Axioms for Delimited Continuations in the CPS Hierarchy

Yukiyoshi Kameyama

Department of Computer Science, University of Tsukuba
Tennodai 1-1-1, Tsukuba, 305-8573, JAPAN
and
Japan Science and Technology Agency
kameyama@acm.org

Abstract. A CPS translation is a syntactic translation of programs, which is useful for describing their operational behavior. By iterating the standard call-by-value CPS translation, Danvy and Filinski discovered the CPS hierarchy and proposed a family of control operators, shift and reset, that make it possible to capture successive *delimited* continuations in a CPS hierarchy.
Although shift and reset have found their applications in several areas such as partial evaluation, most studies in the literature have been devoted to the base level of the hierarchy, namely, to level-1 shift and reset. In this article, we investigate the whole family of shift and reset. We give a simple calculus with level-n shift and level-n reset for an arbitrary $n > 0$. We then give a set of equational axioms for them, and prove that these axioms are sound and complete with respect to the CPS translation. The resulting set of axioms is concise and a natural extension of those for level-1 shift and reset.
Keywords: CPS Translations, Control Operators, Delimited Continuations, Axiomatization, Type System.

1 Introduction

A CPS translation transforms a source term into continuation-passing style (CPS for short). It can be regarded as a compilation step, since it makes explicit the evaluation order of the source program and gives names to intermediate results. Another motivating fact for CPS is that it makes it possible to represent various control mechanisms, such as `callcc` in Scheme and Standard ML of New Jersey, that give programmers first-class continuations in the source language.

Logically, a CPS translation for the simply typed lambda calculus is a double negation interpretation from classical logic into minimal logic, or Friedman's A-translation [12]. The control mechanisms added to the source language can be also understood logically. For instance, Griffin [13] has revealed the Curry-Howard correspondence between the calculus with `callcc` and classical logic.

Danvy and Filinski [7, 8] observed that there is room for a more refined control mechanism. By CPS translating the answer type of the standard CPS

translation, they obtained what they call a CPS hierarchy. Furthermore, they proposed a family of control operators `shift` and `reset` to abstract delimited continuations in this hierarchy. In the literature, many different control operators for delimited continuations have been proposed [10, 14–16]. In contrast to these other control operators, `shift` and `reset` are solely defined in terms of the CPS translation. In addition, they have found applications in partial evaluation [19], one-pass CPS translations [8], and normalization by evaluation [4], as well as to represent layered monads [11] and mobile computation [24].

In this article, we study a theoretical foundation of the control operators in the CPS hierarchy. Specifically, we address the problem of finding direct-style axioms for them. While these operators are used in many applications and their semantics is given by a CPS translation (be it iterated or extended), we often want to reason about source programs directly, rather than treating the image of CPS translations, since the CPS translation is sometimes said to obscure the overall structure of source programs. Also finding a good set of direct-style axioms could lead one to a better understanding of these operators.

We give a simple set of axioms consisting of only three equations for `shift` and three equations for `reset`, and then prove that this set of equations is sound and complete with respect to the iterated CPS translation. This work builds on our previous work, in which we gave a sound and complete axiomatization for level-1 `shift` and `reset` operators [18], and for level-2 [17]. Since completeness proofs of this kind often require quite a lot of calculations, we make the proof more structured by following an idea due to Sabry [22, 23] and reconstructing it in a type-theoretic setting, which further simplifies our proof.

Overview: The rest of this article is organized as follows. In Section 2, we informally introduce `shift` and `reset` and we explain their operational aspect. In Sections 3 and 4, we formally introduce the calculi with these control operators and a CPS translation for them. In Section 5 we present the axioms for control operators. In Section 6, we give a type-theoretic analysis of the CPS translation and we prove completeness. In Section 7, we conclude and mention future work.

Prerequisites: We assume that readers have some familiarity with CPS translations.

2 Control Operators in the CPS hierarchy

We introduce the `shift` and `reset` operators through some examples.

2.1 A Simple Example

The following example uses these operators in a simple way:

$$\begin{aligned} 3 \;+\; \langle 4 \;*\; \mathcal{S}(\lambda k.\; 5 \;+\; (k\;(k\;2)))\rangle &= \texttt{let}\ k\ \texttt{be}\ \lambda x.\langle 4 * x\rangle \\ &\quad\ \texttt{in}\ \ 3 \;+\; \langle 5 \;+\; (k\;(k\;2))\rangle \\ &= 3 \;+\; \langle 5 \;+\; \langle 4 \;*\; \langle 4 \;*\; 2\rangle\rangle\rangle \end{aligned}$$

where $\langle \_ \rangle$ is the reset operator and $\mathcal{S}$ is the shift operator.[1] Unlike the continuation captured by `callcc`, the continuation captured by $\mathcal{S}$ is not the whole rest of the computation (such as $3 + \langle 4 * [\ ] \rangle$), but a part which is delimited by a `reset`, that is, $\langle 4 * [\ ] \rangle$. Also it is not abortive, and thus we can compose the captured continuation with ordinary functions. When several occurrences of `reset` enclose an occurrence of `shift`, the (dynamically determined) closest one is chosen as the delimiter.

As more substantial examples, we borrow the ones by Danvy and Filinski [7].

2.2 Nondeterminism

A non-deterministic choice can be represented by backtracking in direct style using `shift` and `reset`:

$$\begin{aligned}
\texttt{flip}(x) &\stackrel{def}{=} \mathcal{S}_1(\lambda c.\ \texttt{begin}\ \ c(\texttt{true});\ c(\texttt{false});\ \texttt{fail}(_)\ \ \texttt{end}) \\
\texttt{fail}(x) &\stackrel{def}{=} \mathcal{S}_1(\lambda c.\ \texttt{"no"}) \\
\texttt{choice}(n) &\stackrel{def}{=} \texttt{if}\ n < 1\ \texttt{then}\ \texttt{fail}(_) \\
&\qquad \texttt{else if}\ \texttt{flip}(_)\ \texttt{then}\ \texttt{choice}(n-1) \\
&\qquad\quad \texttt{else}\ n
\end{aligned}$$

where $\_$ is a dummy value, `true`, `false` are truth values, and `begin`$\cdots$`end` is for sequencing.

To understand these programs, we CPS translate these three functions as:

$$\begin{aligned}
\texttt{flip-c}(x,k) &\stackrel{def}{=} \texttt{begin}\ \ k(\texttt{true});\ k(\texttt{false});\ \texttt{fail-c}(_,k)\ \ \texttt{end} \\
\texttt{fail-c}(x,k) &\stackrel{def}{=} \texttt{"no"} \\
\texttt{choice-c}(n,k) &\stackrel{def}{=} \texttt{if}\ n < 1\ \texttt{then}\ \texttt{fail-c}(_,\ k) \\
&\qquad \texttt{else}\ \texttt{flip-c}(_,\ \lambda y.\ \texttt{if}\ y\ \texttt{then}\ \texttt{choice-c}(n-1,k) \\
&\qquad\qquad\qquad\qquad \texttt{else}\ k(n))
\end{aligned}$$

Let us consider the program $\langle \texttt{display}(\texttt{choice}(3)) \rangle_1$. It is CPS translated to the program $\texttt{choice-c}(3, \texttt{display})$, which will display 1, 2 and 3 in this order. It is easy to see that `shift` captures the current continuation, which is composable with functions (including other continuations), and that `reset` installs the identity continuation.

2.3 Collecting Successive Results

As a next step, one may wants to collect all answers generated by non-deterministic choices. This is implemented by the function `emit` defined by:

$$\texttt{emit}(n) \stackrel{def}{=} \mathcal{S}_1(\lambda c.\ \texttt{cons}(n,\ c(\texttt{nil})))$$

For instance, $\langle \texttt{begin}\ \ \texttt{emit}(1);\ \texttt{emit}(2);\ \texttt{emit}(3)\ \ \texttt{end} \rangle_1$ will return a list (1 2 3).

[1] Danvy and Filinski used the notation $\xi k.M$ for $\mathcal{S}(\lambda k.M)$.

It is then natural to expect that a combined program $\langle \texttt{emit}(\texttt{choice}(3))\rangle_1$ would work. However it does not, since the control operators in the two programs interfere. To see this, let us CPS translate `emit` as:

$$\texttt{emit-c}(n, k) \stackrel{def}{=} \texttt{cons}(n,\ k(\texttt{nil}))$$

The term $\langle \texttt{emit}(\texttt{choice}(3))\rangle_1$ is CPS translated to $\texttt{choice-c}(3, \lambda x.\texttt{emit-c}(x, \lambda a.a))$, which will generate three lists (1), (2) and (3), but never collect these answers.

A correct way of combining these programs is to make them *layered*. The continuation captured in `emit` should be in a higher level than that captured in `choice`. To achieve this, the CPS counterpart of `emit` should be:

$$\texttt{emit-c2}(n, k, \gamma) \stackrel{def}{=} \texttt{cons}(n,\ \gamma(k(\texttt{nil})))$$

where γ is a level-2 continuation. Its direct-style counterpart is:

$$\texttt{emit-c1}(n, k) \stackrel{def}{=} k(\mathcal{S}_1(\lambda c.\ \texttt{cons}(n,\ c(\texttt{nil}))))$$

which passes a continuation, even though it is not in CPS since the argument of k is not a trivial term. Its direct-style counterpart is:

$$\texttt{emit}(n) \stackrel{def}{=} \mathcal{S}_2(\lambda c.\ \texttt{cons}(n,\ c(\texttt{nil})))$$

This is the point where we need a level-2 control operator in the CPS hierarchy. Executing the term $\langle \texttt{emit}(\texttt{choice}(3))\rangle_2$ returns (1 2 3) as expected.

2.4 Summary and Conclusion

In summary, a direct-style program with level-2 control operators is CPS translated to a 1-CPS program with level-1 control operators, which is then CPS translated to a 2-CPS program with no control operators. CPS translating this program yields a real CPS program where all calls are tail calls and all subterms are trivial. The family of layered control operators thus corresponds to the CPS hierarchy.

A similar situation occurs when we perform partial evaluation of a program using `shift`/`reset` when the partial evaluator itself uses `shift`/`reset`. We refer to the reader to Asai's recent work [1, 2].

3 The Calculi with Control Operators

In this section, we define the language of our calculi, and postpone giving axioms until the next section.

The calculus we choose here is a type-free lambda calculus with control operators for delimited continuations. Later we briefly mention simply typed calculi. We consider the call-by-value evaluation order only.

We shall define calculi $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$ for a natural number n. The former is a calculus with `shift`/`reset`, and the latter a calculus with $\mathcal{C}$/`reset`. The control

operator $\mathcal{C}$ has a slightly different semantics as `shift`, which will be explained later.

We first assume there are infinitely many variables (written x, y, z and so on). Terms of $\lambda\mathcal{S}_n$ are type-free lambda terms augmented with control operators, and defined by:

$$\text{(terms)} \quad M, N ::= x \mid \lambda x.M \mid MN \mid \langle M \rangle_i \mid \mathcal{S}_i$$

where $1 \leq i \leq n$. Terms of $\lambda\mathcal{C}_n$ are defined in the same way with $\mathcal{S}_i$ being replaced by $\mathcal{C}_i$.

The index i denotes the level, which is conceptually the number of iterations of CPS translations that are necessary to interpret the control operator. The construct $\langle \_ \rangle_i$ is level-i reset, and $\mathcal{S}_i$ is `shift`. Note that $\mathcal{S}_i$ (and $\mathcal{C}_i$) is a constant rather than a constructor. We use the abbreviations: $\mathcal{S}_i k.M \equiv \mathcal{S}_i(\lambda k.M)$, $\mathcal{C}_i k.M \equiv \mathcal{C}_i(\lambda k.M)$, and $\mathcal{A}_i \equiv \lambda x.\mathcal{C}_i(\lambda k.x)$. We also define $\langle M \rangle_0 \equiv M$.

A value (written V) is either a variable, λ-abstraction, or a constant ($\mathcal{S}_i$ in $\lambda\mathcal{S}_n$ and $\mathcal{C}_i$ in $\lambda\mathcal{C}_n$). Variables are bound by λ, and free and bound variables of terms are defined as usual. $FV(M)$ denotes the set of free variables in M. We identify two terms which differ only in renaming of bound variables. $M\{x := N\}$ is the result of the usual capture-avoiding substitution of N for x in M.

Contexts and evaluation contexts are defined as follows:

$$\begin{aligned} C &::= [\,] \mid CM \mid MC \mid \langle C \rangle_i \\ E &::= [\,] \mid EM \mid VE \mid \langle E \rangle_i \\ E^i &::= [\,] \mid E^i M \mid V E^i \mid \langle E^i \rangle_j \quad \text{for } j \leq i \end{aligned}$$

E is an evaluation context in call by value, and E^i is a level-i evaluation context in which the level of reset operators enclosing the hole must be equal to or smaller than i. For example, $\langle x[\,] \rangle_2$ and $\langle x[\,] \rangle_2 \langle yz \rangle_3$ are level-2 evaluation contexts. As a special case, E^0 is an evaluation context in which no reset may enclose the hole.

The operational semantics is given by the following rules (where $f \notin FV(E^{j-1})$):

$$\begin{aligned} E[\langle V \rangle_j] &\to E[V] \\ E[\langle E^{j-1}[\mathcal{S}_j V] \rangle_j] &\to E[\langle V(\lambda f.\langle E^{j-1}[f] \rangle_j) \rangle_j] \end{aligned}$$

The first rule says that delimiting a value does nothing. The second rule shows how the $\mathcal{S}$-operator works. It captures the continuation delimited by the (dynamically determined) closest `reset`-operator, as the evaluation context E^{j-1} does not contain a level-j reset operator which encloses the hole. Note that the continuation captured by $\mathcal{S}_j$ is a function, whose body is enclosed by a level-j reset operator. This is an essential difference between Danvy and Filinski's `shift` operator and Felleisen's `control` operator [10].

The rule above is only a special case of the general rule given below. As we explained before, the level of the corresponding reset operator can be higher than j. Therefore a general rule for the second line is (where $j \leq i$ and $f \notin FV(E^{j-1})$):

$$E[\langle E^{j-1}[\mathcal{S}_j V] \rangle_i] \to E[\langle V(\lambda f.\langle E^{j-1}[f] \rangle_j) \rangle_i]$$

The two operators $\mathcal{S}_i$ and $\mathcal{C}_i$ are inter-definable:

$$\mathcal{S}_i = \lambda z.\mathcal{C}_i(\lambda k.z(\lambda x.\langle kx\rangle_i))$$
$$\mathcal{C}_i = \lambda z.\mathcal{S}_i(\lambda k.z(\lambda x.\mathcal{S}_i(\lambda d.kx)))$$

These equations are formally justified by the CPS translation in the next section.

4 CPS Translation

The CPS translation we consider is due to Danvy and Filinski [7]. It translates terms of the source calculi ($\lambda\mathcal{S}_n$ or $\lambda\mathcal{C}_n$) to the type-free lambda calculus without control operators. As we explained in the introduction, their CPS translation can be thought of as a standard CPS translation followed by $n-1$ successive CPS translations of the answer type for $n > 1$. If we fix the number of iterations, then the whole translations can be expressed by a single, uncurried CPS translation, which we call an extended CPS translation. It takes n continuation parameters, each being introduced by the i-th CPS translation (for $1 \leq i \leq n$). This extended CPS translation gives a precise semantics to the level-i control operators (for $i < n$). The n continuation parameters are represented by the variables k_i, which we call a continuation variable of level i (for $1 \leq i \leq n$).

For a fixed n, and given a term M and a value V in the source calculi, we define two translations $[\![\_]\!]$ and $\_^*$, which send a term and a value to terms of type-free lambda calculus. To avoid clutter we present a $\beta\eta$-reduced version. In the following we assume $1 \leq i < n$.

$$\begin{aligned}
[\![V]\!] &\stackrel{def}{=} \lambda k_1.\ k_1 V^* \\
[\![MN]\!] &\stackrel{def}{=} \lambda k_1.\ [\![M]\!](\lambda m.[\![N]\!](\lambda n.mnk_1)) \\
[\![\langle M\rangle_i]\!] &\stackrel{def}{=} \lambda k_1.\cdots\lambda k_{i+1}.\ [\![M]\!]\theta_1\cdots\theta_i(\lambda x.\ \theta_0 x k_1 k_2\cdots k_{i+1})
\end{aligned}$$

$$\begin{aligned}
x^* &\stackrel{def}{=} x \\
(\lambda x.M)^* &\stackrel{def}{=} \lambda x.[\![M]\!] \\
\mathcal{S}_i{}^* &\stackrel{def}{=} \lambda x k_1\cdots k_i.\ x(\lambda y k_1'\cdots k_{i+1}'.\theta_0 y k_1 k_2\cdots k_i(\lambda z.\theta_0 z k_1' k_2'\cdots k_{i+1}'))\theta_1\cdots\theta_i \\
\mathcal{C}_i{}^* &\stackrel{def}{=} \lambda x k_1\cdots k_i.\ x(\lambda y k_1'\cdots k_i'.\theta_0 y k_1 k_2\cdots k_i)\theta_1\cdots\theta_i
\end{aligned}$$

where $\theta_i = \lambda x k_{i+1}.k_{i+1}x$. The term θ_i can be thought of as the image of the identity continuation (the empty evaluation context [5]) of level i.

Let us briefly explain the extended CPS translation. Terms and values without control operators are translated as usual. For the term $\langle M\rangle_i$, the reset operator installs identity continuations up to level i, and composes all continuations of up to level i with the continuation of level $i+1$. The operator $\mathcal{C}_i$ captures the current continuation up to level i, which results in $\lambda y k_1'\cdots k_i'.\theta_0 y k_1 k_2\cdots k_i$ in this context, then applies it to the argument. It also installs instances of the identity continuation up to level i. The CPS translation of $\mathcal{S}_i$ is slightly

more complex, since it captures a non-abortive delimited continuation so that we should compose $k'_1, \cdots, k'_n$ with the captured continuation. Note that the result of the extended CPS translation does not depend on n (if the result is defined).

We show a few examples of the extended CPS translation and $\beta\eta$-reductions in the target calculus:

$$\begin{aligned}
[\![\mathcal{C}_2(\lambda f.fx)]\!] &= \lambda k_1.(\lambda k_1.k_1\mathcal{C}_2{}^*)(\lambda m.(\lambda k_1.k_1(\lambda f.fx)^*)(\lambda n.mnk_1)) \\
&\to_{\beta\eta} \lambda k_1.\ k_1 x \\
[\![\langle xy\rangle_2]\!] &= \lambda k_1 k_2 k_3.\ [\![xy]\!]\theta_1\theta_2(\lambda z.\ \theta_0 z k_1 k_2 k_3) \\
&\to_{\beta\eta} \lambda k_1 k_2 k_3.\ xy\theta_1\theta_2(\lambda z.\ k_1 z k_2 k_3)
\end{aligned}$$

The semantics of the target terms is given by the standard $\beta\eta$-equality (the type-free lambda calculus with $\beta\eta$-equality will be denoted by $\lambda_{\beta\eta}$.) Given a CPS translation and the target theory $\lambda_{\beta\eta}$, the source calculus is given a rigid semantics (CPS semantics). The fundamental question addressed in this article is, what is the equality theory that coincides with this CPS semantics. We first give an answer to this question, and then prove it.

5 Axioms of $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$

We give axioms for the theories $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$. The common axioms for these theories are shown in Figure 1, the specific axioms for $\lambda\mathcal{S}_n$ are in Figure 2, and the specific axioms for $\lambda\mathcal{C}_n$ are in Figure 3. In the presentation of axioms, we assume the levels of all control operators are less than n, namely, $1 \le i, j < n$. For the purpose of comparison, we list the axioms for the base level control operators $\mathcal{S}_1$ and $\langle\_\rangle_1$ in Figure 4 which were given in our joint work with Hasegawa [18].

Recall that E^i is a level-i evaluation context, $\mathcal{S}_i k.M \equiv \mathcal{S}_i(\lambda k.M)$, $\mathcal{C}_i k.M \equiv \mathcal{C}_i(\lambda k.M)$, and $\langle M\rangle_0 \equiv M$. The last abbreviation is used when $i = 1$ in reset-lift-2 and others. (Note that $i - 1$ may be 0.)

Let us explain Figure 1. The first three axioms β_v, η_v and β_Ω are those for Moggi's computational lambda calculus, the canonical calculus in call-by-value [20]. The axiom reset-value is essentially the same as that in level-1 theory (Figure 4). The axioms reset-lift and reset-lift-2 lift a β-redex over a reset operator. The axiom β_Ω can be applied to the level-0 evaluation context only, but with the help of these axioms we can lift a β-redex over a general evaluation context. The axiom reset-lift is a natural extension of its level-1 counterpart, while no counterpart of reset-lift-2 exists in level-1 axioms. The axiom may look strange since the index j appears only in the right-hand side. We may restrict j to i in reset-lift-2 in the presence of the axiom $\langle\langle M\rangle_j\rangle_i = \langle M\rangle_i$ for $j \le i$.

Besides the common axioms, each theory has three specific axioms for $\mathcal{S}_i$ or $\mathcal{C}_i$. The axiom $\mathcal{S}$-reset is a natural extension of its level-1 counterpart, while the axiom $\mathcal{S}$-elim is not quite the same as a natural extension of its level-1 counterpart. In fact, $\mathcal{S}_i k.kN = N$ is not sound for $i > 1$. Danvy and Filinski [7] stated that the current formulation of `shift`/`reset` is not completely satisfactory

$$
\begin{array}{rll}
(\lambda x.M)V = M\{x := V\} & & \beta_v \\
\lambda x.\ Vx = V & & \eta_v, \text{ if } x \notin FV(V) \\
(\lambda x.E^0[x])M = E^0[M] & & \beta_\Omega, \text{ if } x \notin FV(E^0) \\
\langle V\rangle_i = V & & \text{reset-value} \\
\langle(\lambda x.M)\langle N\rangle_i\rangle_j = (\lambda x.\langle M\rangle_j)\langle N\rangle_i & & \text{reset-lift, } j \le i \\
\langle(\lambda x.M)\langle N\rangle_{i-1}\rangle_i = \langle(\lambda x.\langle M\rangle_j)\langle N\rangle_{i-1}\rangle_i & & \text{reset-lift-2, } j \le i
\end{array}
$$

Fig. 1. Common Axioms for $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$ $(1 \le i, j < n)$

$$
\begin{array}{rll}
\mathcal{S}_i k.\langle M\rangle_i = \mathcal{S}_i k.M & & \mathcal{S}\text{-reset} \\
\mathcal{S}_i k.k\langle M\rangle_{i-1} = \langle M\rangle_{i-1} & & \mathcal{S}\text{-elim, if } k \notin FV(M) \\
\langle E^j[\mathcal{S}_i k.M]\rangle_i = \langle M\{k := \lambda f.\ \langle E^j[f]\rangle_i\}\rangle_i & & \mathcal{S}\text{-lift} \\
 & & \text{if } k \notin FV(E^j),\ f \notin FV(kE^j), \text{ and } j < i
\end{array}
$$

Fig. 2. Specific Axioms for $\lambda\mathcal{S}_n$ $(1 \le i, j < n)$

$$
\begin{array}{rll}
\mathcal{C}_i k.\langle M\rangle_i = \mathcal{C}_i k.M & & \mathcal{C}\text{-reset} \\
\mathcal{C}_i k.k\langle M\rangle_{i-1} = \langle M\rangle_{i-1} & & \mathcal{C}\text{-elim, if } k \notin FV(M) \\
\langle E^j[\mathcal{C}_i k.M]\rangle_i = \langle M\{k := \lambda f.\ \mathcal{A}_i\langle E^j[f]\rangle_{i-1}\}\rangle_i & & \mathcal{C}\text{-lift} \\
 & & \text{if } k \notin FV(E^j),\ f \notin FV(kE^j), \text{ and } j < i
\end{array}
$$

Fig. 3. Specific Axioms for $\lambda\mathcal{C}_n$ $(1 \le i, j < n)$

since $\mathcal{S}_2 k.kN = N$ does not hold. However, by restricting N to $\langle M\rangle_i$, we have obtained a sound axiom.

The last axiom $\mathcal{S}$-lift is a natural extension of its level-1 counterpart (called reset-$\mathcal{S}$ formerly).[2] It is also a direct formulation of the operational semantics given in the earlier section, by changing reduction to equality.

We believe that the resulting axioms are simple to understand, and the soundness of these axioms is not surprising. The completeness of $\lambda\mathcal{C}_n$ may be surprising, since one may think it lacks many important equations which were included in our axiomatization of $\mathcal{C}_2$ [17], such as:

$$
\begin{array}{rll}
\langle\langle M\rangle_i\rangle_l = \langle M\rangle_{max(l,i)} & & \text{reset-reset} \\
\langle\mathcal{C}_i k.M\rangle_j = \mathcal{C}_i k.M & & \text{reset-}\mathcal{C}\text{, if } j < i \\
\langle\mathcal{C}_i k.M\rangle_i = \langle M\{k := \mathcal{A}_i\}\rangle_i & & \mathcal{C}\text{-top} \\
(\lambda x.\mathcal{C}_1 k.M)N = \mathcal{C}_1 k.(\lambda x.M)N & & \text{let-}\mathcal{C}_1\text{, if } k \neq x \\
\mathcal{C}_1 M = \mathcal{C}_1 k.Mk & & \mathcal{C}_1\text{-fun, if } k \notin FV(M)
\end{array}
$$

[2] $\mathcal{S}$-lift does not immediately subsume reset-$\mathcal{S}$, since the latter allows an arbitrary M in $\mathcal{S}_1 M$, while the former restricts it to be a function. However, this difference does not matter since we can prove $\mathcal{S}_1 M = \mathcal{S}_1 k.Mk$ for $k \notin FV(M)$ in $\lambda\mathcal{S}_n$.

$$
\begin{array}{rll}
\langle V\rangle_1 = V & & \text{reset-value} \\
\langle(\lambda x.M)\langle N\rangle_1\rangle_1 = (\lambda x.\langle M\rangle_1)\langle N\rangle_1 & & \text{reset-lift} \\
\mathcal{S}_1 k.kM = M & & \mathcal{S}\text{-elim, if } k \notin FV(M) \\
\mathcal{S}_1 k.\langle M\rangle_1 = \mathcal{S}_1 k.M & & \mathcal{S}\text{-reset} \\
\langle E^0[\mathcal{S}_1 M]\rangle_1 = \langle M(\lambda x.\langle E^0[x]\rangle_1)\rangle_1 & & \text{reset-}\mathcal{S}\text{, if } x \notin FV(E^0)
\end{array}
$$

Fig. 4. Axioms for $\lambda_{\mathcal{S}}$ other than β_v, η_v, and β_Ω

Another seemingly missing axiom is an equation for lifting $\mathcal{C}_i$ over an evaluation context such as $E[\mathcal{C}_i k.M] = M\{k := \cdots\}$. In the next theorem we show that these equations are derivable.

From now on, we will mainly investigate $\lambda\mathcal{C}_n$. After proving its completeness, we will come back to $\lambda\mathcal{S}_n$.

Theorem 1. *The equations reset-reset, reset-$\mathcal{C}$, $\mathcal{C}$-top, let-$\mathcal{C}_1$ and $\mathcal{C}_1$-fun as well as the following equations are derivable in $\lambda\mathcal{C}_n$ where $k_1, \cdots, k_i, f$ are fresh variables, and k is not bound by C in $\mathcal{C}$-abort.*

$$
\begin{array}{rll}
E^j[\mathcal{A}_i\langle M\rangle_{i-1}] = \mathcal{A}_i\langle M\rangle_{i-1} & & \mathcal{A}\text{-abort, if } j < i \\
\mathcal{C}_i k.C[E^j[kV]] = \mathcal{C}_i k.C[kV] & & \mathcal{C}\text{-abort, if } j < i \\
E^j[\mathcal{C}_i k.M] = \mathcal{C}_1 k_1.\cdots\mathcal{C}_i k_i.M\{k := N\} & & \text{telescope, if } j < i \\
\quad \text{where } N \text{ is } \lambda f.k_i\langle k_{i-1}\cdots\langle k_1(E^j[f])\rangle_1\cdots\rangle_{i-1} & &
\end{array}
$$

Proof. The equation reset-reset is obtained in this way. By putting $M = x$ in reset-lift, we obtain $\langle\langle N\rangle_i\rangle_j = (\lambda x.\langle x\rangle_j)\langle N\rangle_i$, and by reset-value and β_Ω, we obtain $\langle\langle N\rangle_i\rangle_j = \langle N\rangle_i$ for $j \le i$. Similarly by putting $N = x$ in reset-lift-2, we obtain $\langle M\rangle_i = \langle\langle M\rangle_j\rangle_i$ for $j \le i$, hence we are done.

The equation reset-$\mathcal{C}$ is obtained by putting $E^j = [\ ]$ and $E^j = \langle[\ ]\rangle_j$ in $\mathcal{C}$-lift and comparing the results. The equation $\mathcal{C}$-top is obtained by putting $E^j = [\ ]$ in $\mathcal{C}$-lift and using reset-value and η_v.

For the remaining equations, we first derive by $\mathcal{C}$-elim and $\mathcal{C}$-reset:

$$M = \mathcal{C}_1 k_1.k_1 M = \mathcal{C}_1 k_1.\langle k_1 M\rangle_1 = \mathcal{C}_1 k_1.\mathcal{C}_2 k_2.\langle k_2\langle k_1 M\rangle_1\rangle_2$$

Iterating this process i-times we obtain:

$$M = \mathcal{C}_1 k_1.\mathcal{C}_2 k_2.\cdots\mathcal{C}_i k_i.\langle k_i\cdots\langle k_2\langle k_1 M\rangle_1\rangle_2\cdots\rangle_i$$

By this equation and $\mathcal{C}$-lift, we obtain the following key equation (where $j < i$):

$$
\begin{aligned}
E^j[\mathcal{C}_i k.M] &= \mathcal{C}_1 k_1.\cdots\mathcal{C}_i k_i.\langle k_i\langle\cdots\langle k_1(E^j[\mathcal{C}_i k.M])\rangle_1\cdots\rangle_{i-1}\rangle_i \\
&= \mathcal{C}_1 k_1.\cdots\mathcal{C}_i k_i.\langle M\{k := \lambda f.\mathcal{A}_i\langle k_i\langle\cdots\langle k_1(E^j[f])\rangle_1\cdots\rangle_{i-1}\rangle_{i-1}\}\rangle_i
\end{aligned}
$$

For $\mathcal{A}$-abort, we first note that, if $j < i$ then we can derive $E^j[(\lambda x.N)\langle M\rangle_{i-1}] = (\lambda x.E^j[N])\langle M\rangle_{i-1}$ by reset-lift and β_Ω. Also, if $k \notin FV(M)$, then the right-hand

side of the key equation does not contain E^j, hence $E^j[\mathcal{C}_i k.M] = \mathcal{C}_i k.M$. Then we can compute as follows (for $j < i$):

$$\begin{aligned}
E^j[\mathcal{A}_i\langle M\rangle_{i-1}] &= E^j[(\lambda x.\mathcal{C}_i(\lambda k.x))\langle M\rangle_{i-1}] && \text{by definition}\\
&= (\lambda x.E^j[\mathcal{C}_i(\lambda k.x)])\langle M\rangle_{i-1} && \text{by the above equation}\\
&= (\lambda x.\mathcal{C}_i(\lambda k.x))\langle M\rangle_{i-1} && \text{by the key equation}\\
&= \mathcal{A}_i\langle M\rangle_{i-1}
\end{aligned}$$

For $\mathcal{C}$-abort we compute as follows (where $j < i$):

$$\begin{aligned}
&\mathcal{C}_i k.C[E^j[kV]]\\
&= \mathcal{C}_1 k_1.\cdots\mathcal{C}_i k_i.\langle(C[E^j[kV]])\{k := \lambda f.\mathcal{A}_i\langle k_i\langle\cdots\langle k_1 f\rangle_1\cdots\rangle_{i-1}\rangle_{i-1}\}\rangle_i\\
&= \mathcal{C}_1 k_1.\cdots\mathcal{C}_i k_i.\langle C[E^j[\mathcal{A}_i\langle k_i\langle\cdots\langle k_1 V\rangle_1\cdots\rangle_{i-1}\rangle_{i-1}]]\rangle_i\\
&= \mathcal{C}_1 k_1.\cdots\mathcal{C}_i k_i.\langle C[\mathcal{A}_i\langle k_i\langle\cdots\langle k_1 V\rangle_1\cdots\rangle_{i-1}\rangle_{i-1}]\rangle_i \qquad \text{by } \mathcal{A}\text{-abort}
\end{aligned}$$

Since the final result does not contain E^j, we have $\mathcal{C}_i k.C[E_1^j[kV]] = \mathcal{C}_i k.C[E_2^j[kV]]$ for any level-j evaluation contexts E_1^j and E_2^j. The axiom $\mathcal{C}$-abort then follows.

Applying $\mathcal{C}$-abort to the key equation (by putting $E^j = \mathcal{A}_i\langle[\ ]\rangle_{i-1}$ in $\mathcal{C}$-abort), we obtain the telescope axiom.[3] Verification of let-$\mathcal{C}_1$ and $\mathcal{C}_1$-fun is left for the reader.

This finishes the proof of the theorem.

6 Completeness Proof

The main results of this article are that the theories $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$ are sound and complete with respect to the extended CPS translation into the theory $\lambda_{\beta\eta}$. In the previous work, we proved completeness by the following strategy; (1) we analyzed the syntax of the image of CPS translation, (2) defined an inverse CPS translation, i.e., a direct-style transformation, and (3) proved that the equality is preserved through this inverse translation. The most difficult part was to find a suitable inverse translation, and we found it by trial and error. In this article our source calculus is much more complex than those in the previous studies, and therefore a better strategy is called for.

The proof method we present here is based on an idea by Sabry, who applied it to the axiomatization of a calculus with level-1 shift and *lazy* reset [22]. In this section, we develop his method in the type-theoretic framework so as to make the proof more structured.

6.1 The Target Calculus and its Type Structure

We analyze the set of terms which contains the image of the extended CPS translation and is closed under $\beta\eta$-reductions. We call this language (under the $\beta\eta$-equality) the target language or the target calculus.

[3] This is a generalized version of Murthy's telescope axiom [21].

An important observation on the target calculus is that it is typed by the following type structure:

$$\begin{aligned}
\mathtt{Term}_i &= \mathtt{Cont}_{i+1} \to \mathtt{Term}_{i+1} \quad \text{for } 0 \le i < n \\
\mathtt{Term}_n &= \mathtt{Ans} \\
\mathtt{Cont}_i &= \mathtt{Value} \to \mathtt{Term}_i \quad \text{for } 1 \le i \le n \\
\mathtt{Value} &= \mathtt{Value} \to \mathtt{Term}_0
\end{aligned}$$

where `Ans` is an arbitrary, fixed type, called the answer type [7, 9].[4] The above definition of types makes sense if we have recursive types, that is, if the recursive equation for `Value` has a solution. Note that if we were working in the typed setting from the beginning, that is, our source calculi were simply typed lambda calculi, we would not need recursive types for `Value`.

Using the type structure, terms of the target calculus can be introduced as typed terms. A typing judgment is of the form $\Gamma \vdash P : T$ where Γ is a set of variable-type pairs consisting of either $x : \mathtt{Value}$ or $k_i : \mathtt{Cont}_i$. As usual, a variable may occur at most once in Γ. We have the following eight type inference rules for $0 < i \le n$:

$$\frac{\Gamma \vdash W : \mathtt{Value} \quad \Gamma \vdash W' : \mathtt{Value}}{\Gamma \vdash WW' : \mathtt{Term}_0} \qquad \frac{\Gamma, k_i : \mathtt{Cont}_i \vdash T_i : \mathtt{Term}_i}{\Gamma \vdash \lambda k_i.T_i : \mathtt{Term}_{i-1}}$$

$$\frac{\Gamma \vdash T_{i-1} : \mathtt{Term}_{i-1} \quad \Gamma \vdash K_i : \mathtt{Cont}_i}{\Gamma \vdash T_{i-1}K_i : \mathtt{Term}_i} \qquad \frac{\Gamma \vdash K_i : \mathtt{Cont}_i \quad \Gamma \vdash W : \mathtt{Value}}{\Gamma \vdash K_iW : \mathtt{Term}_i}$$

$$\frac{}{\Gamma, k_i : \mathtt{Cont}_i \vdash k_i : \mathtt{Cont}_i} \qquad \frac{\Gamma, x : \mathtt{Value} \vdash T_i \ : \ \mathtt{Term}_i}{\Gamma \vdash \lambda x.T_i \ : \ \mathtt{Cont}_i}$$

$$\frac{}{\Gamma, x : \mathtt{Value} \vdash x : \mathtt{Value}} \qquad \frac{\Gamma, x : \mathtt{Value} \vdash T_0 \ : \ \mathtt{Term}_0}{\Gamma \vdash \lambda x.T_0 \ : \ \mathtt{Value}}$$

where $\Gamma, x : \mathtt{Value}$ means the set union $\Gamma \cup \{x : \mathtt{Value}\}$.

If we can prove $\Gamma \vdash P : T$ using the typing rules above, we say that P is a term (of type T) in the target calculus. For instance, the term $\lambda k_1.\ k_1 x$ (which is the $\beta\eta$-reduced term of $[\![\mathcal{C}_2(\lambda f.fx)]\!]$) can be typed as follows:

$$\dfrac{\dfrac{\dfrac{}{x : \mathtt{Value}, k_1 : \mathtt{Cont}_1 \vdash k_1 : \mathtt{Cont}_1} \quad \dfrac{}{x : \mathtt{Value}, k_1 : \mathtt{Cont}_1 \vdash x : \mathtt{Value}}}{x : \mathtt{Value}, k_1 : \mathtt{Cont}_1 \vdash k_1 x : \mathtt{Term}_1}}{x : \mathtt{Value} \vdash \lambda k_1.\ k_1 x : \mathtt{Term}_0}$$

For this type structure, it is not difficult to prove the following theorem.

Theorem 2. *(1) Let M and V be a term and a value, resp. in $\lambda\mathcal{C}_n$. Then we can derive $\Gamma \vdash [\![M]\!] : \mathtt{Term}_0$ and $\Gamma' \vdash V^* : \mathtt{Value}$ for some Γ and Γ'.*

(2) If we can derive $\Gamma \vdash P : T$ in the target calculus, and P reduces to Q by $\beta\eta$-reductions, then we can derive $\Gamma \vdash Q : T$.

[4] We can make the answer type parametric, as investigated by Thielecke [25].

6.2 Direct-Style Translation

We define an extended direct-style translation from the target calculus to the source calculus $\lambda\mathcal{C}_n$. We first give it as a syntactic translation $\_^\dagger$ based on the type structure of target terms as follows (for $0 < i \leq n$):

$$\begin{array}{ll} (WW')^\dagger \stackrel{def}{=} W^\dagger W'^\dagger & (\lambda k_i.T_i)^\dagger \stackrel{def}{=} \mathcal{C}_i k_i.{T_i}^\dagger \\ (T_{i-1}K_i)^\dagger \stackrel{def}{=} {K_i}^\dagger\langle {T_{i-1}}^\dagger\rangle_{i-1} & (K_iW)^\dagger \stackrel{def}{=} {K_i}^\dagger W^\dagger \\ {k_i}^\dagger \stackrel{def}{=} k_i & (\lambda x.T_i)^\dagger \stackrel{def}{=} \lambda x.\langle {T_i}^\dagger\rangle_i \\ x^\dagger \stackrel{def}{=} x & (\lambda x.T_0)^\dagger \stackrel{def}{=} \lambda x.{T_0}^\dagger \end{array}$$

The next theorem ensures that $\_^\dagger$ is in fact a translation from $\lambda_{\beta\eta}$ to $\lambda\mathcal{C}_n$.

Theorem 3. *The translation $\_^\dagger$ respects the $\beta\eta$-equality in the target calculus, namely, if P and Q are typable terms in the target calculus and $\lambda_{\beta\eta} \vdash P = Q$, then $\lambda\mathcal{C}_n \vdash P^\dagger = Q^\dagger$.*

We also have that $\_^\dagger$ is really an inverse of the extended CPS translation.

Theorem 4. *If M is a term in $\lambda\mathcal{C}_n$, then $\lambda\mathcal{C}_n \vdash [\![M]\!]^\dagger = M$.*

The proofs of these theorems are not shown here due to lack of space.

Now we can prove the completeness of $\lambda\mathcal{C}_n$.

Theorem 5 (Soundness & Completeness). *Let M and N be terms in $\lambda\mathcal{C}_n$. Then we have:*

$$\lambda\mathcal{C}_n \vdash M = N \text{ if and only if } \lambda_{\beta\eta} \vdash [\![M]\!] = [\![N]\!]$$

Proof. Soundness (the "only-if" direction) can be proved by calculating both sides of axioms in $\lambda\mathcal{C}_n$. For completeness (the "if" direction), suppose $\lambda_{\beta\eta} \vdash [\![M]\!] = [\![N]\!]$. Since $[\![M]\!]$ and $[\![N]\!]$ are of type T_0, we have $\lambda\mathcal{C}_n \vdash [\![M]\!]^\dagger = [\![N]\!]^\dagger$ by Theorem 3. Using Theorem 4, we conclude $\lambda\mathcal{C}_n \vdash M = N$.

6.3 Completeness of $\lambda\mathcal{S}_n$

We finally obtain the completeness of $\lambda\mathcal{S}_n$.

Theorem 6 (Soundness & Completeness). *Let M and N be terms in $\lambda\mathcal{S}_n$. Then we have:*

$$\lambda\mathcal{S}_n \vdash M = N \text{ if and only if } \lambda_{\beta\eta} \vdash [\![M]\!] = [\![N]\!]$$

Proof. Soundness can be proved in the same way as $\lambda\mathcal{C}_n$. For completeness, let ϕ be a translation from terms in $\lambda\mathcal{S}_n$ to terms in $\lambda\mathcal{C}_n$ which replaces $\mathcal{S}_i$ by its "definition" in $\mathcal{C}_i$ given in Section 3. Similarly let ψ be a translation from $\lambda\mathcal{C}_n$ to $\lambda\mathcal{S}_n$. It suffices to prove the following properties with M and M' being terms in $\lambda\mathcal{S}_n$, and N and N' being terms in $\lambda\mathcal{C}_n$:

1. $\lambda_{\beta\eta} \vdash [\![\phi(M)]\!] = [\![M]\!]$.
2. $\lambda\mathcal{C}_n \vdash N = N'$ implies $\lambda\mathcal{S}_n \vdash \psi(N) = \psi(N')$.
3. $\lambda\mathcal{S}_n \vdash \psi(\phi(M)) = M$

All these properties can be proved by calculation.

6.4 Typing Source Calculi

So far we have been studying the simplest possible source calculi. Introducing type structure into the source calculi is an important problem, as most modern programming languages have a built-in type system. Another benefit of introducing types is that, in the presence of appropriate types, we can avoid the full η_v-equality ($\lambda x.Vx = V$ for $x \notin FV(V)$), which is inconsistent with the presence of basic values such as natural numbers. In order to restrict V in η_v to a functional value, we need a type system in the source calculus.

A simple choice of the typing rules of control operators would be:

$$\frac{\Gamma \vdash M : \Phi}{\Gamma \vdash \langle M \rangle_i : \Phi} \qquad \frac{}{\Gamma \vdash \mathcal{S}_i : ((A \to \Phi) \to \Phi) \to A}$$

where Φ is a designated atomic type, and A is an arbitrary type. Then we can prove that the extended CPS translation preserves the typability if we add the type information to the classes $\mathtt{Term}_0$ and `Value` in the target calculus (in which case, the type structure of the target calculus does not need recursive types). All the axioms and the proof of soundness and completeness in this article go through for the simply typed case.

Introduction of types to the source calculi makes explicit the connection of the extended CPS translation and the double negation translation. If we take $n = 1$, then by the definition of types given in Section 6.1 we have $\mathtt{Term}_0 = (\mathtt{Value} \to \mathtt{Ans}) \to \mathtt{Ans}$. If we take $n = 2$, then

$$\mathtt{Term}_0 = (\mathtt{Value} \to (\mathtt{Value} \to \mathtt{Ans}) \to \mathtt{Ans}) \to (\mathtt{Value} \to \mathtt{Ans}) \to \mathtt{Ans}$$

Hence the type `Ans` in the $n = 1$ case (which corresponds to $\bot$ in the double negation translation) is CPS translated to $(\mathtt{Value} \to \mathtt{Ans}) \to \mathtt{Ans}$ in the $n = 2$ case. Thus, we can say the extended translation represents an iterated double-negation translation.

In the literature, Danvy and Filinski [6] and Murthy [21] have proposed more liberal type systems for `shift` and `reset`. Since these type systems are quite complicated, it is not obvious whether our axioms work for them.

7 Conclusion

In this article we have studied a family of control operators in the CPS hierarchy. In particular, we have analyzed the image of the extended CPS translation with type-theoretic machinery, and have obtained a simple set of axioms which is sound and complete for all such control operators. To our knowledge this work is the first such result about the hierarchy of delimited continuation operators. Our axioms for level-n `shift`/`reset` are a simple extension of those for level-1 `shift`/`reset`, and the axioms for level-n $\mathcal{C}$/`reset` are even simpler than those for level-2 $\mathcal{C}$/`reset`.

The control operators in the CPS hierarchy have also been investigated by Murthy [21], who gave an elaborate type system for level-n `shift` and `reset`, and also gave a set of axioms for them. The difference between his work and ours is

that he only proved the soundness of the axioms and did not state completeness, and also that his set of axioms consists of many complex axioms such as the telescope axiom, while ours consists of a small number of simple axioms.

In another line of work, Danvy and Yang [9], Murthy [21], and Biernacka, Biernacki and Danvy [3] studied an operational aspect of the control operators in the CPS hierarchy by giving abstract machines for `shift` and `reset`. It seems interesting to study how our axioms fit with these abstract machines.

Future Work: Besides studying the connection to abstract machines, there are two major avenues for future work:

(1) While we have built a theoretical foundation for the control operators in the CPS hierarchy there remains a question about the application of our axioms. It is almost impossible to use them for automatic verification because they require a degree of insight. Nevertheless, besides obtaining a better understanding of control operators, we hope to use the axioms to prove the correctness of program translations such as compiler optimization.

(2) An even more fundamental question of this study is whether one needs these hierarchical control operators at all. The existence of several application programs and the correspondence between the CPS hierarchy and layered monads [11] seem to give a positive answer to this question. However, there is much room for further work.

Acknowledgments: The author would like to thank Olivier Danvy, Masahito Hasegawa, Amr Sabry and Kenichi Asai for constructive comments and discussions. He also thanks anonymous referees for many insightful comments and criticisms. This work was supported in part by Grant-in-Aid for Scientific Research No. 16500004 from Japan Society for the Promotion of Science.

References

1. K. Asai. Online Partial Evaluation for Shift and Reset. In *Proc. ACM Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, pages 19–30, 2002.
2. K. Asai. Offline Partial Evaluation for Shift and Reset. In *Proc. ACM Workshop on Partial Evaluation and Semantics-Based Program Manipulation*, to appear.
3. M. Biernacka, D. Biernacki, and O. Danvy. An operational foundation for delimited continuations. In *Proc. Fourth ACM SIGPLAN Workshop on Continuations, Technical Report CSR-04-1, School of Computer Science, University of Birmingham*, pages 25–34, 2004.
4. O. Danvy. Type-directed partial evaluation. In *Proc. 23rd Symposium on Principles of Programming Languages*, pages 242–257, 1996.
5. O. Danvy. On evaluation contexts, continuations, and the rest of the computation. In *Proc. Fourth ACM SIGPLAN Workshop on Continuations, Technical Report CSR-04-1, School of Computer Science, University of Birmingham*, pages 13–23, 2004.
6. O. Danvy and A. Filinski. A functional abstraction of typed contexts. Technical Report 89/12, DIKU, University of Copenhagen, 1989.
7. O. Danvy and A. Filinski. Abstracting Control. In *Proc. 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, 1990.

8. O. Danvy and A. Filinski. Representing Control: a Study of the CPS Transformation. *Mathematical Structures in Computer Science*, 2(4):361–391, 1992.
9. O. Danvy and Z. Yang. An Operational Investigation of the CPS Hierarchy. In *Proc. 8th European Symposium on Programming*, Lecture Notes in Computer Science 1576, pages 224–242, 1999.
10. M. Felleisen. The Theory and Practice of First-Class Prompts. In *Proc. 15th Symposium on Principles of Programming Languages*, pages 180–190, 1988.
11. A. Filinski. Representing Layered Monads. In *Proc. 26th Symposium on Principles of Programming Languages*, pages 175–188, 1999.
12. H. Friedman. Classically and intuitionistically provably recursive functions. Lecture Notes in Mathematics 699, pages 21–28, 1978.
13. T. Griffin. A Formulae-as-Types Notion of Control. In *Proc. 17th Symposium on Principles of Programming Languages*, pages 47–58, 1990.
14. C. A. Gunter, D. Rémy, and J. G. Riecke. A Generalization of Exceptions and Control in ML-Like Languages. In *Proc. Functional Programming and Computer Architecture*, pages 12–23, 1995.
15. R. Hieb, R. Dybvig, and C. W. Anderson. Subcontinuations. *Lisp and Symbolic Computation*, 6:453–484, 1993.
16. Y. Kameyama. A Type-Theoretic Study on Partial Continuations. In *Proc. IFIP International Conference on Theoretical Computer Science*, Lecture Notes in Computer Science 1872, pages 489–504, 2000.
17. Y. Kameyama. Axiomatizing higher level delimited continuations. In *Proc. 3rd ACM SIGPLAN Workshop on Continuations, Technical Report 545, Computer Science Department, Indiana University*, pages 49–53, 2004.
18. Y. Kameyama and M. Hasegawa. A Sound and Complete Axiomatization of Delimited Continuations. In *Proc. ACM International Conference on Functional Programming*, pages 177–188, 2003.
19. J. Lawall and O. Danvy. Continuation-based partial evaluation. In *Proc. 1994 ACM Conference on LISP and Functional Programming*, pages 227–238, 1994.
20. E. Moggi. Computational Lambda-Calculus and Monads. In *Proc. 4th Symposium on Logic in Computer Science*, pages 14–28, 1989.
21. C. Murthy. Control operators, hierarchies, and pseudo-classical type systems: A-translation at work. In *Proc. First ACM Workshop on Continuations, Technical Report STAN-CS-92-1426, Stanford University*, pages 49–72, 1992.
22. A. Sabry. Note on Axiomatizing the Semantics of Control Operators. Technical Report CIS-TR-96-03, Dept. of Computer Science, University of Oregon, 1996.
23. A. Sabry and M. Felleisen. Reasoning about Programs in Continuation-Passing Style. *Lisp and Symbolic Computation*, 6(3-4):289–360, 1993.
24. T. Sekiguchi, T. Sakamoto, and A. Yonezawa. Portable Implementation of Continuation Operators in Imperative Languages by Exception Handling. In *Advances in Exception Handling Techniques*, Lecture Notes in Computer Science 2022, pages 217–233, 2001.
25. H. Thielecke. Answer type polymorphism in call-by-name continuation passing. In *Proc. 13th European Symposium on Programming*, Lecture Notes in Computer Science 2986, pages 279–293, 2004.