λf-calculus

– a Functional Calculus with Shared Environments –

Nobuko Yoshida
yoshida@mt.cs.keio.ac.jp

Kohei Honda
kohei@mt.cs.keio.ac.jp

Mario Tokoro
mario@mt.cs.keio.ac.jp

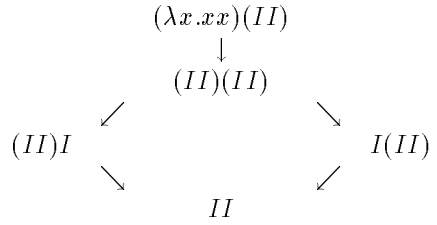
Department of Computer Science, Keio University

Abstract

This paper gives basic definitions of λf-calculus. λf-calculus is a variant of λ-calculus which represents functional computation with shared environments. The formalism has a Church Rosser property and enjoys the universal computing power so that it can be regarded as a self-contained functional calculus. Basic construction of the formalism and some of main results are presented together with suggestions for its possible applications.

1 Introduction

The traditional λ-calculus has no concept of sharing. Consider the following reduction on λ-terms, with $I \equiv \lambda x.x$.



If the argument (II) is substituted for the function $\lambda x.xx$, the reduction yields two copies of (II). In the next step the left (II) or the right one is contracted, then each sequence must be reduced to either $(II)I$ or $I(II)$. But if we share this argument (II), we can go straight from $(II)(II)$ to II without the intermediate states. In the following, we introduce λf-calculus which is a variant of λ-calculus with emphasis on sharing. One important feature of this formalism is to perform β-reduction with shared environments instead of instantiations of terms. Several other formalisms which explicitly manipulate environments have been proposed in [1][3][8]. They all use de Bruijn notation and copy environments. Our construction is based on *weak*-λ-calculus (which does not evaluate under λ's), and executes reductions without copying applications and environments in a clean and tractable formulation. The framework of computing is quite different from that of usual λ-calculus, while one can show that the calculus has essential properties as a functional calculus.

For the lack of space we will omit the most of proofs; for them, the reader can refer to the full version of the paper ([10]). Some notable conventions: $\Rightarrow$ is implication, ϕ is an empty set and $\stackrel{\text{def}}{=}$ is definitional equality.

2 Syntax

The main focus in the syntax of λf-calculus is an *environment*, which also serves as *shared resources*.

Definition 2.1 *Abstract Syntax.* The set of λ_f-terms $\mathcal{L}_f$ is given by the following abstract syntax, where $\mathcal{V}$ is the set of variables.

$$\mathcal{L}_f \quad = \quad \mathcal{V} \quad | \quad \mathcal{L}_f\mathcal{L}_f \quad | \quad \lambda\mathcal{V}.\mathcal{L}_f \quad | \quad \mathcal{L}_f[\mathcal{V} := \mathcal{L}_f]$$

$[x := L]$ is called an *environment*. ∎

Unless otherwise stated, capital letters stand for arbitrary λf-terms, non-capital letters for arbitrary variables. We write $M\{^N/_x\}$ for the substitution of a term N for the free occurrences of x in M. Note that the $\mathcal{L}_f$ clearly includes usual λ-terms.

The environment constructs associate to left.

$$(\cdots((N[x_1 := L_1])[x_2 := L_2])\cdots[x_i := L_i])$$

The expression $M[x := N]$ associates stronger than MN and $\lambda x.M$. Hence, $MN[x := L] \stackrel{\text{def}}{=} M(N[x := L])$.

Definition 2.2 *Binding.* The set of free variables of M, denoted by $\mathcal{FV}(M)$, is defined inductively as follows:

$$\begin{array}{lcl} \mathcal{FV}(x) & = & \{x\} \\ \mathcal{FV}(\lambda x.L) & = & \mathcal{FV}(L) - \{x\} \\ \mathcal{FV}(L_1[x := L_2]) & = & (\mathcal{FV}(L_1) - \{x\}) \cup \mathcal{FV}(L_2) \\ \mathcal{FV}(L_1 L_2) & = & \mathcal{FV}(L_1) \cup \mathcal{FV}(L_2) \end{array}$$ ∎

In addition to the original substitution rules of classical λ-calculus (Definition 1.11 in [4]), there are three rules for a term with an environment.

$$L_1[y := L_2]\{^n/_x\} \stackrel{\text{def}}{=} L_1\{^n/_x\}[y := L_2\{^n/_x\}] \quad (x \neq y \wedge (\, y \neq n \vee x \notin \mathcal{FV}(L_1)\,))$$

$$L_1[n := L_2]\{^n/_x\} \stackrel{\text{def}}{=} L_1\{^z/_n\}\{^n/_x\}[z := L_2\{^n/_x\}] \quad (x \neq n \wedge x \in \mathcal{FV}(L_1) \wedge z \notin \mathcal{FV}(L_1) \cup \{n\})$$

$$L_1[x := L_2]\{^n/_x\} \stackrel{\text{def}}{=} L_1[x := L_2\{^n/_x\}]$$

Then α-convertibility is defined as follows.

$$\begin{array}{rcl} \lambda x.L & \equiv_\alpha & \lambda y.L\{^y/_x\} \\ L[x := L_2] & \equiv_\alpha & L\{^y/_x\}[y := L_2] \end{array}$$

where $y \notin \mathcal{FV}(L)$. Now we introduce structure rules which make formulation of reduction rules concise and tractable.

Definition 2.3 *Structural Congruence.* Let $y \notin \mathcal{FV}(L_1, L_2)$ and $x \notin \mathcal{FV}(L, L_2)$. The structural congruence, denoted by $\equiv$, is the smallest congruence relation over terms defined by:

$$\begin{array}{c} P \equiv_\alpha Q \Rightarrow P \equiv Q \\ L_1[x := L]L_2 \equiv (L_1 L_2)[x := L] \\ L[y := L_1[x := L_2]] \equiv L[y := L_1][x := L_2] \end{array}$$ ∎

The next lemma says all λf-terms can be transformed into *σ-normal form* by $\equiv$.

Proposition 2.4 For any $M \in \mathcal{L}_f$, we have $M \equiv N$ such that, with some $n \geq 0$,

$$N \stackrel{\text{def}}{=} M_0[x_1 := M_1][x_2 := M_2] \cdot \cdot [x_n := M_n]$$

- M_i includes no expressions with a form $[x := M']$ as its subterms.
- x_1, x_2, ... x_n are all distinct.
- $\mathcal{FV}(M_i) \cap \mathcal{BV}(M_k) = \emptyset$, for any i, k s.t. $0 \leq i \leq n$ and $1 \leq k \leq n$.

We call such N a *σ-normal form.* ∎

Also the following notation is convenient for later developments.

Notation 2.5 Let σ, $\tilde{\sigma}$ and ε stand for an environment, a sequence of environments, and null string, respectively. Then we have

$$\begin{array}{lcl} M\varepsilon & \stackrel{\text{def}}{=} & M \\ M(\tilde{\sigma} \cdot \sigma') & \stackrel{\text{def}}{=} & (M\tilde{\sigma})\sigma' \end{array}$$ ∎

To accomplish reduction under sharing of terms with explicit environments, we now define rules corresponding to the β-rule of usual λ-calculus.

3 Reduction

Definition 3.1 *Reductions.*
Reductions on $\mathcal{L}_f$, denoted $\xrightarrow{\text{fw}}$ and $\xrightarrow{\beta_f}$, are the smallest relations inferred by rules in Fig.1 ($\xrightarrow{f}$ denotes $\xrightarrow{\text{fw}}$ or $\xrightarrow{\beta_f}$). ∎

Note reduction under λ's is not allowed in this formalism. β1-rule, β2-rule, and FW-rule[1] are the basic rules, while others induce reduction under composition.

Examples 3.2 *Reductions.*
$(\lambda x.xx)(II) \xrightarrow{\beta_f} (xx)[x := II] \xrightarrow{\beta_f} (xx)[x := z][z := I] \xrightarrow{\text{fw}} (zx)[x := z][z := I] \xrightarrow{\beta_f} y[y := x][x := z][z := I]$, corresponding to $(\lambda x.xx)(II) \rightarrow_\beta (II)(II) \rightarrow_\beta II \rightarrow_\beta I$ in λ-calculus. The argument II is shared automatically. ∎

λf-calculus has a one-step diamond property[10], which is particularly interesting when we consider that λf-calculus does not reduce under λ-body (cf. classical λ-calculus).

Proposition 3.3 *Diamond Property on $\xrightarrow{f}$.*

$$\forall M,\ M_1, M_2 \in \mathcal{L}_f, M \xrightarrow{f} M_1 \bigwedge M \xrightarrow{f} M_2 \Rightarrow \exists M' \in \mathcal{L}_f,\ M_1 \xrightarrow{f} M' \bigwedge M_2 \xrightarrow{f} M'$$ ∎

[1] This name comes from FORWARDER's behavior in an asynchronous process calculus called ν-calculus[5][6] such that $\leftarrow av,\ \mathcal{FW}(ab) \rightarrow \leftarrow bv,\ \mathcal{FW}(ab)$.

$$\langle \mathrm{FW} \rangle \quad (xN)[x := y] \xrightarrow{\mathrm{fw}} (yN)[x := y]$$

$$\langle \beta 1 \rangle \quad (\lambda x.M)N \xrightarrow{\beta_f} M[x := N] \quad (x \notin \mathcal{FV}(N))$$

$$\langle \beta 2 \rangle \quad (xN)[x := \lambda y.M] \xrightarrow{\beta_f} M[y := N][x := \lambda y.M] \quad (x \notin \mathcal{FV}(M))$$

$$\langle \mathrm{Ap(l)} \rangle \quad \frac{M \xrightarrow{f} M'}{MN \xrightarrow{f} M'N} \qquad \langle \mathrm{Ap(r)} \rangle \quad \frac{N \xrightarrow{f} N'}{MN \xrightarrow{f} MN'}$$

$$\langle \sigma(\mathrm{l}) \rangle \quad \frac{M \xrightarrow{f} M'}{M[x := N] \xrightarrow{f} M'[x := N]} \qquad \langle \sigma(\mathrm{r}) \rangle \quad \frac{N \xrightarrow{f} N'}{M[x := N] \xrightarrow{f} M[x := N']}$$

$$\langle \text{Ap-}\sigma(\mathrm{l}) \rangle \quad \frac{M_1[x := N] \xrightarrow{f} M_1'[x := N]}{(M_1 M_2)[x := N] \xrightarrow{f} (M_1' M_2)[x := N]}$$

$$\langle \text{Ap-}\sigma(\mathrm{r}) \rangle \quad \frac{M_2[x := N] \xrightarrow{f} M_2'[x := N]}{(M_1 M_2)[x := N] \xrightarrow{f} (M_1 M_2')[x := N]}$$

$$\langle \sigma\sigma(\mathrm{l}) \rangle \quad \frac{M[x := N_2] \xrightarrow{f} M'[x := N_2]}{M[y := N_1][x := N_2] \xrightarrow{f} M'[y := N_1][x := N_2]}$$

$$\langle \sigma\sigma(\mathrm{r}) \rangle \quad \frac{N_1[x := N_2] \xrightarrow{f} N_1'[x := N_2]}{M[y := N_1][x := N_2] \xrightarrow{f} M[y := N_1'][x := N_2]}$$

Figure 1: Reduction Rules

Outline of proof By induction on $\xrightarrow{f}$. Assume, e.g. $M \equiv (xN)[x := \lambda y.N_2]$. There are two reductions: $M \xrightarrow{f} (xN')[x := \lambda y.N_2] \stackrel{\mathrm{def}}{=} M_1$ by Ap-σ(r)-rule, or $M \xrightarrow{f} N_2[y := N][x := \lambda y.N_2] \stackrel{\mathrm{def}}{=} M_2$ by $\beta 2$-rule. Then $M_1 \xrightarrow{f} N_2[y := N'][x := \lambda y.N_2] \stackrel{\mathrm{def}}{=} M'$ by $\beta 2$-rule. But $M_2 \xrightarrow{f} M'$ too, using $\sigma\sigma$(r)-rule. Other eight cases are similar (in case $M \equiv (N_1 N_2)[x := N_3]$, the fact $M \not\xrightarrow{f} M' \wedge M[x := N] \xrightarrow{f} M'[x := N] \Rightarrow N \not\xrightarrow{f}$, is essential). ∎

It is immediate from this fact that:

Theorem 3.4 *The Church Rosser theorem holds for $\xrightarrow{f}$, which is the reflective and transitive closure of $\xrightarrow{f}$.* ∎

4 Convergence

We consider the reduction below.

$K I \Omega \xrightarrow{\beta_f} (\lambda y.x)\Omega[x := I] \xrightarrow{\beta_f} x[y := \Omega][x := I]$, with $K \equiv \lambda xy.x$ and $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$. The useless environment $[y := \Omega]$ remains and causes the infinite reductions independently. This is in contrast to λ-calculus; $KI\Omega \to_\beta (\lambda y.I)\Omega \to_\beta I$. Thus only strong normalizing terms terminate from the standpoint of λ-calculus. But it is more natural to regard ineffective environments as *garbages* in the same way as in graph reduction where terms not referred by the root-node are garbage-collected.

Definition 4.1 Assume $L \equiv M_0\tilde{\sigma}$ (a σ-normal form) and $\sigma_i \stackrel{\mathrm{def}}{=} [x_i := M_i]$. Garbage collection on $L \in \mathcal{L}_f$, denoted by $\gg_f$, is defined as follows:

$$x_k \notin \mathcal{FV}(M_0(\sigma_1 \cdot\cdot \sigma_{k-1})) \quad \Rightarrow \quad L \gg_f L' \equiv M_0(\sigma_1 \cdot\cdot \sigma_{k-1} \cdot \sigma_{k+1} \cdot\cdot \sigma_n) \quad ∎$$

Now we define convergence of λf-calculus. The notation $M \downarrow N$ is read "M halts with the answer N". We write $M \downarrow$ when there is some N with

$M \downarrow N$, and $M \uparrow$ otherwise. It is easy to see that $\gg_f$ is Church Rosser and strong normalizing.

Definition 4.2 *Convergence.*

$$N \downarrow_f M \quad \text{if} \quad N \xrightarrow{f}{}^{*} \gg_f^{*} M \not\gg_f \ \wedge\ M \not\xrightarrow{f} \qquad \blacksquare$$

Hence $KI\Omega \downarrow_f$, because $KI\Omega \xrightarrow{f} \gg_f x[x := I] \not\xrightarrow{f}$. Also all computable functions can be represented by λf-calculus (compare this to the case of Lazy λ-calculus[9]). The following small lemma is essential.

Lemma 4.3 $M\{^N/_x\} \downarrow_f M' \bigwedge N \downarrow_f N' \Rightarrow$

$M[x := N] \downarrow_f M''[x := N'] \bigwedge M' \equiv M''\{^{N'}/_x\}$ $\blacksquare$

We prove the main result based on definability *á la Curry*. However it would be easy to extend the proof to another definability found e.g. in Barendregt[2]. We use the mapping: $\langle x \rangle_s \equiv x$, $\langle \lambda x.M \rangle_s \equiv \lambda x.\langle M \rangle_s$, $\langle MN \rangle_s \equiv \langle M \rangle_s \langle N \rangle_s$, and $\langle L[x := L'] \rangle_s \stackrel{\text{def}}{=} \langle L \rangle_s \{^{\langle L' \rangle_s}/_x\}$.

Theorem 4.4 *λf-definability of computable functions.* Let $\mathbf{I}$ be the set of natural numbers. If ϕ: $\mathbf{I}^k \to \mathbf{I}$ is a partial recursive function, then for some $F \in \mathcal{L}_f$, one has, for all $\vec{n} \in \mathbf{I}^k$:

- $F \lceil \vec{n} \rceil \downarrow_f M \ \wedge\ \langle M \rangle_s \equiv \lceil \phi(\vec{n}) \rceil$ (ϕ defined)
- $F \lceil \vec{n} \rceil \uparrow_f$ (otherwise)

where $\lceil \cdot \rceil$ is an encoding of the numerals in Λ. $\blacksquare$

Outline of proof Only difficulty lies in treatments of shared arguments. Consider composition of functions; $\phi(\vec{n}) = \chi(\psi_1(\vec{n}), ..., \psi_m(\vec{n}))$. Let $\chi, \psi_1, ..., \psi_m$ be λ-defined by $G, H_1, ..., H_m$, hence $G\lceil \vec{m} \rceil \xrightarrow{f} G'[\vec{y} := \lceil \vec{m} \rceil] \downarrow_f$ and $H_i \lceil \vec{n} \rceil \xrightarrow{f} H_i'[\vec{z} := \lceil \vec{n} \rceil] \downarrow_f$. But they are equated, via $\langle \rangle_s$, with $\lceil \chi(\vec{m}) \rceil$ and $\lceil \psi_i(\vec{n}) \rceil$, respectively by Lemma 4.3. Thus, $\phi(\vec{n})$ is defined by $\lambda \vec{x}.G(H_1\vec{x}) \cdots (H_m\vec{x})$. Similarly for other cases. $\blacksquare$

Theorems 3.4 and 4.4 show that λf-calculus is a self-contained functional calculus.

Finally some remarks are due on relationship between reduction in λ-calculus and reduction in λf-calculus.
In λ-calculus, $K\Omega \to_\beta \lambda x.\Omega$, but in λf-calculus $\lambda x.\Omega \not\xrightarrow{f}$ and $K\Omega \xrightarrow{f} (\lambda y.x)[x := \Omega] \uparrow_f$. On the other hand, in λf-calculus, $(\lambda xy.KIx)\Omega \xrightarrow{f} (\lambda y.KIx)[x := \Omega] \uparrow_f$. This does not accord with λ-calculus; $(\lambda xy.KIx)\Omega \to_\beta{}^{+} \lambda y.I \not\to_\beta$.

In this context, *full λf-calculus* which evaluates under λ-body ($\lambda x.M \xrightarrow{f} \lambda x.M'$) directly corresponds to full λ-calculus in its convergence.
Analogue of theorems 3.4 and 4.4 also holds for full λf-calculus. But this formalism takes us far from our basic framework, since the incorporation of full rules makes it necessary to copy environments in λ-body.

5 Conclusion

We presented a new formal system called λf-calculus which stresses *sharing* as the most essential element and studied the basic properties as a functional calculus. The optimal reduction defined by Lévy[7] is closely related to sharing, and two formalisms[3][8] using environments have been studied from the viewpoint of optimality in Lévy's sense. However **ΛCCL**[3] which executes full rules based on categorical combinators is insufficient for optimality. T^{l}RS[8], whose reduction system is *weak* as well as ours, regards a parallel reduction as one-step *meta-reduction* with labels, so it is difficult and complex to use as a practical formalism. We conjecture that the leftmost strategy on λf-calculus reaches the normal form with minimum β-reductions as far as the weak-reduction scheme goes. If this holds, the results on λf-calculus will be applied to functional executions (e.g. graph reduction) which take the call-by-name with a sharing strategy.

References

[1] Abadi,Cardelli,Curien and Lévy. Explicit substitutions. *POPL*, 1990.

[2] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* North Holland, 1984.

[3] J. Field. On laziness and optimality in lambda interpreters: Tools for specification and analysis. *POPL*, 1990.

[4] J. Hindley and J. Seldin. *Introduction to combinators and λ-calculus.* Cambridge Unversity Press, 1985.

[5] K. Honda and M. Tokoro. A formalism for concurrent object-based computing (japanese). *Computer Software*, 9, March 1992.

[6] K. Honda and M. Tokoro. On asynchronous communication semantics. *LNCS*, 612, 1992.

[7] J.-J. Lévy. Optimal reductions in the lambda-calculus. *To H.B.Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press, 1980.

[8] L. Maranget. Optimal derivations in weak lambda-calculi and in orthogonal terms rewriting systems. *POPL*, 1991.

[9] C-H.L. Ong. The lazy lambda calculus: An investigation into the foundation of functional programming. *Phd thesis in Imperial College*, May 31 1988.

[10] N. Yoshida. The report of λf-calculus. Technical report, manuscript, October 1992.