数理解析研究所講究録
996 巻 1997 年 143-157

A Calculus of Parallel Continuations and its Monad Models

SATO Hiroyuki
Department of Computer Science and Communication Engineering,
Kyushu University,
Kasuga, Fukuoka, 816, Japan
E-mail: schuko@csce.kyushu-u.ac.jp

Abstract

The importance of continuation is increasing in computer science. Because continuation is an abstraction of single thread, however, we need a substantial extension of continuation in order to directly manipulate parallelism using continuation.

In this paper, we re-define continuation with anonymous returns and compositions which make the algebra of continuations tractable. We give a monad model to a subclass of continuations. Moreover, we introduce a parallel construct to continuations, i.e., extension with pairing($\langle -,- \rangle$). We show that the synchronization can be represented in this framework. We also give its model in monoidal categories. Furthermore, we discuss a gap between cartesian categories in which value-based model can be constructed, and monoidal categories in which parallel continuations can be represented.

1 Introduction

The importance of continuation is increasing in computer science. From the viewpoint that continuation is the abstraction of control, a number of advanced concepts of control flow are expressed by using continuations. In the study of control, parallelism is today one of the most important problems. Continuation is, however, essentially an abstraction of *single* thread control. This means that in order to directly manipulate parallelism, we need a substantial extension of the concept of continuation.

Today, continuations are applied to a variety of fields in computer science. Theroretical success[4, 7, 8] and [1, 2]'s practical success of the application of continuation to transforming functional languages(SML) into conventional (sequential) machine architectures are most notable. It is a natural idea that, to utilize these successes, we extend continuations to those which can handle parallelism. Previous works of multiprocessing based on continuations (e.g. [18]) express parallelism as a set of single thread continuations. They do not directly manipulate parallelism, though their contribution to the implementations of concurrent languages are remarkable(e.g. [15]).

Continuation can also be a candidate for the framework of the theory of concurrency because it is an abstraction of control. Existing models of concurrency(CSP[10], CCS[12], Process Algebra[3], Intuitionistic Linear Logic[16], π-calculus, etc.) are equipped with simplicity in syntax and semantics. However, their main aim is representing communication, and they are necessarily not computation-oriented. We aim at extending continuations which is computation-oriented in their nature to those expressive enough to represent communication.

In this paper, we introduce a new constructor $\langle -,- \rangle$(pairing) into continuations in order to express parallelism. First, we re-define conventional continutations for algebraic manipulations. Second, we define an algebraic construct $\langle -,- \rangle$ for parallelism, and discuss the algebra of the extended continuations. Furthermore, we classify the hierarchy of extended continuations with $\langle -,- \rangle$ according to the expressive power of the algebra of extended continuations.

Also in this paper, we give a categorical framework for interpreting our continuataions, both sequential and parallel. Monad theorey is intensively used. It is shown that a number of programming language constructs can be represented as monads[13]. It is well known as Moggi's Hypothesis[17] that every feature of a programming language can be modelled by a monad. From this point of view, this paper aims at modelling parallelism in monad theory. First, a class of sequential continuations which are modelled in monads are specified in the similar mannar as in [17], although the expressive power of monad-theoretic continuations are proved to be rather restricted. Wadler analyzed continuations extended with `call/cc`[14, 5], or `reset` and `shift`[4], gave their monad-theoretic interpretation, and showed that the extensions are indispensable to the monad-theoretic continuations. Our continuations are also interpreted in monad theory. Our **ret** and ; constructs are precisely interpreted as a monad theoretic continuations. We give a construct for monad, as [9]'s representations. Second, we extend our method for modelling parallel continutations. In modelling our extended continuations $\mathcal{PCONT}$, we require that the underlying structure must be monoidal, and that (sync monoidal) monads on monoidal categories are studied. The important concepts such as synchronization are represented in monads on monoidal categories. We show that, unlike the sequential continuation, monads are enough for representing basic concepts of parallel continuations.

The organization of this paper is as follows: Section 2 defines the λ-calculus as the base language of this paper. Section 3 discuss the conventional (sequential) continuations. Section 4 models sequential continuations in monad theory. Section 5 introduces a construct into continuations in order to represent parallelism. In particular, we show that this extended continuations can express synchronization. Section 6 gives a model of this extended continuations in monoidal categories. Section 7 is devoted to the discussion about monoidal categories and cartesian categories from the viewpoint of our modelling of parallelism. Section 8 gives a brief summary.

2 Λ

Let Λ be the usual untyped λ-calculus. Let Y be the fixpoint combinator. In this paper, we develop our theory on Λ.

Definition 1 A value is defined as a closed normal form in Λ. The set of values is denoted by $\mathcal{VAL}$.

Definition 2 [Typability] On Λ, the untyped system, we define the typability relation as follows:

$$\frac{t \text{ is a term of } \Lambda}{t : \Lambda} \qquad \frac{t \text{ is a closed normal form}}{t : \mathcal{VAL}},$$

$$\frac{\begin{matrix}(x:S)\\ \vdots \\ M:T\end{matrix}}{\lambda x.M : S \supset T} \qquad \frac{M : S \supset T \quad N : S}{MN : T}$$

$$\frac{f : S \supset S}{Yf : S \supset S.}$$

The rule of Y is rather technical.

The types and polynomial terms on Λ make a category in the sense of [11].

Definition 3 Let $\mathcal{C}$ be the category defined as:

- an object is a type, and
- an arrow of $a \longrightarrow b$ is a term $\lambda x.t$ to which a type $a \supset b$ can be assigned.

3 Sequential Continuation

3.1 Definition of Sequential Continuation

Let $\mathcal{VAL}$ be defined as before. We first define two constructors for (sequential) continuation.

Definition 4 s$\mathcal{SCONT}$(Structural Sequential CONTinuation) is defined as the domain of λ-expressions with constructs `ret`(anonymous return) and ;(sequential composition):

$$\texttt{ret} \in s\mathcal{SCONT},$$

$$c_0 ; c_1 \in s\mathcal{SCONT}(c_0, c_1 \in s\mathcal{SCONT}),$$

$$a ; c \in s\mathcal{ANS}(a \in s\mathcal{ANS}, c \in s\mathcal{SCONT}).$$

Definition 5 [Typability]
Type assingment rules are augumented as follows:

$$\frac{}{\texttt{ret} : \mathrm{s}\mathcal{SCONT}},$$

$$\frac{v : \mathcal{VAL}}{\texttt{ret}v : \mathrm{s}\mathcal{ANS}},$$

$$\frac{c_0 : \mathrm{s}\mathcal{SCONT} \qquad c_1 : \mathrm{s}\mathcal{SCONT}}{c_0\,;c_1 : \mathrm{s}\mathcal{SCONT}},$$

$$\frac{a : \mathrm{s}\mathcal{ANS} \qquad c : \mathrm{s}\mathcal{SCONT}}{a;c : \mathrm{s}\mathcal{ANS}}.$$

Definition 6 We denote the continuations and answers which can be constructed using `ret` and ; with primitive operators on *Int* by *syntactically-structural.*

In general, s$\mathcal{SCONT}$ is weak in its expressive power. We define full $\mathcal{SCONT}$as follows.

Definition 7 [Full $\mathcal{SCONT}$]
$\mathcal{SCONT}$ is defined as the domain of λ-expressions which are typable as $\mathcal{VAL} \supset \mathcal{ANS}$ with the following rules which may have some primitive operators of $\mathcal{SCONT}$and $\mathcal{ANS}$.

$$\frac{c : s\mathcal{SCONT}}{c : \mathcal{SCONT}}$$

$$\frac{c : s\mathcal{ANS}}{c : \mathcal{ANS}}$$

$$\frac{c : \mathcal{SCONT} \qquad v : \mathcal{VAL}}{cv : \mathcal{ANS}}$$

$$\frac{\begin{array}{c}(x : \mathcal{VAL})\\ \vdots\\ r : \mathcal{ANS}\end{array}}{\lambda x.r : \mathcal{SCONT}}$$

Definition 8 [Reduction]
The reduction rules ⟶ of $\mathcal{SCONT}$ and $\mathcal{ANS}$ are those of ordinary λ-calculus with those for new constructs.

$$\frac{c_0 \longrightarrow c_0'}{c_0\,;c \longrightarrow c_0'\,;c}$$

$$\frac{}{(c_0\,;c_1)v \longrightarrow c_0v\,;c_1}$$

$$\frac{}{(\texttt{ret}\ v)\,;c \longrightarrow cv}$$

Definition 9 [Returning]
Let $c \in \mathcal{SCONT}$ and $v \in \mathcal{VAL}$ are given. Then c is *returning* at v if there exists $w_v \in \mathcal{VAL}$ such that $cv \longrightarrow \texttt{ret}\ w_v$.

Returning property corresponds to the normalizability in the theory of λ-calculus.

Note that a returning continuation is normalizing, but that a normalizing term is not necessarily returning.

3.2 Anonymous Return

The conventional definition of continuation implicitly uses the construct `ret` as the initial continuation at the evaluation.

In this paper, we explicitly define `ret` as the anonymous return continuation. The conventional continuation which returns to the initial continuaion *init* of the execution is obtained by replacing `ret` by *init*. The only difference with the conventional continuation is that in our continuations, the return continuation `ret` are explicitly defined and reduction rules over `ret` are specified. Though the expressive power of our continuations is not richer than that of conventional ones, our definitions of anonymous returns make operations on continuations (`ret` and composition) tractable.

3.3 Basic Properties of $\mathcal{SCONT}$

As a basic property of $\mathcal{SCONT}$, we show that $\mathcal{SCONT}$ enjoys the associativity of ;(composition).

Lemma 1 [Associativity] If c_0 is returning at v, the two continuations $(c_0;c_1);c_2$ and $c_0;(c_1;c_2)$ reduce to the same continuation at v.

Proof

Let $c_0v \longrightarrow w_v$. $((c_0;c_1);c_2)v \longrightarrow (c_0;c_1)v;c_2 \longrightarrow (c_0v;c_1);c_2 \longrightarrow (\texttt{ret}\ w_v;c_1);c_2 \longrightarrow c_1w_v;c_2$. On the other hand, $(c_0;(c_1;c_2))v \longrightarrow c_0v;(c_1;c_2) \longrightarrow \texttt{ret}\ w_v;(c_1;c_2) \longrightarrow (c_1;c_2)w_v \longrightarrow c_1w_v;c_2$. □

From this lemma, we add the axiom

$$\frac{}{c_0;(c_1;c_2) \longrightarrow (c_0;c_1);c_2},$$

and omit parentheses in the composition of ;(sequential composition).

Example 1 Let

$$\begin{aligned}\text{fact} \ \equiv\ & Y(\lambda f.\lambda x.\texttt{if}\ x = 0\ \texttt{then ret}\ 1\\ & \quad \texttt{else}\ (f(x-1);\lambda v.\texttt{ret}\ v * x)),\end{aligned}$$

where numerals and `if ... then ... else` are appropriately defined. Then

$$\begin{aligned}\text{fact3} &\longrightarrow \text{fact2};\lambda v.\texttt{ret}(v*3)\\ &\longrightarrow \text{fact1};\lambda w.\texttt{ret}(w*2);\lambda v.\texttt{ret}(v*3)\\ &\longrightarrow \text{fact0};\lambda u.\texttt{ret}(u*1);\lambda w.\texttt{ret}(w*2);\lambda v.\texttt{ret}(v*3)\\ &\longrightarrow \texttt{ret1};\lambda u.\texttt{ret}(u*1);\lambda w.\texttt{ret}(w*2);\lambda v.\texttt{ret}(v*3)\\ &\longrightarrow \texttt{ret1};\lambda w.\texttt{ret}(w*2);\lambda v.\texttt{ret}(v*3)\\ &\longrightarrow \texttt{ret2};\lambda v.\texttt{ret}(v*3)\\ &\longrightarrow \texttt{ret}(2*3) = \texttt{ret6}.\end{aligned}$$

We further define the transformation by which we can hide the application of a value to a continuation.

Definition 10 [Representation of Application]

We define the translation $(-)^\circ : \Lambda \longrightarrow \Lambda$ as:

- $(cv)^\circ \equiv\ \texttt{ret}v;c^\circ$ $(c : \mathcal{SCONT},\ v : \mathcal{VAL})$,
- $(cc')^\circ \equiv c^\circ(c')^\circ$, other application,
- $(\lambda x.t)^\circ \equiv \lambda x.t^\circ$,
- $t^\circ \equiv t$, otherwise.

This transformation is commutative with the reduction in the following sense:

Lemma 2 For two terms m and r, if $m \longrightarrow r$, $m^\circ \longrightarrow r$ holds.

Proof

We only prove the case of applications. By the definition of $\longrightarrow$, $\texttt{ret}v;c \longrightarrow cv$. This means that if $cv \longrightarrow r$, $(cv)^\circ \longrightarrow r$ because $(cv)^\circ \equiv \texttt{ret}v;c \longrightarrow cv \longrightarrow r$. □

From this lemma, we only consider the °-transformed terms.

Proposition 1 Under the °-transforamtion, the law of reduction

$$(c_0;c_1)v \longrightarrow c_0v;c_1 \qquad (\mathit{Appl})$$

can be replaced with the associativity law:

$$c_0;(c_1;c_2) \longrightarrow (c_0;c_1);c_2 \qquad (\mathit{Assoc})$$

Proof
Under the °-transforamtion, the law (Appl) is transformed to

$$\mathtt{ret}v\,\mathtt{;}(c_0\,\mathtt{;}c_1) \longrightarrow (\mathtt{ret}v\,\mathtt{;}c_0)\,\mathtt{;}c_1.$$

(Assoc) ⇒ (Appl) is now obvious. □
We have already showed that from Lemma 1, (Appl) ⇒ (Assoc) makes sense if c_0 is returning. We have furthermore showed that (Assoc) and (Appl) are equivalent under the condition that c_0 is returning under the °-transformation. The above discussions imply that the two constructors `ret` and `;` are useful for building some meaningfull class.

From the Lemma 2, by the °-transformation, the application of a value to a continuation can be said to be structural. Therefore we freely use the applications to continuations in the discussion of syntactic structurality because it can be removed by °-transformations.

4 Interpretations of $\mathcal{SCONT}$Using Monads

In this section, we give a categorical semantics of $\mathcal{SCONT}$.

In presenting monad, We adopt Kleisli's notation.

Definition 11 [Monad] A monad is defined as

1. a data constructor M,
2. arrows $\eta_a : a \longrightarrow Ma$ and an operation $\star$ which combines two arrows $f : a \longrightarrow Mb$ and $g : b \longrightarrow Mc$ as $f \star g : a \longrightarrow Mc$, and
3. the following laws:

   Left Unit $\eta v \star k = kv$.

   Right Unit $m \star \eta = m$.

   Associative $m \star (k \star h) = (m \star k) \star h$.

We moreover require that η must be a natural transformation $I \longrightarrow M$.

Proposition 2 Given $c : A \longrightarrow MC$, we can construct $c^* : MA \longrightarrow MC$ such that $c^* \cdot \eta = c$.

Proof
Let $c^* \equiv id(: MA \longrightarrow MA) \star c(: A \longrightarrow MC)$. This satisfies the condition. □
We show some examples of monads.

Example 2 [Identity Monad]
The identity functor makes itself a monad

$$\begin{aligned} M(A) &\equiv A \\ \eta v &\equiv v \\ m \star k &\equiv k \cdot m. \end{aligned}$$

Example 3 [CPS-Monad]
Let R be an object representing "result" or value. Then the CPS-monad[13] is defined as:

$$\begin{aligned} M(A) &\equiv R^{R^A} \\ \eta v &\equiv \lambda c.cv \\ m \star k &\equiv \lambda c.m(\lambda v.kvc). \end{aligned}$$

In this section, we analyze continuations using monads, and show some theorems as a basis for analyzing our parallel continuations.

Monad is widely used as one of standard methods for analyzing data constructors(e.g. [13]). In particular, [17] analyzes continuations using monads. In [17], Wadler defines levels of continuations. The following definition is the one of meta-level continuation.

Definition 12 Let $\mathcal{M}$ be a category which models types and terms of Λ in the sense that there is a structure preserving functor from $\mathcal{C}$ of Definition 3. On $\mathcal{M}$, monad-theoretically, we define:

continuation is an arrow from an object a to Mb for some a and b, and the domain of

answer is the union of objects Ma for a, an object.

We classify continuations using monad theory. We consider a core class of continuations as the one which can be expressed by using only monads.

Note that clearly we cannot represent the full expressive power of continuations by using only monads, but we need extra constructors(e.g. `shift` and `reset`[9], `call/cc`[14, 5]).

Definition 13 We denote the continuations which can be constructed using only η and $\star$ by *semantically-structural.*

We first interpret $\mathcal{SCONT}$ by means of monads.

Definition 14 Given a monad M, we define the interpretation $[\![-]\!]$ of $\mathcal{SCONT}$in the following:

$$\begin{array}{lcl} \mathcal{ANS} & \longmapsto & \bigcup_{a\in Obj} Ma \\ \mathcal{SCONT} & \longmapsto & \bigcup_{a,b\in Obj}(a \longrightarrow Mb). \end{array}$$

As for terms, we give definitions related with continuations.

$$\begin{array}{rcl} [\![\mathtt{ret}]\!] & \equiv & \eta, \\ [\![m\,;k]\!] & \equiv & [\![m]\!]\star[\![k]\!], \\ [\![\lambda x.b]\!] & \equiv & x \mapsto [\![b]\!], \\ [\![ab]\!] & \equiv & [\![a]\!][\![b]\!]. \end{array}$$

Here, $x \mapsto t$ represents an arrow in $\mathcal{M}$ in the form of generalized elements, or equivalently, an arrow in the polynomial categories[11] in x.

As for terms of Λ,the interpretation is given in $\mathcal{M}$. Note that when we consider a continuation as a value of Λ, we must consider curried terms of continutations. In this case, definitions of abstraction and appllication must be modified as:

$$\begin{array}{rcl} [\![\lambda x.b]\!] & \equiv & curry(x \mapsto [\![b]\!]), \\ [\![ab]\!] & \equiv & uncurry([\![a]\!])[\![b]\!], \end{array}$$

respectively.

Theorem 1 The interpretation $[\![-]\!]$ is sound in the sense that if $t \longrightarrow s$, then $[\![t]\!] = [\![s]\!]$.

Proof

We prove the three laws of reductions.

1. As for

$$\frac{c_0 \longrightarrow c_0'}{c_0\,;c \longrightarrow c_0'\,;c,}$$

obviously $[\![c_0\,;c]\!] = [\![c_1\,;c]\!]$ if $[\![c_0]\!] = [\![c_1]\!]$.

2. As for the rule:

$$\frac{}{c_0\,;(c_1\,;c_2) \longrightarrow (c_0\,;c_1)\,;c_2,}$$

$$\begin{array}{rcl} [\![c_0\,;(c_1\,;c_2)]\!] & = & [\![c_0]\!]\star([\![c_1]\!]\star[\![c_2]\!]) \\ & = & ([\![c_0]\!]\star[\![c_1]\!])\star[\![c_2]\!](\text{Associativity}) \\ & = & [\![(c_0\,;c_1)\,;c_2)]\!]. \end{array}$$

3. As for

$$\frac{}{(\mathtt{ret}\ v)\,;c \longrightarrow cv,}$$

$$\begin{array}{rcl} [\![(\mathtt{ret}\ v)\,;c]\!] & = & \eta[\![v]\!]\star[\![c]\!] \\ & = & [\![c]\!][\![v]\!](\text{Right Unit}) \\ & = & [\![cv]\!] \end{array}$$

□

The following theorem shows the relation between syntactically-structural terms and semantically-structural arrows.

Proposition 3 A term is syntactically-structural if and only if its interpretation is semantically-structural.

Proof

Straightforward. □

We have now obtained, categorically, a subclass of continuations:

$$\begin{tikzcd}
\text{Structural Continuation} \arrow[r, Leftrightarrow] \arrow[d, hook, "\textit{extension}"'] & \text{Monad} \arrow[d, hook, "\textit{extension}"'] \\
\textit{An}\ \ \mathcal{SCONT} \arrow[r, Leftrightarrow] & \text{An Extended Monad?}
\end{tikzcd}$$

From this proposition, we use simply *"structural"* for both syntactically-structural terms and semantically-structural arrows if there is no fear of confusion.

For the analysis of the fullset continuations, the general theory of monads is not enough. We need a specific monad and a non-monadic arrows. In this sense, the monad theory gives a core theory of continuations (i.e. the theory of structural continuations), but the fullset continuation in which, for example, `call/cc` can be discussed must be studied in a specific monad such as CPS-monad and related arrows[17].

5 λ-Calculus and Continuation for Parallel Computing $\times$

In this and the next section, we define $\mathcal{PCONT}$. Parallel continuations with a new constructor $\langle -, - \rangle$are defined.

5.1 $\Lambda^{\times}$

We extend Λ and $\mathcal{SCONT}$ in order to express the parallel computing.

Definition 15 $[\Lambda^{\times}]$

$\Lambda^{\times}$ is the λ-calculus with finite products($\times$). We introduce the constructor (-,-) for the products.

$$\begin{array}{c}
\Lambda \subset \Lambda^{\times} \\
(t_0, \cdots, t_n) \in \Lambda^{\times} \quad (t_i \in \Lambda^{\times})
\end{array}$$

In [11], it is denoted by the λ-calculus with (surjective) pairing.

Example 4 The two projection terms are represented as $\lambda(t_0, t_1).t_0$ and $\lambda(t_0, t_1).t_1$ respectively.

5.2 $\mathcal{PCONT}$

$\mathcal{PCONT}$ is defined as the continuations for $\Lambda^{\times}$. We add the constructor and rules for the computation of products.

Definition 16 $[\mathcal{PCONT}]$

$$\begin{array}{ll}
\mathcal{SCONT} \subset \mathcal{PCONT}, & \\
\langle c_0, \cdots, c_n \rangle \in \mathcal{PCONT} & (c_i \in \mathcal{PCONT}), \\
\langle c_0, \cdots, a_n \rangle \in \mathcal{ANS} & (a_i \in \mathcal{ANS}),
\end{array}$$

with the following reduction rules:

$$\frac{c_0 \longrightarrow c_0' \qquad \cdots \qquad c_n \longrightarrow c_n'}{\langle c_0, \cdots, c_n \rangle \longrightarrow \langle c_0', \cdots, c_n' \rangle}(\times\text{-}1)$$

$$\frac{}{\langle \mathtt{ret}v_0, \cdots, \mathtt{ret}v_n \rangle \longrightarrow \mathtt{ret}(v_0, \cdots, v_n)}(\times\text{-}2)$$

$$\frac{}{\langle c_0, \cdots, c_n \rangle v \longrightarrow \langle c_0 v, \cdots c_n v \rangle.}(\times\text{-}3)$$

The $\mathtt{ret}$ is used in the synchronization in the sense that Multiple continuations run in parallel($\times$-1), and returns at the same time($\times$-2).

Lemma 3 [Projection Lemma]

If continuations $c_i \in \mathcal{PCONT}(i = 0, 1)$ are returning, there exist a projection continuation $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$ such that $\langle c_0, c_1\rangle ; \boldsymbol{\pi}(\boldsymbol{\pi}')$ and $c_0(c_1)$ reduce to the same answers respectively.

Proof

Let $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$ be $\lambda(t_0, t_1).\mathtt{ret}t_0$ and $\lambda(t_0, t_1).\mathtt{ret}t_1$ respectively. Let $c_iv \longrightarrow \mathtt{ret}w_i^v$.

Then $((\langle c_0, c_1\rangle ; \boldsymbol{\pi})v \longrightarrow \langle \mathtt{ret}w_0^v, \mathtt{ret}w_1^v\rangle ; \boldsymbol{\pi} \longrightarrow \mathtt{ret}w_0^v$. On the other hand, $((\langle c_0, c_1\rangle ; \boldsymbol{\pi}')v \longrightarrow \langle \mathtt{ret}w_0^v, \mathtt{ret}w_1^v\rangle ; \boldsymbol{\pi}' \longrightarrow \mathtt{ret}w_1^v$.

□

Because Λ is extended to $\Lambda^\times$, it is natural to extend the definition of syntactic structurality accordingly.

Definition 17 We extend the definition of syntactic structurality as follows:

1. A term t is also syntactically structural if it is syntactically structural on $\Lambda^\times$ in the sense of Section 3,

2. $\boldsymbol{\pi}$ and $\boldsymbol{\pi}'$ are structural.

3. if $c_0, \cdots, c_n$ are structural, then $\langle c_0, \cdots, c_n\rangle$ is structural.

5.3 Equivalence between $\mathcal{PCONT}$s

Before discussing the expressive power of $\mathcal{PCONT}$, we define a coarse relation on $\mathcal{PCONT}$.

Definition 18 [Equivalence]

We say that two continuations c_0 and c_1 are product-equivalent and denoted as $c_0 \sim c_1$ if for every value v, $c_0 ; \boldsymbol{\pi}(')$ and $c_1 ; \boldsymbol{\pi}(')$ return the same value respectively.

Optimization is a typical application of our algebra of extended continuations. Program transformation by algebraic manipulations of continuation is successful in optimizing codes[1]. The product-equivalence is a candidate for the algebra of optimizing program transformation.

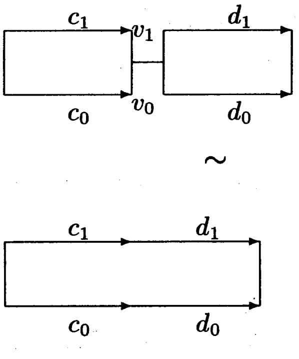
We can easily prove the following basic properties of products.

Lemma 4 Two continuations $c;\langle c_0, c_1\rangle$ and $\langle c;c_0, c;c_1\rangle$ are product-equivalent if c is returning.

Lemma 5 Two continuations $\langle\langle c_0, c_1\rangle ; \boldsymbol{\pi}, \langle c_0, c_1\rangle ; \boldsymbol{\pi}'\rangle$ and $\langle c_0, c_1\rangle$ are product-equivalent.

Two product-equivalent continuations do not necessarily have same synchronization points, or do not have the same number of "threads."

Example 5 Two continuations $\langle c_0, c_1\rangle ; \lambda(v_0, v_1).\langle d_0v_0, d_1v_1\rangle$ and $\langle c_0 ; d_0, c_1 ; d_1\rangle$ are product equivalent.



The parallel continuation represents the heterogeneous parallel computation. The rule ($\times$-1) expresses that computation proceeds in parallel. On the other hand, the rule ($\times$-2) represents the synchronization of parallel computations.

5.4 Representation of Synchronous Message Passing in $\mathcal{PCONT}$

The rule ($\times$-2) is interpreted as a barrier synchronization. The continuation $\langle c_0, \cdots, c_n\rangle$ returns first when all c_i's ($i = 0, \cdots, n$) return. Moreover, we can express the synchronous message passing style communication.

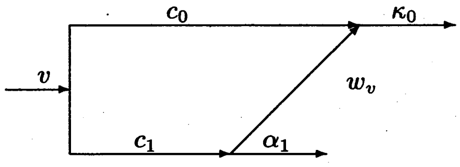
Example 6 [Synchonous Message Passing]

Let $c_0 v \longrightarrow \mathtt{ret}(\kappa_0)$ and $c_1 v \longrightarrow \mathtt{ret}(w_v, \alpha_1)$. Then, synchonous message passing between c_0 and c_1 is expressed as

$$\langle c_0, c_1\rangle v\,;\lambda(t_0,(t_1,t_2)).\langle t_0 t_1, t_2\rangle.$$

Let us observe the reduction of the above term.

$$\begin{array}{ll} & (\langle c_0, c_1\rangle v\,;\lambda(t_0,(t_1,t_2)).\langle t_0 t_1, t_2\rangle) \\ \longrightarrow & \mathtt{ret}(\kappa_0,(v,\alpha_1))\,;\lambda(t_0,(t_1,t_2)).\langle t_0 t_1, t_2\rangle \\ \longrightarrow & \langle \kappa_0 w_v, \alpha_1\rangle \end{array}$$



This means that c_0 proceeds computation and returns the value that proceeds computation as the continuation κ_0 , and that c_1 proceeds computation and returns the value that sends value w_v, and α_1, the rest of the compuation. Finally, κ_0 proceeds computation with the value w_v obtained from c_0. That is, the term represents that an answer(message) from $c_1(=w_v)$ is passed to (the descendant of) c_0.

Definition 19 Let two continuations c_0 and c_1 be given. Then

$$SMP(c_0, c_1) \equiv (\langle c_0, c_1\rangle\,;\lambda(t_0,(t_1,t_2)).\langle t_0 t_1, t_2\rangle)$$

SMP denotes "the Synchonous Message Passing."

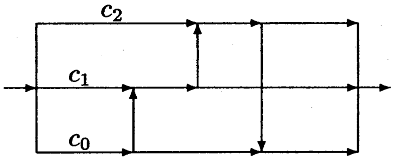
In fact,

Corolloary 1 SMP is not syntactically structural.

More than three continuations can pass messages. The following example is commonly observed in the representation of "delegation."

Example 7 [Message Passing among more than three Continuations]

Consider the following diagram.



Vertical lines represent message passing between c_0, c_1, and c_2 and their succedents. Then, it can be expressed as:

$$\begin{array}{l} \mathtt{deleg} \equiv \\ \quad \lambda v.\langle c_0 v, c_1 v\rangle\,; \\ \qquad \lambda((t_0,t_1),t_2).(\langle t_2 t_1, c_2 v\rangle\,; \\ \qquad\quad \lambda((s_0,s_1),s_2).(\langle s_2 s_1, t_0\rangle\,; \\ \qquad\qquad \lambda((u_0,u_1),u_2).\langle u_2 u_1, s_0, u_0\rangle)). \end{array}$$

If the reduction system allows the rule:

$$\frac{t \longrightarrow s}{\lambda x.t \longrightarrow \lambda x.s},$$

the reduction of deleg can be "optimized" to the one that three continuations are activated at the same time.

6 Interpretations of $\mathcal{PCONT}$ by Sync Monoidal Monads

In this section, we study monads on monoidal categories as models of parallel continuations. The notations of this section is based on [6].

Definition 20 [Monoidal Category] A monoidal category $\mathcal{V} \equiv (\mathcal{V}_0, \otimes, I, r, l, a)$ consists the following data:

1. a cateogry $\mathcal{V}_0$;
2. a functor $\otimes : \mathcal{V}_0 \times \mathcal{V}_0 \longrightarrow \mathcal{V}_0$;
3. an object I of $\mathcal{V}$;
4. a natural isomorphism $r \equiv r_A : A \otimes I \longrightarrow A$;
5. a natural isomorphism $\ell \equiv \ell_A : I \otimes A \longrightarrow A$;
6. a natural isomorphism $a \equiv a_{ABC} : (A \otimes B) \otimes C \longrightarrow A \otimes (B \otimes C)$,

with the following commutative diagrams:

$$\begin{tikzcd}
(A \otimes I) \otimes B \arrow[rr, "a"] \arrow[dr, "r \otimes 1"'] & & A \otimes (I \otimes B) \arrow[dl, "1 \otimes \ell"] \\
& A \otimes B &
\end{tikzcd}
\qquad
\begin{tikzcd}
((A \otimes B) \otimes C) \otimes D \arrow[r, "a"] \arrow[d, "a \otimes 1"'] & (A \otimes B) \otimes (C \otimes D) \arrow[r, "a"] & A \otimes (B \otimes (C \otimes D)) \\
A \otimes (B \otimes C)) \otimes D \arrow[rr, "a"'] & & A \otimes ((B \otimes C) \otimes D) \arrow[u, "1 \otimes a"']
\end{tikzcd}$$

Example 8 [Cartesian Category]

If $\mathcal{V}_0$ admits finite products, it is a monoidal cateogry $\mathcal{V} \equiv (\mathcal{V}_0, \times, 1, r, \ell, a)$. We call such a category cartesian.

Definition 21 [Monoidal Functor]

Let two monoidal categories $\mathcal{V} = (\mathcal{V}_0, \otimes, I, r, \ell, a)$ and $\mathcal{V}' = (\mathcal{V}'_0, \otimes', I', r', \ell', a')$ be given. A monoidal functor $F \equiv (f, \tilde{f}, f^0)$ consists of

1. a functor $f : \mathcal{V}_0 \longrightarrow \mathcal{V}'_0$;
2. a natural transformation $\tilde{f} \equiv \tilde{f}_{AB} : fA \otimes fB \longrightarrow f(A \otimes B)$;
3. a morphism $f^0 : I' \longrightarrow fI$,

with the following commutative diagrams:

$$\begin{tikzcd}
fI \otimes fA \arrow[r, "\tilde{f}"] & f(I \otimes A) \arrow[d, "f\ell"] \\
I' \otimes fA \arrow[u, "f^0 \otimes 1"] \arrow[r, "\ell'"'] & fA
\end{tikzcd}
\qquad
\begin{tikzcd}
fA \otimes fI \arrow[r, "\tilde{f}"] & f(A \otimes I) \arrow[d, "fr"] \\
fA \otimes I' \arrow[u, "1 \otimes f^0"] \arrow[r, "r'"'] & fA
\end{tikzcd}$$

$$\begin{tikzcd}
f((A \otimes B) \otimes C) \arrow[r, "fa"] & f(A \otimes (B \otimes C)) \\
f(A \otimes B) \otimes fC \arrow[u, "\tilde{f}"] & fA \otimes f(B \otimes C) \arrow[u, "\tilde{f}"'] \\
(fA \otimes fB) \otimes fC \arrow[u, "\tilde{f} \otimes 1"] \arrow[r, "a'"'] & fA \otimes (fB \otimes fC) \arrow[u, "1 \otimes \tilde{f}"']
\end{tikzcd}$$

We sometimes omit a, r, ℓ components if the definitions are clear.

Definition 22 We denote the arrow:

$$\widetilde{f} \equiv \widetilde{f}_{AB} : fA \otimes fB \longrightarrow f(A \otimes B)$$

by the sync-structure of f.

Definition 23 [Monoidal Natural Transformation]

Let monoidal functors $F \equiv (f, \widetilde{f}, f^0), G \equiv (g, \widetilde{g}, g^0) : \mathcal{V} \longrightarrow \mathcal{V}'$ be given. A monoidal natural transformation $\nu : F \longrightarrow G : \mathcal{V} \longrightarrow \mathcal{V}'$ consists of a natural transformation $\nu : f \longrightarrow g : \mathcal{V}_0 \longrightarrow \mathcal{V}_0'$ satisfying the following two commutative diagrams:

$$\begin{tikzcd}
I' \arrow[r, "f^0"] \arrow[dr, "g^0"'] & fI \arrow[d, "\nu_I"] \\
 & gI
\end{tikzcd}
\qquad
\begin{tikzcd}
fA \otimes fB \arrow[r, "\widetilde{f}"] \arrow[d, "\nu \otimes \nu"'] & f(A \otimes B) \arrow[d, "\nu"] \\
gA \otimes gB \arrow[r, "\widetilde{g}"'] & g(A \otimes B),
\end{tikzcd}$$

Definition 24 [Cartesian Funcor and Cartesian Natural Transformation]

A monoidal functor $F \equiv (f, \widetilde{f}, f^0)$ between caresian categories is caretesian when $\widetilde{f} : fA \times fB \longrightarrow f(A \times B)$ is an isomorphism.

A monoidal natural transformation is a cartesian natural transformation when it is monoidal between cartesian functors.

We consider a monad $M : \mathcal{V} \longrightarrow \mathcal{V}$ where $\mathcal{V}$ is a monoidal category.

Traditionally, we take CCM+NNO[11] as a model of λ-calculus with pairs and natural numbers. In CCM, we have a pair(product) of two terms, and, more importantly, projections of two terms. Terms can be decomposed by using projections.

In modelling parallel continuations, we do not need each component of parallel computations, but we only need the returned value of the computations as `ret`-synchronizations. This implies that in modelling the domain of values, we need a cartesian category, while in modelling the domain of continuations we relax the requirement to a monoidal category.

In the rest of this paper, we study the monad of the following type.

$$M : \mathcal{V} \longrightarrow \mathcal{V}'$$

with $\mathcal{V} \equiv (\mathcal{V}_0, \times, 1)$, a cartesian cateogry with $\times$, and $\mathcal{V}' \equiv (\mathcal{V}_0, \otimes, M1)$, a monoidal category.

Moreover we require that $M \equiv (M, sync^M, id)$ is a monoidal functor, and η be a monoidal natural transformation.

Specifically,

1. $M : \mathcal{V}_0 \longrightarrow \mathcal{V}_0$, a monad,
2. as $\widetilde{M}$, the sync-structure

$$sync^M \equiv sync^M_{A,B} : MA \otimes MB \longrightarrow M(A \times B)$$

is defined,

3. as M^0, $id : M1 \longrightarrow M1$ is given,

Moreover, we require that η preserves the sync structure in the following sense.

Definition 25 [Sync monoidal monad]

Given two monads $I : \mathcal{V} \longrightarrow \mathcal{V}$, and $M : \mathcal{V} \longrightarrow \mathcal{V}'$, and arrows $c : A \longrightarrow MC$ and $d : B \longrightarrow MD$, an arrow $c \bullet d : A \times B \longrightarrow MC \otimes MD$ is defined and the following diagram commutes:

$$\begin{tikzcd}
A \times B \arrow[r, "="] \arrow[d, "\eta_A \bullet \eta_B"'] \arrow[dr, phantom, "\bigcirc"] & A \times B : \arrow[d, "\eta_{A \times B}"] & I \arrow[d, "\bullet"] \\
MA \otimes MB \arrow[r, "sync"'] & M(A \times B) : & M
\end{tikzcd}$$

We denote such η by sync monoidal monad unit.
Moreover, the following diagram commutes:

$$\begin{tikzcd}
A\times B \arrow[r, "\eta\times\eta"] \arrow[dr, "\eta\bullet\eta"'] & MA\times MB \arrow[d, "id\bullet id"] \\
 & MA\otimes MB
\end{tikzcd}$$

As for funtoriality, we require that $(f\otimes g)\cdot(c\bullet d)=(fc\bullet gd)$.

Definition 26 If the natural transformation η satisfies the conditions above, we say that it is a sync-monoidal.

Lemma 6 Given $c:A\longrightarrow MC$ and $d:B\longrightarrow MD$, the following diagram commutes:

$$\begin{tikzcd}
A\times B \arrow[r, "c\times d"] \arrow[d, "\eta\bullet\eta"'] \arrow[dr, phantom, "\bigcirc"] & MC\times MD \arrow[d, "id\bullet id"] \\
MA\otimes MB \arrow[r, "c^*\otimes d^*"'] \arrow[d, "sync"'] \arrow[dr, phantom, "\bigcirc"] & MC\otimes MD \arrow[d, "sync"] \\
M(A\times B) \arrow[r, "(sync\cdot(c\bullet d))^*"'] & M(C\times D)
\end{tikzcd}$$

Let $f:A\longrightarrow B$ and $g:A\longrightarrow C$ be given. We denote by $(f,g):A\longrightarrow B\times C$ the usual pair of arrows. Let us define the diagonal as follows.

Definition 27 We define $diag_A:A\longrightarrow MA\otimes MA$ by the following diagram:

$$\begin{tikzcd}
A \arrow[r, "(id,id)"] \arrow[dr, "diag_A"'] & A\times A \arrow[d, "\eta\bullet\eta"] \\
 & MA\otimes MA
\end{tikzcd}$$

Immediately, we have

Corolloary 2 The following diagram commutes:

$$\begin{tikzcd}
A \arrow[r, "diag_A"] \arrow[dr, "\eta"'] & MA\otimes MA \arrow[d, "sync"] \\
 & M(A\times A)
\end{tikzcd}$$

Definition 28 Let $f:MA\longrightarrow MB$ and $g:A\longrightarrow MC$ be given. We denote by $<f,g>:A\longrightarrow MB\otimes MC$ the composition:

$$\begin{tikzcd}
A \arrow[r, "(id,id)"] \arrow[drr, "<f,g>"'] & A\times A \arrow[r, "\eta\bullet\eta"] & MA\otimes MA \arrow[d, "f^*\otimes g^*"] \\
 & & MB\otimes MC
\end{tikzcd}$$

It is naturally extended to the case of more than three arrows.

Definition 29 We define the sync monoidal monad on $\mathcal{V}$ as the monoidal monad $M \equiv (M, sync, id) : (\mathcal{V}_0, \times, 1, r, \ell, a) \longrightarrow (\mathcal{V}_0, \otimes, M1, r, \ell, a)$ if it has a sync monoidal unit.

The aim in this paper is to construct a model of $\mathcal{PCONT}$. We first give the interpretation $\llbracket - \rrbracket$.

Definition 30 Given a monoidal category $\mathcal{V}$ and a sync monoidal monad M, we define the interpretation $\llbracket - \rrbracket$ as follows;
as for domains,

$$\begin{array}{lcl} \mathcal{ANS} & \longmapsto & \bigcup_{a \in Obj} Ma \\ \mathcal{SCONT} & \longmapsto & \bigcup_{a,b \in Obj}(a \longrightarrow Mb) \\ \mathcal{PCONT} & \longmapsto & \bigcup_{a,a_0,\cdots,a_n \in Obj}(a \longrightarrow Ma_0 \otimes \cdots \otimes Ma_n) \end{array}$$

where $a_0 \otimes \cdots a_n$ is the abbreviation of $(a_0 \otimes \cdots a_{n-1}) \otimes a_n$ as usual;
and as for the interpretation of terms of $\mathcal{PCONT}$,

$$\begin{array}{rcl} \llbracket \mathbf{ret} \rrbracket & \equiv & \eta \\ \llbracket \langle f, \cdots, g \rangle \rrbracket & \equiv & \begin{cases} < \llbracket f \rrbracket, \cdots, \llbracket g \rrbracket > & (\langle f, \cdots, g \rangle \in \mathcal{PCONT}) \\ sync < \llbracket f \rrbracket, \cdots, \llbracket g \rrbracket > & (\langle f, \cdots, g \rangle \in \mathcal{ANS}) \end{cases} \\ \llbracket c; d \rrbracket & \equiv & \begin{cases} sync(\llbracket c \rrbracket) \star \llbracket d \rrbracket & (c \in \mathcal{PCONT}) \\ \llbracket c \rrbracket \star \llbracket d \rrbracket & (c \in \mathcal{ANS}) \end{cases} \end{array}$$

where $< t_0, \cdots, t_n >$ is the abbreviation of
$<< t_0, \cdots, t_{n-1} >, t_n >$.

A problem lies in the interpretation of $\mathcal{PCONT}$. A term of $\mathcal{PCONT}$sometimes reduces to a term of $\mathcal{SCONT}$. For example, in (×-2), the LHS is in $\mathcal{PCONT}$, while the RHS is in $\mathcal{SCONT}$. We implicitly insert *sync*, and coerce the "type."

Theorem 2 Let $\mathcal{V}$ be both cartesian and monoidal, and M be a sync monoidal monad. Then the interpretation$\llbracket - \rrbracket$ is sound in the sense that if $t \longrightarrow s$, then $\llbracket t \rrbracket = \llbracket s \rrbracket$.

Proof

We only prove that the theorem holds in the three rules (×-1,2,3).

(×-1) As for the rule

$$\frac{c_0 \longrightarrow c_0' \quad \cdots \quad c_n \longrightarrow c_n'}{\langle c_0, \cdots, c_n \rangle \longrightarrow \langle c_0', \cdots, c_n' \rangle},$$

If $\llbracket c_i \rrbracket = \llbracket c_i' \rrbracket (\mathrm{i} = 0 \cdots \mathrm{n})$, then obviously
$< \llbracket c_0 \rrbracket, \cdots, \llbracket c_n \rrbracket > = < \llbracket c_0' \rrbracket, \cdots, \llbracket c_n' \rrbracket >$.

(×-2) As for the rule

$$\frac{}{\langle \mathbf{ret} v_0, \cdots, \mathbf{ret} v_n \rangle \longrightarrow \mathbf{ret}(v_0, \cdots, v_n)},$$

$\llbracket \langle \mathbf{ret} v_0, \cdots, \mathbf{ret} v_n \rangle \rrbracket = sync < \eta \llbracket v_0 \rrbracket, \cdots, \eta \llbracket v_n \rrbracket >$
because LHS: $\mathcal{ANS}$.

$$\begin{array}{rcl} sync < \eta \llbracket v_0 \rrbracket, \cdots, \eta \llbracket v_n \rrbracket > & = & \eta(\llbracket v_0 \rrbracket, \cdots, \llbracket v_n \rrbracket) \\ & = & \llbracket \mathbf{ret}(v_0, \cdots, v_n) \rrbracket \end{array}$$

(×-3) As for the rule

$$\frac{}{\langle c_0, \cdots, c_n \rangle v \longrightarrow \langle c_0 v, \cdots c_n v \rangle.}$$

$$\begin{array}{rcl} \llbracket \langle c_0 v, \cdots, c_n v \rangle \rrbracket & = & < \llbracket c_0 v \rrbracket, \cdots, \llbracket c_n v \rrbracket > \\ & = & (\llbracket c_0 \rrbracket \bullet \cdots \bullet \llbracket c_n \rrbracket) \cdot (\llbracket v \rrbracket, \cdots, \llbracket v \rrbracket) \\ & = & < \llbracket c_0 \rrbracket, \cdots, \llbracket c_n \rrbracket > \cdot \llbracket v \rrbracket. \end{array}$$

□

The discussion of structurality of $\mathcal{SCONT}$is naturally extended to $\mathcal{PCONT}$.

Proposition 4 A term is syntactically-structural if and only if its interpretation is semantically-structural.

Proof

Straightforward. □

In particular, the synchronization is structural in the above sense.

Corolloary 3 $SMP(c_0, c_1)$ is structural.

Proof

$$\begin{aligned}&[\![SMP(c_0, c_1)]\!] = \\ &\quad (\mathit{sync}\cdot < [\![c_0]\!], [\![c_1]\!] >)\star < \mathit{uncurry}(\pi_0) \cdot \pi_{10}, \pi_{11} >\end{aligned}$$

□

This reflects the fact that message passing is essential in the concurrency.

We have now obtained a parallel version of categorical hierarchy of continuations:

$$\begin{tikzcd} \text{Structural Continuation} \arrow[r, Leftrightarrow] \arrow[d, hook] & \text{Sync Monoidal Monad} \arrow[d, hook] \\ A \quad \mathcal{PCONT} \arrow[r, Leftrightarrow] & \text{An Extended Monad?} \end{tikzcd}$$

Note that sync monoidality applies to both the general monad which models s$\mathcal{SCONT}$and the CPS-monad which models $\mathcal{SCONT}$, the fullset continuations. In this sense, our discussion of sync monoidality is orthogonal to the discussion of $\mathcal{SCONT}$'s expressive power. However, to analyze the fullset parallel continuations, our sync monoidal monad is not enough. This situation is parallel to the fullset sequential continuations.

However, our sync monoidal monad can represent the parallelly running continuations and the synchronous message passing. This is expressive enough for the core theory of parallelism.

7 Cartesian Monad as a Model of Parallel Continuations

In this section, we consider the case when a monoidal structure is strengthened to cartesian structure between cartesian structure and monoidal structure.

Definition 31 We say that a monoidal monad M is cartesian if

1. $M : (\mathcal{V}_0, \times, id) \longrightarrow (\mathcal{V}_0, \times, id)$,
2. M preserves $\times$, and
3. *sync* is an iso:

$$MA \times MB \xrightarrow{sync} M(A \times B) = MA \times MB$$

Proposition 5 If a sync monoidal monad M is cartesian, we may take $f \bullet g$ as $f \times g$.

Proof

Obvious. □

In a cartesian monad model, because *sync* is iso, the interpretation of parallel continuations given in Definition 30 is reduced to the interpretation of sequential continuations given in Definition 14 with the interepretation of pairing of $\Lambda^\times$.

If the monad is cartesian, every arrow to an object $MA \times MB$ ($\mathcal{PCONT}$) is isomorphic to an arrow to $M(A \times B)$, which is then in $\mathcal{SCONT}$.

In Section 5, we have showed that the relation $\sim$ can express the computed value, though the parallelism, synchronization, and other concepts of parallelism cannot be expressed. From the viewpoint of semantics, this is expressed in that the monoidal structure $\otimes$ is degenerated to the cartesian structure $\times$. In this sense, pairing in the domain of values is expressed as $\times$, while pairing in the domain of computations is expressed as $\otimes$.

Example 9 The CPS-monad is, in general, not cartesian. The identity monad is cartesian.

8 Concluding Remarks

In this paper, we algebraically reformulated sequential continuations with ret, anonymous returns and composition. By these constructs, we presented, categorically, a subclass of continuations, and showed that a core class of continuations are modelled in monads.

Moreover, we defined a parallel construct $\langle -, - \rangle$ of continuations. We represented the synchonous message passing using this construct. This implies that $\langle -, - \rangle$ is expressive enough to represent basic concepts of parallelism. We also showed that this can be modelled in the sync monoidal monads.

The discussion of this paper implies that in order to study the full strength sequential continuations, we need a stronger structure than monads as already shown in [17], but that monads are enough for representing basic concepts of parallel continuations.

References

[1] Appel, A.W., Compiling with Continuations, 1992.

[2] Appel, A.W. and Jim, T., *"Continuation-Passing, Closure-Passing Style,"* Proc. 1989 Principles of Programming Languages, 1989, pp. 293–302.

[3] Baeton, J.C.M and Weijland, W.P., Process Algebra, 1990.

[4] Danvy, O. and Filinski, A., *"Abstracting Control,"* Proc. 1990 Lisp and Functional Programming, 1990, pp. 151–160.

[5] Duba, B.,Harper, R., and MacQueen, D., *"Typing first-class continuations in ML,"* Journal of Functional Programming, Vol.3, 1993, pp. 465–484.

[6] Eilenberg, S. and Kelly, G.M., *"Closed Categories"* Proc. Conf. Categorical Algebra(La Jolla 1965), 1966.

[7] Felleisen, M. Wand, M., Friedman, D. and Duba, B., *"Abstract Continuations:A Mathematical Semantics for Handling Full Functional Jumps,"* Proc. 1988 Lisp and Functional Programming, 1988, pp. 52–62.

[8] Felleisen, M., *"$\lambda$-V-CS: An Extended $\lambda$-Calculus for Scheme,"* Proc. 1988 Lisp and Functional Programming, 1988, pp. 72–85.

[9] Filinski, A., *"Representing Monads,"* Proc. 1994 Principles of Programming Languages, 1994, pp. 446–457.

[10] Hoare, C.A.R, Communicating Sequential Processes, 1985.

[11] Lambek, J. and Scott, P., Introduction to Higher Order Categorical Logic, 1986.

[12] Milner R., Communication and concurrency, 1989.

[13] Moggi, E., *"Notions of Computation and Monads,"* Information and Computation, vol. 93, 1991, pp. 55–92.

[14] Reed, J., Clinger, W., and et.al., *"Revised report on the algorithmic language Scheme,"* SIGPLAN Notices, Vol. 21, No. 12, 1986, pp. 37–79.

[15] Reppy, J.H., *"CML: A Higher-order Concurrent Language,"* Proc. 1991 Programming Language Design and Implementation, 1991, pp. 293–305.

[16] Troelstra, A.S. Lectures on Linear Logic, 1992.

[17] Wadler, P., *"Monads and Composable Continuations,"* LISP and Symbolic Computation, vol. 7, 1994, pp. 39–56.

[18] Wand, M., *"Continuation-Based Multiprocessing,"* Proc. 1980 Lisp and Functional Programming, 1980, pp. 19–28.