y. Kameyama, A Simply Typed Context Calculus with First-Class Environments, Proc. Fifth International Symposium on Functional and Logic Programming, Lecture Notes in Computer Science, to appear.

[11] A. A. Taha, M. Sato, Y. Kameyama, A Type-Free Context Calculus, Journal of Information Processing Society, Japan, 42(1):38-47, 2001.

A Appendix: The Encoding of λ_{2xc} Types and Terms in the General Syntax for a Term

In this appendix, we show how the λ_{2xc}-types and terms can be encoded in the general syntax of terms defined in Section 2.3.

Definition 1

The λ_{2xc}-types can be encoded as follows:

$$
\begin{array}{lcl}
x^B & \equiv & (\mathsf{power}, x, B). \\
A \Rightarrow B & \equiv & (\mathsf{arrowtype}, A, B). \\
\forall \alpha.A & \equiv & (\mathsf{all}, \{\alpha\}[A]). \\
A\langle \overline{\nu} := \overline{B} \rangle & \equiv & (\mathsf{apply}, \{\overline{\nu}\}[A], B). \\
A^{\{\overline{\alpha}\}} & \equiv & (\mathsf{abstype}, \{\overline{\alpha}\}[A]). \\
A^{\{\overline{x^B}\}} & \equiv & (\mathsf{apptype}, A, \overline{x^B}).
\end{array}
$$

The general syntax of the term given in Section 2.3 is equivalent to the following:

$$\begin{array}{lll} \mathsf{Var} & ::= & \mathsf{TV} \\ \mathsf{Const} & ::= & \mathsf{K} \mid \mathsf{power} \mid \mathsf{arrowtype} \\ & & \mid \mathsf{all} \mid \mathsf{apply} \mid \mathsf{abstype} \\ & & \mid \mathsf{apptype} \mid \#^n x \mid ... \\ \mathsf{StructuredTerm} & ::= & x^B \mid A \Rightarrow B \mid \forall \alpha.A \\ & & \mid A\langle \overline{\nu} := \overline{B} \rangle \mid A^{\{\overline{\alpha}\}} \\ & & \mid A^{\{\overline{x^B}\}} \\ \mathsf{Abstraction} & ::= & \{\overline{\nu}\}[A] \end{array}$$

By this encoding the definitions of $\mathtt{FTV}(A)$, pushing a $\lambda_{2\mathrm{xc}}$-type through a finite set of type variables H, written as $A \uparrow H$, and pulling a $\lambda_{2\mathrm{xc}}$-type from a finite set of type variables, written as $A \downarrow H$, can be deduced from the corresponding definitions in Section 2.3.

Example 6

1. $\mathtt{FTV}(\forall\alpha.(\alpha \Rightarrow \#\alpha))$
   $= \mathtt{FTV}(\mathsf{all}, \{\alpha\}[(\mathsf{arrowtype}, \alpha, \#\alpha)])$
   $= \mathtt{FTV}(\mathsf{all}) \circ \mathtt{FTV}(\{\alpha\}[(\mathsf{arrowtype}, \alpha, \#\alpha)])$
   $= \mathtt{FTV}(\mathsf{arrowtype}, \alpha, \#\alpha) - \{\alpha\}$
   $= \mathtt{FTV}(\mathsf{arrowtype}) \circ \mathtt{FTV}(\alpha) \circ \mathtt{FTV}(\#\alpha) - \{\alpha\}$
   $= (\alpha, \#\alpha) - \{\alpha\} = \alpha.$
2. $\#\alpha \uparrow \{\alpha, \#^2\alpha\} = \#^3\alpha.$
3. $(\forall\alpha.(\alpha \Rightarrow \#\alpha)) \uparrow \alpha$
   $= (\mathsf{all}, \{\alpha\}[(\mathsf{arrowtype}, \alpha, \#\alpha)]) \uparrow \alpha$
   $= (\mathsf{all} \uparrow \alpha, \{\alpha\}[(\mathsf{arrowtype} \uparrow_\alpha \alpha, \alpha \uparrow_\alpha \alpha, \#\alpha \uparrow_\alpha \alpha)])$
   $= (\mathsf{all}, \{\alpha\}[(\mathsf{arrowtype}, \alpha, \#^2\alpha)]) = \forall\alpha(\alpha \Rightarrow \#^2\alpha)$

Definition 2

The definitions of $\mathtt{FV}(M)$, pushing a $\lambda_{2\mathrm{xc}}$-term through a finite set of term variable G, written as $M \uparrow G$, and the definition of pulling a $\lambda_{2\mathrm{xc}}$-term from a finite set of term variable, written as $M \downarrow G$, can be deduced from the corresponding definitions in Section 2.3 by making the following encoding:

$$\begin{array}{lcl} \#^n x^B & \equiv & (\mathsf{power}, \#^n x, B),\ n \geq 0. \\ \lambda x^A.M & \equiv & (\mathsf{lambda}, \{x\}[M], A). \\ MN & \equiv & (\mathsf{apply}_1, M, N). \\ M\langle \overline{v^B} := \overline{N} \rangle & \equiv & (\mathsf{apply}_2, \{\overline{v}\}[M], \overline{B}, \overline{N}). \\ \{\overline{\alpha}\}.M & \equiv & (\mathsf{apply}_3, \overline{\alpha}, M). \\ \{\overline{x^B}\}.M & \equiv & (\mathsf{absterm}, \{\overline{x}\}[M], \overline{B}). \\ M \bullet [\langle \overline{\alpha} := \overline{C} \rangle] & \equiv & (\mathsf{apply}_4, M, \overline{\alpha}, \overline{C}). \\ M \bullet [\langle \overline{x^B} := \overline{N} \rangle] & \equiv & (\mathsf{apply}_5, M, \overline{N}, \{\overline{x}\}[\overline{B}]). \\ \Lambda\alpha.M & \equiv & (\mathsf{Lambda}, \alpha\ \mathsf{M}). \\ MA & \equiv & (\mathsf{apply}_6, M, A). \\ M\langle \overline{\#^n\alpha} := \overline{B} \rangle & \equiv & (\mathsf{apply}_7, M, \overline{\#^n\alpha}, \overline{B}). \end{array}$$

Note that when we push $\lambda_{2\mathrm{xc}}$-term through term variables $(x, y, z, ...)$, the types $\#^n\alpha, ...A, B, ...$ are considered as constants.

The general syntax of a term defined in Section 2.3 is equivalent to the following:

$$\begin{array}{lll} \mathsf{Var} & ::= & V^A \\ \mathsf{Const} & ::= & \#^n\alpha \mid \#^m\beta \mid ... \mid A \mid ... \\ & & \mid \mathsf{power} \mid \mathsf{lambda} \mid \mathsf{apply}_1 \\ & & \mid \mathsf{apply}_2 \mid \mathsf{apply}_3 \mid \mathsf{absterm} \\ & & \mid \mathsf{apply}_4 \mid \mathsf{apply}_5 \mid \mathsf{Lambda} \\ & & \mid \mathsf{apply}_6 \mid \mathsf{apply}_7 \\ \mathsf{Abstraction} & ::= & \{\overline{\#^n x^B}\}[M] \end{array}$$

Note that, all the terms here are considered as a structured term.

Example 7

1. $\#^2 x^A \uparrow \{\#^2 x, \#^3 x\}$
   $= (\mathsf{power}, \#^2 x, A) \uparrow \{\#^2 x, \#^3 x\}$
   $= (\mathsf{power} \uparrow \{\#^2 x, \#^3 x\}, \#^2 x \uparrow \{\#^2 x, \#^3 x\}, A \uparrow \{\#^2 x, \#^3 x\})$
   $= (\mathsf{power}, \#^4 x, A) = \#^4 x^A.$
2. $(\lambda x^A.x^A) \uparrow x$
   $= (\mathsf{lambda}, \{x\}[(\mathsf{power}, x, A)], A) \uparrow x$
   $= (\mathsf{lambda} \uparrow x, \{x\}[(\mathsf{power} \uparrow_x x, x \uparrow_x x, A \uparrow_x x)], A \uparrow x)$
   $= (\mathsf{lambda}, \{x\}[(\mathsf{power}, x, A)], A) = \lambda x^A.x^A$

Definition 3

The definitions of $\mathtt{FTV}(M)$, pushing a $\lambda_{2\mathrm{xc}}$-term through a finite set of type variables H, written as $M \uparrow H$, and pulling a $\lambda_{2\mathrm{xc}}$-term from a finite set of type variables, written as $M \downarrow H$, can be deduced from the corresponding definitions in Section 2.3 by making the following encoding:

$$\begin{array}{lcl} \#^n x^A & \equiv & (\mathsf{power}, \#^n x, A),\ n \geq 0. \\ \lambda x^A.M & \equiv & (\mathsf{lambda}, x^A, M). \\ MN & \equiv & (\mathsf{apply}_1, M, N). \\ M\langle \overline{\#^n x^A} := \overline{N} \rangle & \equiv & (\mathsf{apply}_2, M, \overline{\#^n x^A}, \overline{N}). \\ \{\overline{\alpha}\}.M & \equiv & (\mathsf{abstype}, \{\overline{\alpha}\}[M]). \\ \{\overline{x^B}\}.M & \equiv & (\mathsf{apply}_3, \overline{x^B}, M). \\ M \bullet [\langle \overline{x^B} := \overline{N} \rangle] & \equiv & (\mathsf{apply}_4, M, \overline{x^B}, \overline{N}). \\ M \bullet [\langle \overline{\alpha} := \overline{C} \rangle] & \equiv & (\mathsf{apply}_5, M, \overline{C}, \{\overline{\alpha}\}[\overline{\alpha}]) \\ \Lambda\alpha.M & \equiv & (\mathsf{Lambda}, \{\alpha\}[M]). \\ MA & \equiv & (\mathsf{apply}_6, M, A). \\ M\langle \overline{\#^n\alpha} := \overline{B} \rangle & \equiv & (\mathsf{apply}_7, \{\overline{\#^n\alpha}\}[M], \overline{B}) \end{array}$$

Note that, when we push $\lambda_{2\mathrm{xc}}$-term through type variables $(\alpha, \beta, ...)$ the term variables $\#^n x, \#^m y, ...$ are considered as constants. Since the notions of constants, variables, structured terms and abstractions are not the same in case of pushing $\lambda_{2\mathrm{xc}}$-terms throught term variables and

in case of pushing $\lambda_{2\mathrm{xc}}$-terms through type variables, we have to make different encoding one for each case.

The general syntax of a term is equivalent to the following:

Var	::=	TV
Const	::=	power | lambda | apply_1
		| apply_2 | abstype | apply_3
		| apply_4 |apply_5 | Lambda
		| apply_6 | apply_7 | $\#^n x...$
Abstraction	::=	$\{\overline{\#^n \alpha}\}[M]$

Note that, all the terms here are considered as a structured term.

Example 8

1. $\mathtt{FTV}(\{\alpha\}.x^{\alpha}) = \mathtt{FTV}(\mathsf{abstype}, \{\alpha\}[(\mathsf{power}, x, \alpha)])$
   $= \mathtt{FTV}(\mathsf{abstype}) \circ \mathtt{FTV}(\{\alpha\}[(\mathsf{power}, x, \alpha)])$
   $= \mathtt{FTV}(\mathsf{power}, x, \alpha) - \{\alpha\} = (\alpha) - \{\alpha\} = ()$
2. $(\#^2 x^{\alpha}) \uparrow \{\alpha, \#\alpha, \#^2\alpha\}$
   $= (\mathsf{power}, \#^2 x, \alpha) \uparrow \{\alpha, \#\alpha, \#^2\alpha\}$
   $= (\mathsf{power}, \#^2 x, \alpha \uparrow \{\alpha, \#\alpha, \#^2\alpha\})$
   $= (\mathsf{power}, \#^2 x, \#^3\alpha) = \#^2 x^{\#^3\alpha}$
3. $(\lambda x^{\alpha}.x^{\alpha}) \uparrow \alpha$
   $= (\mathsf{lambda}, (\mathsf{power}, x, \alpha), (\mathsf{power}, x, \alpha)) \uparrow \alpha$
   $= (\mathsf{lambda}, (\mathsf{power}, x, \alpha \uparrow \alpha), (\mathsf{power}, x, \alpha \uparrow \alpha))$
   $= (\mathsf{lambda}, (\mathsf{power}, x, \#\alpha), (\mathsf{power}, x, \#\alpha))$
   $= \lambda x^{\#\alpha}.x^{\#\alpha}.$