$\lambda\mu$ 計算の抽象機械

An abstract machine for type-free $\lambda\mu$-calculus

藤田 憲悦 [†]
Ken-etsu FUJITA

[†] 群馬大学 工学部 情報工学科
Department of Computer Science, Gunma University
fujita@cs.gunma-u.ac.jp

We provide a stack based machine for type-free $\lambda\mu$-calculus. The machine is derived from a sound and complete CPS-translation for $\lambda\mu$-calculus. The target calculus has let-expressions as a primitive notion, in order to handle substitution information elegantly and to simplify technical matters on the completeness. The technical improvement naturally leads to the abstract machine which handles environments explicitly. We show that the machine transitions are correct with respect to continuation semantics.

1 Introduction

We first provide a sound and complete CPS-translation for type-free $\lambda\mu$-calculus. An analysis on the calculi without type restrictions reveals core properties of the CPS-translation. Continuations are handled as a list or a stack of denotations, and formalized as a pair consisting of a denotation and a continuation in this order. The study on the type free cases also makes clear the distinction between λ-abstraction and μ-abstraction, from the viewpoint of continuations: an λ-abstraction is viewed as a constructor for a function taking only the first component of such a pair, and on the other hands, an μ-abstraction is interpreted as a constructor for a function over continuations. Our target calculus has let-expressions as a primitive notion, in order to handle substitution information (environment) elegantly and to simplify extremely technical matters on the completeness. The technical improvement naturally leads to an abstract machine for $\lambda\mu$-calculus, which handles explicitly environments. We show that the machine transitions are correct with respect to continuation semantics.

2 Type-free $\lambda\mu$-calculus and target calculus $\lambda^{\texttt{let}}$

Definition 1 (λ-calculus with `let` $\Lambda^{\texttt{let}}$)

$$\begin{aligned}\Lambda^{\texttt{let}} \ni M \quad ::= \quad & x \mid \lambda x.M \mid MM \mid \langle M, M\rangle \\ & \mid \texttt{let } \langle x, x\rangle = M \texttt{ in } M\end{aligned}$$

(β) $(\lambda x.M_1)M_2 \to M_1[x := M_2]$

(η) $\lambda x.Mx \to M$ *if* $x \notin FV(M)$

(let) $\texttt{let } \langle x_1, x_2\rangle = \langle M_1, M_2\rangle \texttt{ in } M$
$\quad \to M[x_1 := M_1, x_2 := M_2]$

($\texttt{let}_\eta$) $\texttt{let } \langle x_1, x_2\rangle = M_1 \texttt{ in } M[x := \langle x_1, x_2\rangle]$
$\quad \to M[x := M_1]$ *if* $x_1, x_2 \notin FV(M)$

Definition 2 ($\lambda\mu$-calculus $\Lambda\mu$)

$$\Lambda\mu \ni M ::= x \mid \lambda x.M \mid MM \mid \mu\alpha.[\alpha]M$$

(β) $(\lambda x.M_1)M_2 \to M_1[x := M_2]$

(η) $\lambda x.Mx \to M$ *if* $x \notin FV(M)$

(μ) $(\mu\alpha.N)M \to \mu\alpha.N[\alpha \Leftarrow M]$

(μ_β) $\mu\alpha.[\beta](\mu\gamma.N) \to \mu\alpha.N[\gamma := \beta]$

(μ_η) $\mu\alpha.[\alpha]M \to M$ *if* $\alpha \notin FN(M)$

3 Sound and complete CPS-translation

Definition 3 (CPS-Translation : $\Lambda\mu \to \Lambda^{\mathtt{let}}$)

(i) $[\![x]\!] = x$

(ii) $[\![\lambda x.M]\!] = \lambda k.(\mathtt{let}\ \langle x, k'\rangle = k\ \mathtt{in}\ [\![M]\!]k')$

(iii) $[\![M_1 M_2]\!] = \lambda k.[\![M_1]\!]\langle [\![M_2]\!], k\rangle$

(iv) $[\![\mu\alpha.[\beta]M]\!] = \lambda\alpha.[\![M]\!]\beta$

Remarked that term constructors are related between source and target calculi, as follows:

Source calculus: $\lambda\mu$	Target calculus: $\lambda^{\mathtt{let}}$
λ-abstraction	`let`-expression
application	pair
μ-abstraction	λ-abstraction
name	continuation variable

Proposition 1 (Soundness) *Let $M_1, M_2 \in \Lambda\mu$. If we have $M_1 =_{\lambda\mu} M_2$, then $[\![M_2]\!] =_{\lambda^{\mathtt{let}}} [\![M_2]\!]$.*

Proof. By induction on the derivation of $M_1 =_{\lambda\mu} M_2$. □

We introduce a grammar $\mathcal{R}$ that served as the domain of an inverse translation. Let $n, m \geq 0$. Then we write $\langle M_0, M_1, \ldots, M_n\rangle$ for $\langle M_0, \langle M_1, \ldots, M_n\rangle\rangle$, and $\langle M\rangle \equiv M$.

$$\begin{aligned}\mathcal{R} \quad ::= \quad & x \\ & \mid \lambda a.\mathcal{R}\langle \mathcal{R}_1, \ldots, \mathcal{R}_n, a\rangle \\ & \mid \lambda a.(\mathtt{let}\ \langle x, a\rangle = \langle \mathcal{R}_1, \ldots, \mathcal{R}_n, a\rangle \\ & \qquad\qquad \mathtt{in}\ \mathcal{R}\langle \mathcal{R}_1, \ldots, \mathcal{R}_m, a\rangle)\end{aligned}$$

Lemma 1 (1) *The category $\mathcal{R}$ is closed under the reduction rules of $\lambda^{\mathtt{let}}$.*

(2) *It is a well-defined equivalence relation that the binary relation $=_{\mathcal{R}}$ generated by $\to^*_{\lambda^{\mathtt{let}}}$ with respect to $\mathcal{R}$.*

Proof. Substitutions are closed with respect to $\mathcal{R}$, and hence $\mathcal{R}$ is closed under the reduction rules. □

Proposition 2 (Soundness w.r.t. $\mathcal{R}$) *Let $M_1, M_2 \in \Lambda\mu$. If we have $M_1 =_{\lambda\mu} M_2$, then $[\![M_2]\!] =_{\mathcal{R}} [\![M_2]\!]$.*

We introduce an inverse translation $\natural$ from $\mathcal{R}$ back to $\Lambda\mu$, see also the correspondence table in the above.

Definition 4 (Inverse Translation $\natural : \mathcal{R} \to \Lambda\mu$)

(i) $x^\natural = x$

(ii) $(\lambda a.R\langle R_1, \ldots, R_n, b\rangle)^\natural = \mu a.[b](R^\natural R_1^\natural \ldots R_n^\natural)$

(iii) $$\begin{aligned}&(\lambda a.(\mathtt{let}\ \langle x, b\rangle = \langle R_1, \ldots, R_m, c\rangle \\ &\qquad\qquad \mathtt{in}\ S\langle S_1, \ldots, S_n, d\rangle))^\natural \\ &\quad = \mu a.[c]((\lambda x.(\lambda b.S\langle S_1, \ldots, S_n, d\rangle)^\natural)R_1^\natural \ldots R_m^\natural)\end{aligned}$$

Lemma 2 (1) *Let $M \in \Lambda\mu$. Then we have that $[\![M]\!]^\natural \to^*_{\mu_\eta} M$.*

(2) *Let $R \in \mathcal{R}$. Then we have $[\![R^\natural]\!] \to^*_\beta R$.*

Proof. By induction on the structures of $M \in \Lambda\mu$ and $R \in \mathcal{R}$. □

Lemma 3 *Let $R, R_1, \ldots, R_n \in \mathcal{R}$. Then we have $(R[a := \langle R_1, \ldots, R_n, a\rangle])^\natural = R^\natural[a \Leftarrow R_1^\natural, \ldots, R_n^\natural]$.*

Proof. By induction on the structure of $R \in \mathcal{R}$. □

Lemma 4 *Let $P, Q \in \mathcal{R}$.*

(1) *If $P \to_\beta Q$ then $P^\natural \to^+_{\mu\mu_\beta} Q^\natural$.*

(2) *If $P \to_\eta Q$ then $P^\natural \to_{\mu_\eta} Q^\natural$.*

(3) *If $P \to_{\mathtt{let}} Q$ then $P^\natural \to^+_{\beta\mu\mu_\beta} Q^\natural$.*

(4) *If $P \to_{\mathtt{let}_\eta} Q$ then $P^\natural =_{\beta\eta\mu\mu_\eta} Q^\natural$.*

Proof. By induction on the derivation of $P \to Q$. □

Proposition 3 (Completeness) *Let $P_1, P_2 \in \mathcal{R}$. If we have $P_1 =_{\mathcal{R}} P_2$, then $P_1^\natural =_{\lambda\mu} P_2^\natural$.*

Theorem 1 (i) *Let $M_1, M_2 \in \Lambda\mu$. $M_1 =_{\lambda\mu} M_2$ if and only if $[\![M_1]\!] =_{\mathcal{R}} [\![M_2]\!]$.*

(ii) *Let $P_1, P_2 \in \mathcal{R}$. $P_1 =_{\mathcal{R}} P_2$ if and only if $P_1^\natural =_{\lambda\mu} P_2^\natural$.*

4 Abstract machine with explicit environment

Finally we introduce a stack based machine for $\lambda\mu$-calculus, which handles environments explicitly and is motivated by our target calculus with `let`-expressions that encapsulate environments consisting of denotations or continuations.

There exists a well-known connection between continuation passing style [Seli98, SR98, Fuji03a] and abstract machines [Plot75, Bier98, deGr98]. For instance, according to [SR98], we have relations between $\langle$denotation, continuation, environment$\rangle$ and $\langle$closure, stack, environment$\rangle$, as follows:

Continuation Denotational Semantics	denotation D $[\![\ ]\!] : \Lambda \times E \to D$ $D = [K \to R]$	continuation K $D \times K$
Abstract Machine	closure $Clos$ $\Lambda \times E$	stack S $Clos \times S$

Continuation Denotational Semantics	environment E $Var \to D$ $Cvar \to K$
Abstract Machine	environment E $Var \to Clos$

where Λ is a set of terms, and R is a domain of responses.

The connection makes it possible to relate an abstract machine to its continuation denotational semantics as done in [SR98, Seli98]. Here, we are interested in compiler correctness, based on which $\lambda\mu$-terms are compiled into codes of an abstract machine. Such a machine is the so-called Krivine's abstract machine, compared with Landin's SECD machine [Lan64] for call-by-value.

We introduce here an abstract machine with a modification, such that the environment explicitly handles substitution information. The machine has configurations of the form $\langle\!|[M, E], K|\!\rangle$, where $[M, E]$ is the closure consisting of a term M (instruction) and the environment E, and K is the continuation consisting of a closure and a continuation. Environments are defined by a list of substitution information together with two kinds of distinguished indices denoted by x and α. We employ `nil` and :: for the empty list and cons, respectively.

Environment (list of continuations or closures)

$$E ::= \texttt{nil} \mid (x = cl) :: E \mid (\alpha = K) :: E$$

Continuation (stack of closures)

$$K ::= k \mid \langle cl, K\rangle \mid E(\alpha) \mid \texttt{snd}(K)$$

Closure

$$cl ::= [M, E] \mid E(x) \mid \texttt{fst}(K)$$

The transition function $\Rightarrow_i$ specifies how to execute the machine following terms (instructions code), in the sense that one step execution transforms the configuration $\langle\!|[M, E], K|\!\rangle$.

1. $\langle\!|[x, E], K|\!\rangle \Rightarrow_i \langle\!|E(x), K|\!\rangle$
2. $\langle\!|[\lambda x.M, E], \langle cl, K'\rangle|\!\rangle$
   $\Rightarrow_i \langle\!|[M, (x = cl) :: E], K'|\!\rangle$
3. $\langle\!|[\lambda x.M, E], K|\!\rangle \Rightarrow_i$
   $\langle\!|[M, (x = \texttt{fst}(K)) :: E], \texttt{snd}(K)|\!\rangle$ otherwise
4. $\langle\!|[M_1 M_2, E], K|\!\rangle \Rightarrow_i \langle\!|[M_1, E], \langle [M_2, E], K\rangle|\!\rangle$
5. $\langle\!|[\mu\alpha.[\beta]M, E], K|\!\rangle$
   $\Rightarrow_i \langle\!|[M, (\alpha = K) :: E], ((\alpha = K) :: E)(\beta)|\!\rangle$

Moreover, environments are also handled by the transition function $\Rightarrow_e$.

1. $\langle\!|((x = cl) :: E)(x), K|\!\rangle \Rightarrow_e \langle\!|cl, K|\!\rangle$
2. $\langle\!|((x' = cl) :: E)(x), K|\!\rangle \Rightarrow_e \langle\!|E(x), K|\!\rangle$
   if $x \not\equiv x'$
3. $\langle\!|((k = K') :: E)(x), K|\!\rangle \Rightarrow_e \langle\!|E(x), K|\!\rangle$
4. $\langle\!|cl, ((\alpha = K) :: E)(\alpha)|\!\rangle \Rightarrow_e \langle\!|cl, K|\!\rangle$
5. $\langle\!|cl, ((\alpha' = K) :: E)(\alpha)|\!\rangle \Rightarrow_e \langle\!|cl, E(\alpha)|\!\rangle$
   if $\alpha \not\equiv \alpha'$
6. $\langle\!|cl, ((x = cl') :: E)(\alpha)|\!\rangle \Rightarrow_e \langle\!|cl, E(\alpha)|\!\rangle$

For each $M \in \Lambda\mu$, the abstract machine is executed from $\texttt{Comp}(M) \overset{\text{def}}{=} \langle\!|[M, \texttt{nil}], k|\!\rangle$, denoted by $\texttt{Comp}(M) \Rightarrow \langle cl_1, K_1\rangle \Rightarrow \langle cl_2, K_2\rangle \Rightarrow \ldots$, as follows:

1. Following the instruction M, we apply $\Rightarrow_i$. Then apply $\Rightarrow_e$ in the case of $\Rightarrow_i$ (1) or (4) as follows:
   (a) Case of $\langle\!|E(x), K|\!\rangle$
   Apply repeatedly $\Rightarrow_e$ (1,2,3) until either $\langle\!|\texttt{nil}(x), K|\!\rangle$ or $\langle\!|cl, K|\!\rangle$ where $cl \not\equiv E(x)$ is obtained.
   (b) Case of
   $$\langle\!|[M, (\alpha = K) :: E], ((\alpha = K) :: E)(\beta)|\!\rangle$$
   Apply repeatedly $\Rightarrow_e$ (4,5,6) until either $\langle\!|cl, \texttt{nil}(\alpha)|\!\rangle$ or $\langle\!|cl, K|\!\rangle$ where $K \not\equiv E(\beta)$ is obtained.
2. Repeat the process above.

That is, a sequence of the machine transitions $\Rightarrow^*$ consists of $(\Rightarrow_i \cdot \Rightarrow_e^*)^*$, where $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$.

At the application of $\Rightarrow_i$, the machine always has the following configuration $\langle\!|[M, E], K_{nf}|\!\rangle$:

$$K_{nf} ::= \texttt{snd}^n(k) \mid \texttt{snd}^n(\texttt{nil}(\alpha)) \mid \langle [M', E'], K_{nf}\rangle$$

If the machine halts, then the state has the configuration $\langle\!|cl_{nf}, K_{nf}|\!\rangle$, as follows:

$$cl_{nf} ::= \texttt{fst}(\texttt{snd}^n(k)) \mid \texttt{fst}(\texttt{snd}^n(\texttt{nil}(\alpha))) \mid \texttt{nil}(x)$$

Now configurations can be assumed to be in the following form $\langle\!|cl_{nf}, K_{nf}|\!\rangle$:

$$\begin{array}{lcl} cl_{nf} & ::= & \texttt{nil}(x) \mid \texttt{fst}(\texttt{snd}^n k) \\ & & \mid \texttt{fst}(\texttt{snd}^n(\texttt{nil}(\alpha))) \mid [M, E_{nf}] \\ E_{nf} & ::= & \texttt{nil} \mid (x = cl_{nf}) :: E_{nf} \\ & & \mid (\alpha = K_{nf}) :: E_{nf} \\ K_{nf} & ::= & \texttt{snd}^m(k) \mid \texttt{snd}^m(\texttt{nil}(\alpha)) \\ & & \mid \langle [M, E_{nf}], K_{nf}\rangle \end{array}$$

Let K_{nf} be $\langle cl_{nf_1}, \ldots, cl_{nf_p}, \texttt{snd}^m(\texttt{nil}^\delta(k))\rangle$ where $m, n \geq 0$ and δ is either 0 or 1. Then we have $n < m$, since we have the second clause of $\Rightarrow_i$. Now, for $\langle\!|\langle cl_{nf}, K_{nf}\rangle|\!\rangle$, its CPS-code in terms of $\lambda^{\texttt{let}}$ is defined as follows:

$$[\![\langle\!|cl_{nf}, K_{nf}|\!\rangle]\!] = \lambda k.\texttt{let}\ \langle x_1, \ldots, x_m, k\rangle = k\ \texttt{in}\ [\![cl_{nf}]\!][\![K_{nf}]\!] \quad \text{if } 0 \leq n < m$$

$$[\![\langle\!|cl_{nf}, K_{nf}|\!\rangle]\!] = \lambda k.[\![cl_{nf}]\!][\![K_{nf}]\!] \quad \text{if } m = 0$$

1. $[\![[M, E_{nf}]]\!] = [\![M]\!]_{E_{nf}}$
   (a) $[\![x]\!]_{E_{nf}} = x$ if $E_{nf}(x) = \texttt{nil}(x)$
   $[\![x]\!]_{E_{nf}} = [\![M']\!]_{E'_{nf}}$ if $E_{nf}(x) = [M', E'_{nf}]$
   $[\![x]\!]_{E_{nf}} = x_{n+1}$ if $E_{nf}(x) = \texttt{fst}(\texttt{snd}^n(\texttt{nil}^\delta(k)))$
   (b) $[\![\lambda x.M]\!]_{E_{nf}} = \lambda k'.(\texttt{let}\ \langle x, k\rangle = k'\ \texttt{in}\ [\![M]\!]_{E_{nf}} k)$
   (c) $[\![M_1 M_2]\!]_{E_{nf}} = \lambda k.[\![M_1]\!]_{E_{nf}}\langle [\![M_2]\!]_{E_{nf}}, k\rangle$
   (d) $[\![\mu\alpha.[\beta]M]\!]_{E_{nf}} = \lambda\alpha.[\![M]\!]_{E_{nf}}[\![K'_{nf}]\!]$ where $K'_{nf} = E_{nf}(\beta)$
2. $[\![\texttt{fst}(\texttt{snd}^n(\texttt{nil}^\delta(k)))]\!] = x_{n+1}$
3. $[\![\texttt{nil}(x)]\!] = x$

1. $[\![\langle [M', E'_{nf}], K'_{nf}\rangle]\!] = \langle [\![M']\!]_{E'_{nf}}, [\![K'_{nf}]\!]\rangle$
2. $[\![\texttt{snd}^m(\texttt{nil}^\delta(k))]\!] = k$

Lemma 5 *1. We have* $[\![M]\!]_{(x=cl_{nf})::E_{nf}} = [\![M]\!]_{E_{nf}}[x := [\![cl_{nf}]\!]]$*, provided that* $x \notin Dom(E_{nf})$*.*

2. $[\![M]\!]_{(\alpha=K_{nf})::E} = [\![M]\!]_{E_{nf}}[\alpha := [\![K_{nf}]\!]]$*, provided that* $\alpha \notin Dom(E_{nf})$*.*

Proof. By induction on the structure of $M \in \Lambda\mu$. □

Proposition 4 *If we have* $\langle\!|cl_1, K_1|\!\rangle \Rightarrow \langle\!|cl_2, K_2|\!\rangle$*, then* $[\![\langle\!|cl_1, K_1|\!\rangle]\!] \rightarrow^*_{\beta\texttt{let}} [\![\langle\!|cl_2, K_2|\!\rangle]\!]$*.*

Proof. By induction on the structure of $cl_{nf} = [M, E_{nf}]$. □

Theorem 2 (Adequacy of the Machine)

1. $[\![\texttt{Comp}(M)]\!] =_{\lambda^{\texttt{let}}} [\![M]\!]$

2. If we have $\langle\!|cl_1, K_1|\!\rangle \Rightarrow^* \langle\!|cl_2, K_2|\!\rangle$*, then* $[\![\langle\!|cl_1, K_1|\!\rangle]\!] =_{\lambda^{\texttt{let}}} [\![\langle\!|cl_2, K_2|\!\rangle]\!]$*.*

Proof. We have $[\![\texttt{Comp}(M)]\!] \rightarrow^+_\beta [\![M]\!]$. From Proposition 4, we have the second clause. □

Corollary 1 *For each machine transition* $\langle\!|cl_1, K_1|\!\rangle \Rightarrow \langle\!|cl_2, K_2|\!\rangle$*, there exist the corresponding* $\lambda\mu$*-reductions* $M_1 \rightarrow^*_{\lambda\mu} M_2$ *for some* M_1, M_2*.*

Proof. If we have $\langle\!|cl_1, K_1|\!\rangle \Rightarrow \langle\!|cl_2, K_2|\!\rangle$, then $[\![\langle\!|cl_1, K_1|\!\rangle]\!] \to^*_{\beta\mathtt{let}} [\![\langle\!|cl_2, K_2|\!\rangle]\!]$ from Proposition 4. Hence, we have $\lambda\mu$-reductions $[\![\langle\!|cl_1, K_1|\!\rangle]\!]^\natural \to^*_{\beta\mu\mu_\beta} [\![\langle\!|cl_2, K_2|\!\rangle]\!]^\natural$ from Lemma 4. □

Lemma 6 *We have* $[\![M[x := M']]\!]_{E_{nf}} = [\![M]\!]_{E_{nf}}[x := [\![M']\!]_{E_{nf}}]$, *provided that* x *can appear only in* M.

Proof. By induction on $M \in \Lambda\mu$. □

Lemma 7 *We have* $[\![M[\alpha \Leftarrow M']]\!]_{E_{nf}} \to^*_\beta [\![M]\!]_{(\alpha=\langle[M',E_{nf}],K'\rangle)::E_{nf}}$ *where* $K' = E_{nf}(\alpha)$, *provied that* $\alpha \notin FN(M')$.

Proof. By induction on $M \in \Lambda\mu$. □

Lemma 8 *If we have* $M \to_{\lambda\mu} M'$, *then* $[\![M]\!]_{E_{nf}} =_{\lambda^{\mathtt{let}}} [\![M']\!]_{E_{nf}}$.

Proof. By induction on the derivation of $M \to_{\lambda\mu} M'$. □

We show that $\lambda\mu$-reductions are executed by the abstract machine whose configurations are equal each other in the sense of the continuation semantics.

Proposition 5 *If we have* $M_1 \to_{\lambda\mu} M_2$, *then* $[\![\langle\!|[M_1, E], K|\!\rangle]\!] =_{\lambda^{\mathtt{let}}} [\![\langle\!|[M_2, E'], K'|\!\rangle]\!]$ *for some* E' *and* K'.

Proof. It is enough to show that we have $[\![M_1]\!]_{E_{nf}}[\![K_{nf}]\!] =_{\lambda^{\mathtt{let}}} [\![M_2]\!]_{E'_{nf}}[\![K'_{nf}]\!]$ for some E'_{nf} and K'_{nf}. □

Theorem 3 (Correctness for $\lambda\mu$-reductions) *If we have* $M_1 \to_{\lambda\mu} M_2 \to_{\lambda\mu} M_3 \to_{\lambda\mu} \cdots$, *then* $[\![\mathtt{Comp}(M_1)]\!] =_{\lambda^{\mathtt{let}}} [\![\langle\!|[M_2, E_2], K_2|\!\rangle]\!] =_{\lambda^{\mathtt{let}}} [\![\langle\!|[M_3, E_3], K_3|\!\rangle]\!] =_{\lambda^{\mathtt{let}}} \cdots$ *for some* E_i *and* K_i.

5 Concluding remarks

The CPS-translation into the target calculus with `let`-expressions as a primitive notion can be naturally carried over, for instance, to polymorphic λ-calculus, system F and 2nd order typed $\lambda\mu$-calculus, as done in [Fuji05, Hasse05].

We have shown the adequacy and correctness of the abstract machine with respect to continuation semantics only in terms of $\lambda^{\mathtt{let}}$. This result can also be enlarged with respect to continuation denotation semantics [SR98]. Our abstract machine on this version is deterministic and works not only for weak head reductions but also for head reductions. Moreover, for full $\lambda\mu$-reductions, the machine is still correct with respect to continuation semantics, as shown in Theorem 3 for $\to_{\lambda\mu}$, although it cannot handle directly components in the environment. but can do only after $\Rightarrow_i$ (1) or (4), i.e., the machine has neither a value closure nor a value environment because of call-by-name. The machine can also be modified as a lower level machine with nameless terms by the use of de Bruijn indices [deGr98].

Acknowledgement This research has been supported by Grants-in-Aid for Scientific Research (C)17500004, Japan Society for the Promotion of Science.

参考文献

[Bare84] H. P. Barendregt: *The Lambda Calculus, Its Syntax and Semantics* (revised edition), North-Holland, 1984.

[Bier98] G. M. Bierman: A computational interpretation of the $\lambda\mu$-calculus, Lecture Notes in Computer Science 1450, pp. 336-345, 1998.

[deGr98] Ph. de Groote: An environment machine for the $\lambda\mu$-calculus, *Math. Struct. in Compu. Science*, 1998.

[Fuji03a] K. Fujita: Simple Models of Type Free $\lambda\mu$-Calculus, *Computer Software*, Japan Society for Software Science and Technology, Vol. 20, No. 3, pp. 73–79, 2003.

[Fuji03b] K. Fujita: A sound and complete CSP-translation for $\lambda\mu$-Calculus, Lecture Notes in Computer Science 2701, pp. 120–134, 2003.

[Fuji05] K. Fujita: Galois embedding from polymorphic types into existential types, Lecture Notes in Computer Science 3461, pp. 194–208, 2005.

[Grif90] T. G. Griffin: A Formulae-as-Types Notion of Control, *Proc. the 17th Annual ACM Symposium on Principles of Programming Languages*, pp. 47–58, 1990.

[Hasse05] M. Hasegawa: Relational Parametricity and Control, *Proc. IEEE Symposium on Logic in Computer Science*, 2005.

[HS97] M. Hofmann and T. Streicher: Continuation models are universal for $\lambda\mu$-calculus, *Proc. the 12th Annual IEEE Symposium on Logic in Computer Science*, pp. 387–395, 1997.

[How80] W. Howard: The Formulae-as-Types Notion of Constructions, in: *To H.B.Curry: Essays on combinatory logic, lambda-calculus, and formalism*, Academic Press, pp. 479–490, 1980.

[Lan64] P. J. Landin: The mechanical evaluation of expressions, Computer Journal Vol. 6, pp. 308–320, 1964.

[Murt91] C. R. Murthy: An Evaluation Semantics for Classical Proofs, *Proc. the 6th Annual IEEE Symposium on Logic in Computer Science*, pp. 96–107, 1991.

[Pari92] M. Parigot: $\lambda\mu$-Calculus: An Algorithmic Interpretation of Classical Natural Deduction, Lecture Notes in Computer Science 624, pp. 190–201, 1992.

[Pari93] M. Parigot: Classical Proofs as Programs, Lecture Notes in Computer Science 713, pp. 263–276, 1993.

[Pari97] M. Parigot: Proofs of Strong Normalization for Second Order Classical Natural Deduction, *J. Symbolic Logic*, Vol. 62, No. 4, pp. 1461–1479, 1997.

[Plot75] G. Plotkin: Call-by-Name, Call-by-Value and the λ-Calculus, *Theoretical Computer Science*, Vol. 1, pp. 125–159, 1975.

[Reyn93] J. C. Reynolds: The discoveries of continuation, *Lisp and Symbolic Computation*, Vol. 6, pp. 233–247, 1993.

[Seli98] P. Selinger: An implementation of the call-by-name $\lambda\mu\nu$-calculus, manuscript, 1998.

[Seli01] P. Selinger: Control Categories and Duality: on the Categorical Semantics of the Lambda-Mu Calculus, *Math. Struct. in Compu. Science*, Vol. 11, pp. 207–260, 2001.

[SR98] T. Streicher and B. Reus: Classical Logic, Continuation Semantics and Abstract Machines, *J. Functional Programming*, Vol. 8, No. 6, pp. 543–572, 1998.