Axioms for Control Operators in the CPS Hierarchy *

Yukiyoshi Kameyama kameyama@acm.org
Department of Computer Science, University of Tsukuba

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 translation, control operator, delimited continuation, 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 [17]. The control mechanisms added to the source language can also be understood logically. For instance, Griffin [18] has revealed the Curry-Howard correspondence between the calculus with callcc and classical logic.

Danvy and Filinski [9, 10] have observed that there is room for a more refined control mechanism. By repeatedly CPS translating the answer type of the standard CPS translation, they obtained what they call a CPS hierarchy. Furthermore, they have proposed a family of

* This article is a revised version of the paper presented at CSL'04, September, 2004.

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 [12, 19, 20, 21]. In contrast to these control operators, shift and reset are defined by the CPS translation in a purely functional way. In addition, they have found applications in partial evaluation [24], one-pass CPS translation [10], and normalization by evaluation [7], as well as to represent layered monads [15] and mobile computation [30].

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 about the image of a translation, since a CPS translation is a global program transformation which significantly changes 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 two equations for reset, and then prove that this set of equations is sound and complete with respect to the CPS translation. This work builds on our previous work, in which we gave a sound and complete axiomatization for level-1 shift and reset [23], and for level-2 shift and reset [22]. Since completeness proofs of this kind often require quite a lot of calculations, we make the proof more structured by reconstructing Sabry's proof method [28] in a type-theoretic setting.

Overview: The rest of this article is organized as follows. In Section 2, we informally introduce shift and reset and we explain their operational aspects. In Section 3 we formally introduce the calculi with these control operators, and in Section 4, we give a CPS translation for them as well as a type structure for the target calculus of the CPS translation. In Section 5 we present the axioms for control operators and develop useful equations from them. In Section 6, we give a direct-style translation as an inverse of the CPS translation, and by using it we prove soundness and 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

2.1. A Simple Example

The following example uses shift and reset in a simple way:

$$\begin{aligned} 1 \ + \ \langle 2 \ * \ \mathcal{S}k.\ (3 \ + \ (k\ (k\ 4)))\rangle \ &= \ \texttt{let}\ k\ \texttt{be}\ \lambda x.\langle 2 * x\rangle \\ &\quad\ \texttt{in}\ \ 1 \ + \ \langle 3 \ + \ (k\ (k\ 4))\rangle \\ &= \ 1 \ + \ \langle 3 \ + \ \langle 2 \ * \ \langle 2 \ * \ 4\rangle\rangle\rangle \\ &= \ 20 \end{aligned}$$

where $\langle \_ \rangle$ is the reset operator and $\mathcal{S}$ is the shift operator, which binds the variable k in the scope $3 \ + \ (k\ (k\ 4))$. Unlike the continuation captured by callcc, the continuation captured by shift is not the whole rest of the computation (such as $1 + \langle 2 * [\ ]\rangle$), but the part which is delimited by reset, that is, $\langle 2 * [\ ]\rangle$. 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) nearest one is chosen as the delimiter.

As more substantial examples, we borrow the ones by Danvy and Filinski [9].

2.2. Nondeterminism

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

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

where $\_$ is an anonymous variable, `true` and `false` are truth values, and `begin`$\cdots$`end` is for sequencing.

To understand the program, we CPS translate[1] these three functions as:

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

Suppose `display` is a function for displaying values to the screen, and consider the program $\langle\texttt{display}(\texttt{choice}(3))\rangle$. Its CPS translation (as a program) is $\texttt{choice-c}(3,\texttt{display})$, which displays 1, 2 and 3. Thus, shift captures the current continuation, which is composable with an ordinary function. The control operator reset delimits the extent of the captured evaluation context. Its role can be made clearer by comparing the following two programs:

$$\begin{aligned}
p_1 &\stackrel{def}{=} \texttt{begin}\ \langle\texttt{display}(\texttt{choice}(3))\rangle;\ \texttt{display}(10)\ \texttt{end}\\
p_2 &\stackrel{def}{=} \langle\texttt{begin}\ \texttt{display}(\texttt{choice}(3));\ \texttt{display}(10)\ \texttt{end}\rangle
\end{aligned}$$

The program p_1 is CPS translated to

$$\texttt{begin}\ \texttt{choice-c}(3,\texttt{display});\ \texttt{display}(10)\ \texttt{end}$$

which, when executed, displays 1, 2, 3 and then 10. On the other hand, the program p_2 is CPS translated to

$$\texttt{choice-c}(3,\lambda x.(\texttt{begin}\ \texttt{display}(x);\ \texttt{display}(10)\ \texttt{end}))$$

which, when executed, displays 1, 10, 2, 10, 3, and 10.

2.3. Collecting Successive Results

Consider now the function `emit` defined as follows, which is meant to collect a series of answers:

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

where `cons` and `nil` are for lists. For instance,

$$\langle\texttt{begin}\ \texttt{emit}(1);\ \texttt{emit}(2);\ \texttt{emit}(3);\ \texttt{nil}\ \texttt{end}\rangle$$

returns a list (1 2 3).

[1] The definition of the CPS translation appears in a later section.

It is then natural to expect that the combined program $\langle \texttt{emit}(\texttt{choice}(3))\rangle$ generates the list (1 2 3) of all non-deterministic choices. 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 $\langle \texttt{emit}(\texttt{choice}(3))\rangle$ is CPS translated into

$$\texttt{choice-c}(3, \lambda x.\texttt{emit-c}(x, \lambda x.x)),$$

which generates three lists (1), (2) and (3), but never collects 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:

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

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

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

which passes a continuation, even though it is not in CPS since the argument of k is not a trivial term.[2] Its direct-style counterpart is:

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

This is the point where we need a level-2 control operator $\mathcal{S}_2$ in the CPS hierarchy.

Finally, by replacing control operators without indices by those with level-1 (for instance $\mathcal{S}$ is replaced by $\mathcal{S}_1$), and enclosing the whole term by a level-2 reset, we get a program:

$$\langle \texttt{begin}\ \langle \texttt{emit}(\texttt{choice}(3))\rangle_1;\ \texttt{nil}\ \texttt{end}\rangle_2$$

which evaluates to (1 2 3) as expected.

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

[2] A trivial term is a synonym for a value.

where all calls are tail calls and all subterms are trivial. The family of layered control operators thus corresponds to the CPS hierarchy.

3. The Calculi with Control Operators

In this section, we introduce the languages of our source calculi and their operational semantics.

3.1. Language of the Calculi

The calculus we choose here is a type-free call-by-value lambda calculus with control operators for delimited continuations. Later we will briefly discuss typed calculi.

We define two calculi $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$ for a natural number n. The former is a calculus with shift and reset, and the latter is a calculus with $\mathcal{C}$ and reset. The control operator $\mathcal{C}$ will be explained later.

We assume that there are infinitely many variables (written x, y, z, k and so on). Terms of $\lambda\mathcal{S}_n$ are defined by the following grammar ($1 \leq i \leq n$):

$$M, N \ ::= \ x \mid \lambda x.M \mid MN \mid \langle M \rangle_i \mid \mathcal{S}_i k.M$$

The index i denotes the level of control operators, which is conceptually the number of the iterations of CPS translations that are necessary to interpret the control operator. $\langle M \rangle_i$ is a term with a level-i reset operator, and $\mathcal{S}_i k.M$ is a term with a level-i shift operator. The variable k is bound in $\mathcal{S}_i k.M$. We define $\langle M \rangle_0 \stackrel{def}{=} M$.

Terms of $\lambda\mathcal{C}_n$ are defined in the same way as $\lambda\mathcal{S}_n$ where $\mathcal{S}_i k.M$ is replaced by $\mathcal{C}_i k.M$. The variable k is bound in $\mathcal{C}_i k.M$.

A value (written V) is either a variable or λ-abstraction. Variables are bound by λ and $\mathcal{S}_i$ in $\lambda\mathcal{S}_n$ and λ and $\mathcal{C}_i$ in $\lambda\mathcal{C}_n$. Free and bound variables of terms are defined as usual. $\mathrm{FV}(M)$ 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 capture-avoiding substitution of N for x in M.

Contexts (C), evaluation contexts (E) and level-j evaluation contexts (E^j for $0 \leq j \leq n$) in $\lambda\mathcal{S}_n$ are defined as follows:

$$\begin{aligned} C \ &::= \ [\,] \mid CM \mid MC \mid \lambda x.C \mid \mathcal{S}_i k.C \mid \langle C \rangle_i \\ E \ &::= \ [\,] \mid EM \mid VE \mid \langle E \rangle_i \\ E^j \ &::= \ [\,] \mid E^j M \mid V E^j \mid \langle E^j \rangle_l \ \text{ for } \ 1 \leq l \leq j \end{aligned}$$

Contexts and evaluation contexts are the standard ones in call by value with the addition of shift and reset. E^j is a level-j evaluation context in

which the levels of reset operators enclosing the hole must be equal to or less than j. 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. In our previous work [23], E^0 was called a *pure evaluation context*. Contexts, evaluation contexts and level-j evaluation contexts in $\lambda\mathcal{C}_n$ are defined similarly.

As we will see in the next section, being aware of the levels of evaluation contexts is important in defining reduction rules for control operators, since level-j shift captures the delimited continuation up to level-j.

3.2. Operational Semantics

We introduce the operational semantics of $\lambda\mathcal{S}_n$ in this section. The formal definitions of shift and reset will be given in terms of the CPS translation, and the operational semantics in this section should be considered as an informal explanation of these control operators. Nevertheless, we present it here to provide intuition about their operational behavior.

The small-step operational semantics is given a set of reduction rules for programs, where programs are terms that have no free variables[3]. We have the following three reduction rules (where $x \notin \mathrm{FV}(E^{j-1})$ and $1 \le j \le n$):

$$\begin{aligned} E[(\lambda x.M)V] &\rightarrow E[M\{x := V\}] \\ E[\langle V\rangle_j] &\rightarrow E[V] \\ E[\langle E^{j-1}[\mathcal{S}_j k.M]\rangle_j] &\rightarrow E[\langle M\{k := \lambda x.\langle E^{j-1}[x]\rangle_j\}\rangle_j] \end{aligned}$$

The first rule is the standard call-by-value β-reduction. The second rule says that delimiting a value does nothing. The third rule shows how shift and reset work. Shift captures the continuation delimited by the reset which has the same level as shift. If there are multiple occurrences of reset which enclose the term $\mathcal{S}_j k.M$, the (dynamically determined) nearest reset is chosen, as the evaluation context E^{j-1} contains no occurrences of level-j reset which enclose the hole.

In order to understand the third reduction rule, let us compare it with the rule for callcc, the control operator for *unlimited* continuations in Scheme and Standard ML of New Jersey:

$$E^0[\mathtt{callcc}(\lambda k.M)] \rightarrow E^0[M\{k := \lambda x.\mathtt{abort}(E^0[x])\}]$$

where `abort` has the reduction rule: $E^0[\mathtt{abort}(V)] \rightarrow V$. In the case of shift, the continuation delimited by the corresponding reset operator

[3] We will refine the definition of programs later in this section.

(the evaluation context $\langle E^{j-1}\rangle_j$) is captured and k is bound to a function $\lambda x.\langle E^{j-1}[x]\rangle_j$. In the case of callcc, the whole evaluation context E^0 is captured and k is bound to a function $\lambda x.\mathtt{abort}(E^0[x])$. Thus, continuations captured by shift are called *delimited* continuations. Another difference between these operators is that the former contains an additional reset, while the latter contains an additional `abort`. As a consequence continuations captured by shift are composable with other functions.

To illustrate how the three reduction rules are used, we evaluate the `choice-emit` example. The program we want to evaluate is $\langle E[M]\rangle_2$ where E is a level-1 evaluation context ($\mathtt{begin}\ \langle[\ ]\rangle_1;\ \mathtt{nil}\ \mathtt{end}$) and M is $\mathtt{emit}(\mathtt{choice}(1))$. $\rightarrow^*$ denotes the reflexive, transitive closure of $\rightarrow$.

$$
\begin{aligned}
& \langle E[M]\rangle_2 \\
\rightarrow^* \ & \langle E[\mathtt{emit}(\mathtt{if}\ (\mathcal{S}_1 c.\mathtt{begin}\ c(\mathtt{true}); c(\mathtt{false}); \mathtt{fail}(\_)\ \mathtt{end}) \\
& \qquad \mathtt{then}\ \mathtt{choice}(0)\ \mathtt{else}\ 1)]\rangle_2 \\
\rightarrow^* \ & \langle E[\mathtt{begin}\ (\lambda x.\langle \mathtt{emit}(\mathtt{if}\ x\ \mathtt{then}\ \mathtt{choice}(0)\ \mathtt{else}\ 1)\rangle_1)(\mathtt{true}); \\
& \qquad\quad (\lambda x.\langle \mathtt{emit}(\mathtt{if}\ x\ \mathtt{then}\ \mathtt{choice}(0)\ \mathtt{else}\ 1)\rangle_1)(\mathtt{false}); \\
& \qquad\quad \mathtt{fail}(\_) \\
& \qquad \mathtt{end}]\rangle_2 \\
\rightarrow^* \ & \langle E[\mathtt{begin}\ \langle \mathcal{S}_2 c.\mathtt{cons}(1, c(\mathtt{nil}))\rangle_1; \mathtt{fail}(\_)\ \mathtt{end}]\rangle_2
\end{aligned}
$$

Now the continuation captured by $\mathcal{S}_2$ is $\langle E[\mathtt{begin}\ \langle[\ ]\rangle_1;\ \mathtt{fail}(\_)\ \mathtt{end}]\rangle_2$. If we had used $\mathcal{S}_1$ instead of $\mathcal{S}_2$ in the definition of `emit`, the captured continuation would have been $\langle[\ ]\rangle_1$, in which case the function `emit` could not collect all the answers of non-deterministic choices.

The whole evaluation proceeds as follows:

$$
\begin{aligned}
\rightarrow^* \ & \langle \mathtt{cons}(1, c(\mathtt{nil}))\{c := \lambda x.\langle E[\mathtt{begin}\ \langle x\rangle_1;\ \mathtt{fail}(\_)\ \mathtt{end}]\rangle_2\}\rangle_2 \\
\rightarrow^* \ & \langle \mathtt{cons}(1, \langle E[\mathtt{begin}\ \langle \mathtt{nil}\rangle_1;\ \mathtt{fail}(\_)\ \mathtt{end}]\rangle_2)\rangle_2 \\
\rightarrow^* \ & \mathtt{cons}(1, \mathtt{nil})
\end{aligned}
$$

Similarly we have:

$$\langle \mathtt{begin}\ \langle \mathtt{emit}(\mathtt{choice}(3))\rangle_1;\ \mathtt{nil}\ \mathtt{end}\rangle_2 \rightarrow^* \mathtt{cons}(1, \mathtt{cons}(2, \mathtt{cons}(3, \mathtt{nil})))$$

which is the desired result.

The reduction rule for shift (the third reduction rule) is applicable only when the level of shift and that of reset agree, and we must generalize it to the case when they are different. Shift and reset are layered control operators in that the evaluation context captured by a level-j shift is delimited not only by a level-j reset but by any higher-level reset. Therefore a general reduction rule is (where $j \leq i$ and

$f \notin \mathrm{FV}(E^{j-1})$):

$$E[\langle E^{j-1}[\mathcal{S}_j k.M]\rangle_i] \;\rightarrow\; E[\langle M\{k := \lambda x.\langle E^{j-1}[x]\rangle_j\}\rangle_i]$$

Put differently, a level-i reset delimits not only a level-i evaluation context, but also any level-j evaluation contexts if $j \leq i$.

3.3. Comparison of Control Operators for Delimited Continuations

We briefly compare several control operators for delimited continuations proposed in the literature.

The first group of control operators consists of shift/reset and their relatives, $\mathcal{C}$/reset and $\mathcal{D}$/reset, whose key reduction rules are defined as follows (for $j \leq i$):

$$E[\langle E^{j-1}[\mathcal{C}_j k.M]\rangle_i] \;\rightarrow\; E[\langle M\{k := \lambda x.\mathcal{A}_j\langle E^{j-1}[x]\rangle_j\}\rangle_i]$$
$$E[\langle E^{j-1}[\mathcal{D}_j k.M]\rangle_i] \;\rightarrow\; E[\langle E^{j-1}[M\{k := \lambda x.\mathcal{A}_j\langle E^{j-1}[x]\rangle_j\}]\rangle_i]$$

The control operator $\mathcal{C}$ was investigated by Murthy [26], and we use it in the completeness proof in a later section. The control operator $\mathcal{D}$ generalizes callcc, for $\mathcal{D}_1$ coincides with callcc in the absence of reset [23]. Filinski studied the relationship among these control operators [14].

Despite the difference of reduction rules, these control operators have the same expressive power, for they are inter-definable (in the presence of $\mathcal{A}$). More precisely we have:

$$(\mathcal{S}_i, \mathrm{reset}) \leftrightarrow (\mathcal{C}_i, \mathrm{reset}) \leftrightarrow (\mathcal{D}_i, \mathcal{A}_i, \mathrm{reset})$$

where $\leftrightarrow$ means inter-definability. For instance, $\mathcal{S}_i$ and $\mathcal{C}_i$ are defined by each other as:

$$\mathcal{S}_i k.M \;=\; \mathcal{C}_i k'.M\{k := \lambda x.\langle k' x\rangle_i\}$$
$$\mathcal{C}_i k.M \;=\; \mathcal{S}_i k'.M\{k := \lambda x.\mathcal{S}_i \_.k' x\}$$

where $\_$ is an anonymous variable. These equations can be formally justified by the CPS translation in the next section.

In the literature shift and reset are sometimes called control operators for "composable continuations", as the continuations they capture are composable, while those captured by callcc (and by $\mathcal{C}_i$ and $\mathcal{D}_i$) are abortive. We think that "delimited continuations" is a better name, since as long as we are concerned only with expressivity, it does not matter whether the captured continuations are composable or not, while it does matter whether the calculus in consideration has a delimiter (reset) or not.

The second group consists of Felleisen's prompt (denoted by #) and $\mathcal{F}$, the first control operator for delimited continuations [12], and their relatives (e.g. Gunter, Rémy and Riecke's operator "cupto" [19]). The reduction rule of Felleisen's operators is defined by:

$$E[\#(E^0[\mathcal{F}k.M])] \;\rightarrow\; E[\#(M\{k := \lambda x.E^0[x]\})]$$

The reduction rule looks similar to that for shift/reset if we put $i = j = 1$, and identify # with reset and $\mathcal{F}$ with shift of level-1. A notable difference is that the evaluation context captured by $\mathcal{F}$ does not contain the enclosing prompt (as indicated by $\lambda x.E^0[x]$ in the righthand side), and the captured continuation is therefore merged with an outer continuation, which leads to the dynamic behavior of these control operators.

Control operators in the first group are *static* and have simple CPS translations expressed as pure lambda terms, while those in the second group are *dynamic* and are not known to have such simple CPS translations. In this article, we use the CPS translations as an essential tool and develop a rigid theory by making use of it, hence we confine ourselves to the static control operators. We will come back to this issue in Section 7.

4. CPS Translation

In this section we follow Danvy and Filinski [9] and give CPS translations for higher level shift and reset. For simplicity, the version we present here does not use environments nor it does perform optimization for administrative redexes.

4.1. The First CPS Translation

The first CPS translation is the standard Plotkin-style call-by-value CPS translation extended to the calculi $\lambda\mathcal{S}_1$ and $\lambda\mathcal{C}_1$. It consists of two translations $[\![\_]\!]_1$ for terms and $\_^{*_1}$ for values in $\lambda\mathcal{S}_1$ (or $\lambda\mathcal{C}_1$). The results of these translations are type-free lambda terms without control

operators:

$$
\begin{aligned}
[\![V]\!]_1 &\stackrel{def}{=} \lambda k.\ k(V^{*_1}) \\
[\![MN]\!]_1 &\stackrel{def}{=} \lambda k.\ [\![M]\!]_1(\lambda m.\ [\![N]\!]_1(\lambda n.\ mnk)) \\
[\![\mathcal{S}_1 c.M]\!]_1 &\stackrel{def}{=} \lambda k.\ [\![M]\!]_1\{c := \lambda x k'.\ k'(kx)\}(\lambda x.x) \\
[\![\mathcal{C}_1 c.M]\!]_1 &\stackrel{def}{=} \lambda k.\ [\![M]\!]_1\{c := \lambda x k'.\ kx\}(\lambda x.x) \\
[\![\langle M \rangle_1]\!]_1 &\stackrel{def}{=} \lambda k.\ k([\![M]\!]_1(\lambda x.x)) \\
x^{*_1} &\stackrel{def}{=} x \\
(\lambda x.M)^{*_1} &\stackrel{def}{=} \lambda x.\ [\![M]\!]_1
\end{aligned}
$$

For shift (the third line), the current continuation k is captured and the variable c is bound to a function which composes k and k'. For $\mathcal{C}$ (the fourth line), the function bound to c discards k', and it is abortive. For reset (the fifth line), the current continuation is reset to the identity function $\lambda x.x$ (which corresponds to the empty evaluation context $[\ ]$). The resulting terms are not in the usual CPS form in that some subterms are not trivial (kx and $[\![M]\!]_1(\lambda x.x)$ are in the argument positions of functional application). Shift and reset represent a new programming pattern where one can compose continuations with functions and reset continuations, which Danvy and Filinski called *Continuation Composing Style* (CCS for short).

We introduce the following type structure for the target terms of the first CPS translation:

$$
\begin{aligned}
\mathtt{Exp}_0 &\stackrel{def}{=} \mathtt{Cont}_1 \to \mathtt{Ans} \\
\mathtt{Cont}_1 &\stackrel{def}{=} \mathtt{Value} \to \mathtt{Ans} \\
\mathtt{Value} &\stackrel{def}{=} \mathtt{Value} \to \mathtt{Exp}_0
\end{aligned}
$$

where `Ans` is the type of answers. The type `Value` is defined by a recursive type, reflecting that the source calculi are type-free.

The type structure above works for the source calculi without control operators. In the presence of control operators in the source calculi, we need one more constraint $\mathtt{Ans} = \mathtt{Value}$ which is necessary to type terms in CCS, for instance, $k'(kx)$. The target terms of the first CPS translation are typable under this constraint, namely, for a term M and a value V in $\lambda\mathcal{S}_1$ (or $\lambda\mathcal{C}_1$), $[\![M]\!]_1$ and V^{*_1} have the types $\mathtt{Exp}_0$ and `Value`, resp.

4.2. The Second CPS Translation

The results of the first CPS translation are not in the ordinary CPS form, but if we CPS translate them once more, the resulting terms are ordinary CPS terms, and the call-by-value and call-by-name semantics coincide (Plotkin's indifference theorem [27]). Rather than sending a source term M to $[\![[\![M]\!]_1]\!]_1$, Danvy and Filinski defined their second CPS translation as translating terms of the answer type only. A rationale for this choice is that, after the first CPS translation, a term of type $\mathtt{Ans}$ is not necessarily a trivial term, while a target term of other types is a trivial term.

We first introduce the type structure for the target terms of the second CPS translation:

$$\begin{array}{rcl@{\qquad}rcl} \mathtt{Exp}_0 & \stackrel{def}{=} & \mathtt{Cont}_1 \to \mathtt{Exp}_1 & \mathtt{Exp}_1 & \stackrel{def}{=} & \mathtt{Cont}_2 \to \mathtt{Ans} \\ \mathtt{Cont}_1 & \stackrel{def}{=} & \mathtt{Value} \to \mathtt{Exp}_1 & \mathtt{Cont}_2 & \stackrel{def}{=} & \mathtt{Value} \to \mathtt{Ans} \\ \mathtt{Value} & \stackrel{def}{=} & \mathtt{Value} \to \mathtt{Exp} & & & \end{array}$$

where $\mathtt{Ans}$ is an arbitrary type.

The second CPS translation $[\![\_]\!]_2^\sigma$ is parametrized over types σ. It sends a term of type $\mathtt{Ans}$ to a term of type $\mathtt{Exp}_1$ $(= (\mathtt{Value} \to \mathtt{Ans}) \to \mathtt{Ans})$, and it is homomorphic on a term of other types. We remark that after the first CPS translation, a term of type $\mathtt{Ans}$ $(= \mathtt{Value})$ is either one of the forms: MK (for $M : \mathtt{Exp}_0$ and $K : \mathtt{Cont}_1$), KW (for $K : \mathtt{Cont}_1$ and $W : \mathtt{Value}$), x (for $x : \mathtt{Value}$), or $\lambda x.M$ (for $x : \mathtt{Value}$ and $M : \mathtt{Exp}_0$).

$$\begin{array}{rcl@{\qquad}rcl} [\![KW]\!]_2^{\mathtt{Ans}} & \stackrel{def}{=} & \lambda\gamma.[\![W]\!]_2^{\mathtt{Ans}}(\lambda v.[\![K]\!]_2^{\mathtt{Cont}_1} v\gamma) & [\![\lambda k.A]\!]_2^{\mathtt{Exp}_0} & \stackrel{def}{=} & \lambda k.[\![A]\!]_2^{\mathtt{Ans}} \\ [\![MK]\!]_2^{\mathtt{Ans}} & \stackrel{def}{=} & \lambda\gamma.[\![M]\!]_2^{\mathtt{Exp}_0}[\![K]\!]_2^{\mathtt{Cont}_1}\gamma & [\![W_1W_2]\!]_2^{\mathtt{Exp}_0} & \stackrel{def}{=} & [\![W_1]\!]_2^{\mathtt{Ans}}[\![W_2]\!]_2^{\mathtt{Ans}} \\ [\![x]\!]_2^{\mathtt{Ans}} & \stackrel{def}{=} & \lambda\gamma.\gamma x & [\![k]\!]_2^{\mathtt{Cont}_1} & \stackrel{def}{=} & k \\ [\![\lambda x.M]\!]_2^{\mathtt{Ans}} & \stackrel{def}{=} & \lambda\gamma.\gamma(\lambda x.[\![M]\!]_2^{\mathtt{Exp}_0}) & [\![\lambda x.A]\!]_2^{\mathtt{Cont}_1} & \stackrel{def}{=} & \lambda x.[\![A]\!]_2^{\mathtt{Ans}} \end{array}$$

In the first two definitions in the left column, we gave an optimized version by making use of the fact that terms of type $\mathtt{Exp}$ or $\mathtt{Cont}_1$ are trivial terms.

By composing two CPS translations, we obtain a single CPS translation, called 2-CPS translation. After performing the administrative $\beta\eta$-reductions, the 2-CPS translation can be represented as follows

where two continuation variables k and γ are introduced:

$$\begin{aligned}
[\![V]\!] &\stackrel{def}{=} \lambda k\gamma.\ kV^*\gamma \\
[\![MN]\!] &\stackrel{def}{=} \lambda k\gamma.\ [\![M]\!](\lambda m\gamma'.\ [\![N]\!](\lambda n\gamma''.\ mnk\gamma'')\gamma')\gamma \\
[\![\mathcal{S}_1 c.M]\!] &\stackrel{def}{=} \lambda k\gamma.\ [\![M]\!]\{c := \lambda xk'\gamma'.\ kx(\lambda y.k'y\gamma')\}(\lambda a\gamma''.\gamma'' a)\gamma \\
[\![\mathcal{C}_1 c.M]\!] &\stackrel{def}{=} \lambda k\gamma.\ [\![M]\!]\{c := \lambda xk'.\ kx\}(\lambda a\gamma''.\gamma'' a)\gamma \\
[\![\langle M\rangle_1]\!] &\stackrel{def}{=} \lambda k\gamma.\ [\![M]\!](\lambda a\gamma'.\gamma' a)(\lambda x.kx\gamma) \\
x^* &\stackrel{def}{=} x \\
(\lambda x.M)^* &\stackrel{def}{=} \lambda x.\ [\![M]\!]
\end{aligned}$$

It is easy to check that the results are typable under the type structure above, namely, for a source term M and a value V, $[\![M]\!]$ and V^* are of types $\mathtt{Exp}_0$ and `Value`, resp.

4.3. CPS Hierarchy

The 2-CPS translation sends every term with shift and reset to the CPS form. That all the translated terms are in CPS means that there is more room yet for continuation composing style in the target language, namely, resetting and composing continuations, which should correspond to another set of control operators in the source language. By this observation, Danvy and Filinski proposed to iterate CPS translations, and obtained a *hierarchy* of higher level shift and reset control operators. What is interesting here is that we only have to iterate the standard CPS translations to obtain a series of control operators, and nothing else is needed.

Let us summarize the relation between the levels of source terms (that is, the maximum levels of control operators in source terms) and the number of iterations of CPS translations. Let M be a term with level $\leq n$ control operators. If we n-times CPS translate M, we get a term without control operators, but the result is not necessarily in the ordinary CPS form (a term in CCS). If we $n+1$-times CPS translate M, we get a term without control operators which is an ordinary CPS term. To type CCS terms, we need an additional constraint $\mathtt{Ans} = \mathtt{Value}$, while to type CPS terms, we do not need this constraint and the answer type can be an arbitrary type. In the following, we only consider $(n+1)$-times (or more) iteration of CPS translations for level-n control operators so that all the target terms are in the ordinary CPS forms.

We abbreviate sequences as follows (for $1 \leq i \leq n+1$):

$$\begin{aligned}
\lambda\overline{k_{1,i}}.M &\stackrel{def}{=} \lambda k_1.\lambda k_2.\cdots\lambda k_i.M \\
M\overline{k_{1,i}} &\stackrel{def}{=} Mk_1k_2\cdots k_i \\
M\overline{\theta_{1,i}} &\stackrel{def}{=} M\theta_1\theta_2\cdots\theta_i \\
\overline{\mathcal{C}k_{1,i}}.M &\stackrel{def}{=} \mathcal{C}_1k_1.\mathcal{C}_2k_2.\cdots\mathcal{C}_ik_i.M
\end{aligned}$$

We define a CPS translation from the source calculi $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$ ($n \geq 0$) to pure lambda calculus. The translation is a composition of $n+1$ CPS translations, which takes $n+1$ continuation variables $k_1, \cdots, k_{n+1}$ each introduced by the i-th CPS translation. Given a term M and a value V in the source calculi, the translations $[\![\_]\!]$ and $\_^*$ send a term and a value, resp, to terms of pure lambda calculus, To avoid clutter, we present an η-reduced version here.

$$\begin{aligned}
[\![V]\!] &\stackrel{def}{=} \lambda k_1.\ k_1V^* \\
[\![MN]\!] &\stackrel{def}{=} \lambda k_1.\ [\![M]\!](\lambda m.[\![N]\!](\lambda n.mnk_1)) \\
[\![\langle M\rangle_i]\!] &\stackrel{def}{=} \lambda\overline{k_{1,i+1}}.\ [\![M]\!]\overline{\theta_{1,i}}(\lambda y.\ \theta_0y\overline{k_{1,i+1}}) \\
[\![\mathcal{S}_ic.M]\!] &\stackrel{def}{=} \lambda\overline{k_{1,i}}.\ [\![M]\!]\{c := \lambda y\overline{k'_{1,i+1}}.\ \theta_0y\overline{k_{1,i}}(\lambda z.\theta_0z\overline{k'_{1,i+1}})\}\overline{\theta_{1,i}} \\
[\![\mathcal{C}_ic.M]\!] &\stackrel{def}{=} \lambda\overline{k_{1,i}}.\ [\![M]\!]\{c := \lambda y\overline{k'_{1,i}}.\ \theta_0y\overline{k_{1,i}}\}\overline{\theta_{1,i}} \\
x^* &\stackrel{def}{=} x \\
(\lambda x.M)^* &\stackrel{def}{=} \lambda x.[\![M]\!]
\end{aligned}$$

where $1 \leq i \leq n$, the variables $k_1, \cdots, k_{i+1}, k'_1, \cdots, k'_{i+1}$ are fresh, and $\theta_j \stackrel{def}{=} \lambda xk_{j+1}.\ k_{j+1}x$ (for $0 \leq j \leq n$), The term θ_j is the image of the empty evaluation context $[\ ]$. Although all θ_j are equal as type-free terms, we attach indices to distinguish its level.

Let us 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 continuations of up to level i with the level $i+1$ continuation. The operator $\mathcal{C}_i$ captures the current continuation up to level i and composes them (as the term $\lambda y\overline{k'_{1,i}}.\ \theta_0y\overline{k_{1,i}}$), and the variable c is bound to it. At the same time, it installs i identity continuations. The CPS translation of shift ($\mathcal{S}_i$) is slightly more complex than $\mathcal{C}_i$, since it captures a non-abortive delimited continuation so that we should compose $k'_1, \cdots, k'_{i+1}$ with the captured continuation. Note that the result of the extended CPS translation does not depend on n if it is defined.

Since the target terms of the extended CPS translation are in CPS, they are evaluation-order independent, and we choose $\beta\eta$-equality as its semantics. We can prove that the operational semantics in Section 3.2 is sound with respect to the CPS translation, namely, $M \to^* N$ implies $[\![M]\!] = [\![N]\!]$.

As an example, let us CPS translate the function `emit`. To do this, we assume that our source language contains `nil` and `cons` as values, and we do not CPS translate primitive functions, that is, they are CPS translated[4] by:

$$\begin{aligned} \mathtt{nil}^* &\stackrel{def}{=} \mathtt{nil} \\ \mathtt{cons}^* &\stackrel{def}{=} \lambda x k.\ k(\lambda y k'.k'(\mathtt{cons}(x,y))) \end{aligned}$$

Then by the extended CPS translation, `emit` is CPS translated as follows[5]:

$$\begin{aligned} \mathtt{emit}^* &\equiv (\lambda n.\mathcal{S}_2 c.\mathtt{cons}(n, c(\mathtt{nil})))^* \\ &\to^* \lambda n k\gamma.\ ((\lambda k.c(\mathtt{nil})(\lambda x.k(\mathtt{cons}(n,x))))\theta_1\theta_2) \\ &\qquad\qquad \{c := \lambda x k'\gamma'\delta'.kx\gamma(\lambda y.k'y\gamma'\delta')\} \\ &\to^* \lambda n k\gamma\delta.\ k(\mathtt{nil})\gamma(\lambda y.\delta(\mathtt{cons}(n,y))) \end{aligned}$$

The term $\mathtt{emit}^*$ differs from `emit-c2`, since the former is the result of 3-CPS translation, while the latter is the result of 2-CPS translation. If we translate `emit-c2` (on the answer type only), we get $\mathtt{emit}^*$.

To see how the layered control operators work, let p_3 be a term $\langle \mathtt{begin}\ \langle \mathtt{emit}(1)\rangle_1;\ \langle \mathtt{emit}(2)\rangle_1\ \mathtt{end}\rangle_2$, and CPS translate it as follows:

$$\begin{aligned} [\![p_3]\!] &\to^* \lambda k\gamma\delta.\ [\![\mathtt{emit}(1)]\!]\theta_1(\lambda\_.[\![\mathtt{emit}(2)]\!]\theta_1\theta_2)(\lambda x.kx\gamma\delta) \\ &\to^* \lambda k.\ k(\mathtt{cons}(1,\mathtt{cons}(2,\mathtt{nil}))) \end{aligned}$$

The calculation above does not work if all the occurrences of reset in p_3 have the same level.

For the CPS translation of a complete program (rather than a term), we need to supply an identity continuation $\lambda a.a$ to its CPS translation as a term. Since we iterate CPS translations (n+1) times, we need to supply (n+1) identity continuations, and all but the last of these identity continuations are translated to θ_j by the successive CPS translations. For instance, the program p_3 is CPS translated to $[\![p_3]\!]\theta_1\theta_2(\lambda a.a)$. which $\beta\eta$-reduces to $\mathtt{cons}(1,\mathtt{cons}(2,\mathtt{nil}))$ as expected.

Remark on Complete Programs: In their original article [9], Danvy and Filinski had an implicit assumption that every complete program

[4] We assume that `cons` is curryed.

[5] We use k, γ, and δ for the continuation variables k_1, k_2, and k_3 for readability.

is implicitly enclosed by a reset of sufficiently high level. For instance, a program $\mathcal{S}_2 c.c0$ does not cause an error. Rather, it is executed as if it were $\langle \mathcal{S}_2 c.c0 \rangle_2$. This implicit assumption corresponds to the fact that we supply a sufficient number of identity continuations to a complete program.

In this article we will consider only terms (and values), but not consider complete programs, since every equation over programs can be deduced from equations over terms; in the calculus $\lambda\mathcal{S}_n$, two complete programs p and q are equal if and only if $\langle p \rangle_n$ and $\langle q \rangle_n$ are equal as terms.

4.4. The Target Calculus

We define and analyze the target calculus $\lambda\mathcal{T}_n$ of the extended CPS translation in detail.

The set of terms in $\lambda\mathcal{T}_n$ is the least set which contains all subterms of the targets of the extended CPS translation, and is closed under the $\beta\eta$-reductions. The grammar of the terms in $\lambda\mathcal{T}_n$ is defined by:

$$P, Q ::= x \mid k_i \mid \lambda x.P \mid \lambda k_i.P \mid PQ \quad \text{for} \quad 1 \leq i \leq n+1$$

where x is an ordinary variable (in the source calculi) and k_i is a continuation variable introduced by the CPS translation. We assume that the set of the ordinary variables and that of the continuation variables do not overlap.

A type structure can be introduced to the target terms [9, 11] as follows:

$$\begin{aligned}
\mathtt{Exp}_i &\stackrel{def}{=} \begin{cases} \mathtt{Cont}_{i+1} \to \mathtt{Exp}_{i+1} & \text{for} \quad 0 \leq i \leq n \\ \mathtt{Ans} & \text{for} \quad i = n+1 \end{cases} \\
\mathtt{Cont}_i &\stackrel{def}{=} \mathtt{Value} \to \mathtt{Exp}_i \quad \text{for} \quad 1 \leq i \leq n+1 \\
\mathtt{Value} &\stackrel{def}{=} \mathtt{Value} \to \mathtt{Exp}_0
\end{aligned}$$

Again we need a recursive type for `Value`, but `Ans` can be an arbitrary type, and in fact we can make the answer type parametric as in the work by Thielecke [32]. Since we will use this type structure to define a direct-style translation in Section 6.2, we present the typing rules in detail.

A typing judgment is of the form $\Gamma \vdash P : \sigma$ where P is a term, σ is a type, and 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 ($1 \leq i \leq n+1$):

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

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

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

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

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

If we can prove $\Gamma \vdash P : \sigma$ using the typing rules above, we say that P is a term of type σ in $\lambda\mathcal{T}_n$. For instance, the term $\lambda k_1.\ k_1 x$ ($\equiv [\![x]\!]$) can be typed as follows:

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

A variable of type $\mathtt{Cont}_i$ is called a level-i continuation variable.

In addition to the type structure above, $\lambda\mathcal{T}_n$ enjoys the following important property:

> (Property-#) A target term of type $\mathtt{Exp}_{n+1}$ or $\mathtt{Cont}_{n+1}$ has exactly one free occurrence of level-$(n+1)$ continuation variables, and a target term of other types does not have any free occurrences of level-$(n+1)$ continuation variables.

For example, the target term $[\![\langle M \rangle_n]\!]$ is of type $\mathtt{Exp}_0$ and has no free occurrences of level-$(n+1)$ continuation variables, and it subterm $[\![M]\!]\overline{\theta_{1,n}}(\lambda x.\ \theta_0 x \overline{k_{1,n+1}})$ is of type $\mathtt{Exp}_{n+1}$ and has one occurrence of the level-$(n+1)$ continuation variable k_{n+1}.

Property-# can be explained in this way: recall that we use $(n+1)$-CPS translation for $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$. The first n CPS translations map source terms to pure lambda terms, and the last, $(n+1)$-st CPS translation maps them to terms in CPS, which is therefore the standard CPS translation. It is well known that the standard CPS translation for pure lambda terms needs only one continuation variable, which leads to Property-#. By this property it is possible to refine our type structure as $\mathtt{Exp}_n = \mathtt{Cont}_{n+1} \multimap \mathtt{Exp}_{n+1}$ where $\multimap$ is the linear implication [23].

Property-# does not hold for continuation variables of level$\leq n$, as can be seen by the CPS translation of $[\![\mathcal{S}_i c.M]\!]$ and $[\![\mathcal{C}_i c.M]\!]$, where terms which freely contains continuation variables $\overline{k_{1,i}}$ are substituted for c.

For the type structure with Property-#, we have the following theorem.

THEOREM 1. *Let $\Gamma \vdash P : \sigma$ mean that we can derive $\Gamma \vdash P : \sigma$ using the type structure above, and also Property-# holds for any subterm of the term P. Then we have the following three properties.*

(1) Let M be a term and V be a value in $\lambda\mathcal{S}_n$ or $\lambda\mathcal{C}_n$, and their free variables be $x_1, \cdots, x_n$. Then we have

$$x_1 : \mathtt{Value}, \cdots, x_n : \mathtt{Value} \ \vdash \ [\![M]\!] : \mathtt{Exp}_0, \ \textit{and}$$
$$x_1 : \mathtt{Value}, \cdots, x_n : \mathtt{Value} \ \vdash \ V^* : \mathtt{Value}.$$

(2) (Substitution Property) If $\Gamma \vdash W : \mathtt{Value}$ and $\Gamma, x : \mathtt{Value} \vdash P : \sigma$, we have $\Gamma \vdash P\{x := W\} : \sigma$. Similarly, if $\Gamma \vdash K_i : \mathtt{Cont}_i$ and $\Gamma, k_i : \mathtt{Cont}_i \vdash P : \sigma$ for $1 \leq i \leq n+1$, we have $\Gamma \vdash P\{k_i := K_i\} : \sigma$.

(3) (Subject Reduction) If we have $\Gamma \vdash P : \sigma$, and P reduces to Q by $\beta\eta$-reductions, then we have $\Gamma \vdash Q : \sigma$.

All properties can be proved straightforwardly, and their proofs are omitted.

We introduce $\beta\eta$-equality in the target calculus, and write $\lambda\mathcal{T}_n \vdash P = Q$ if P and Q are equal under $\beta\eta$-equality. Given the CPS translation and the target calculus $\lambda\mathcal{T}_n$ equipped with $\beta\eta$-equality, the source calculus is given an induced semantics called CPS-semantics. The fundamental question addressed in this article is, what is the equality theory that coincides with this CPS semantics, or equivalently, how to axiomatize the CPS semantics in direct style.

5. Axioms for Shift and Reset

In this section we give axioms of the theories $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$, which will be proved sound and complete with respect to the extended CPS translation, namely, two terms are equal in $\lambda\mathcal{S}_n$ (or $\lambda\mathcal{C}_n$) if and only if they are mapped to equal terms in $\lambda\mathcal{T}_n$ by the extended CPS translation.

We think that this kind of axiomatization (called Direct-Style Axiomatization) is important and worth studying for the following reasons.

- Reasoning about source terms (by humans) is usually much easier than reasoning about CPS terms. It is true that we can reason about a source program after CPS translating and $\beta\eta$-reducing it. However, the CPS translation makes the resulting term longer than the source term, and it often obscures the structure of the source term since it is a global translation. If we have a direct-style axiomatization, we do not have to CPS translate every term to reason about its properties, and thus we would get a simpler proof of properties of source programs.

$(\lambda x.M)V = M\{x := V\}$	β_v
$\lambda x.\ Vx = V$	η_v, if $x \notin \mathrm{FV}(V)$
$(\lambda x.E^0[x])M = E^0[M]$	β_Ω, if $x \notin \mathrm{FV}(E^0)$
$\langle V\rangle_i = V$	reset-value
$\langle(\lambda x.M)\langle N\rangle_i\rangle_j = (\lambda x.\langle M\rangle_j)\langle N\rangle_i$	reset-lift

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

– More importantly, having a simple set of axioms helps us to understand the source calculus intuitively and formally, as has been proved by several works in the literature, most notably, by Sabry and Felleisen's direct-style axiomatization of type-free lambda calculus with and without callcc [29]. Their equational axiomatization of the calculus without callcc is equivalent to Moggi's computational lambda calculus [25], and consequently they essentially gave (another) evidence that computational lambda calculus is the canonical choice for the call-by-value setting. Their axioms have another application in A-normal form translation [16].

  We believe that studying direct-style axiomatization is the first and solid step toward good understanding of control operators in the source calculus.

5.1. Axioms

In this section we give the following three sets of axioms:

– common axioms for $\lambda\mathcal{S}_n$ and $\lambda\mathcal{C}_n$ (Figure 1),

– specific axioms for $\lambda\mathcal{S}_n$ (Figure 2), and

– specific axioms for $\lambda\mathcal{C}_n$ (Figure 3).

The theory $\lambda\mathcal{S}_n$ consists of the axioms in Figures 1 and 2, and the theory $\lambda\mathcal{C}_n$ consists of those in Figures 1 and 3.

In the presentation of these axioms, we assume that the levels of all control operators are equal to or less than n, namely, the indices i and j range over the finite set $\{1, 2, \cdots, n\}$. Recall that E^i is a level-i evaluation context, $\langle M\rangle_0 \stackrel{def}{=} M$, and $\mathcal{A}_i \stackrel{def}{=} \lambda x.\mathcal{C}_i\_.x$ where $\_$ is an anonymous variable. Note also that i and j cannot be 0, but $i-1$ and $j-1$ may be 0.

We shall explain axioms in Figure 1. The first three axioms β_v, η_v and β_Ω are those for Moggi's computational lambda calculus[25],

or equivalently, Sabry and Felleisen's axiomatization of pure lambda calculus in call-by-value [29].

The axiom reset-value corresponds to the reduction rule $E[\langle V\rangle_i] \to E[V]$.

The axiom reset-lift does not have a direct counterpart in the operational semantics. To explain its role, let us abbreviate $(\lambda x.M)N$ as $\texttt{let}\ x\ \texttt{be}\ N\ \texttt{in}\ M$, and consider under which condition the following equation (called β-lift) holds:

$$\texttt{let}\ x\ \texttt{be}\ N\ \texttt{in}\ E[M] \;=\; E[\texttt{let}\ x\ \texttt{be}\ N\ \texttt{in}\ M] \quad \text{for } x \notin \mathrm{FV}(E).$$

This equation is useful to guarantee the correctness of certain kinds of program transformations, for instance, closure conversion. Another example is the let-insertion technique used in partial evaluation, one of the successful applications of shift and reset.

In the absence of reset (more precisely, in the case when E is a level-0 evaluation context E^0), the equation holds, as can be proved by:

$$\begin{aligned}\texttt{let}\ x\ \texttt{be}\ N\ \texttt{in}\ E^0[M] \;&=\; \texttt{let}\ x\ \texttt{be}\ N\ \texttt{in}\ E^0[\texttt{let}\ x\ \texttt{be}\ x\ \texttt{in}\ M] \quad \text{by } \beta_v \\ &=\; E^0[\texttt{let}\ x\ \texttt{be}\ N\ \texttt{in}\ M] \quad \text{by } \beta_\Omega\end{aligned}$$

In the presence of reset, however, the equation does not hold in general, as can be seen by CPS translating both sides of the equation. More intuitively, $\langle(\texttt{let}\ x\ \texttt{be}\ \mathcal{S}_1 c.c(c1)\ \texttt{in}\ \langle x\rangle_1) + 5\rangle_1$ evaluates to 11, while $\langle\langle\texttt{let}\ x\ \texttt{be}\ \mathcal{S}_1 c.c(c1)\ \texttt{in}\ x\rangle_1 + 5\rangle_1$ evaluates to 6, so the two terms $\texttt{let}\ x\ \texttt{be}\ \mathcal{S}_1 c.c(c1)\ \texttt{in}\ \langle x\rangle_1$ and $\langle\texttt{let}\ x\ \texttt{be}\ \mathcal{S}_1 c.c(c1)\ \texttt{in}\ x\rangle_1$ cannot be equal even under contextual equivalence.

The axiom reset-lift partially recovers the equation β-lift under a certain condition. Namely, we can prove the following equation by reset-lift when $i \geq j$:

$$\texttt{let}\ x\ \texttt{be}\ \langle N\rangle_i\ \texttt{in}\ E^j[M] = E^j[\texttt{let}\ x\ \texttt{be}\ \langle N\rangle_i\ \texttt{in}\ M] \quad \text{for } x \notin \mathrm{FV}(E^j).$$

In other words, we can lift a local binding (a β-redex) through an evaluation context if the bound term (the above N) is enclosed by a reset of "sufficiently" high level.

Besides the common axioms, there are three specific axioms for shift and three for $\mathcal{C}_i$ shown in Figures 2 and 3, resp. The axiom $\mathcal{S}$-reset is a natural extension of its level-1 counterpart $\mathcal{S}_1 k.\langle M\rangle_1 = \mathcal{S}_1 k.M$. The axiom $\mathcal{S}$-elim is not quite the same as a natural extension of its level-1 counterpart $\mathcal{S}_1 k.kM = M$ (for $k \notin \mathrm{FV}(M)$). In fact, $\mathcal{S}_i k.kM = M$ is not sound for $i > 1$. Danvy and Filinski stated in [9] that the current formulation of shift/reset is not completely satisfactory since a natural equality $\mathcal{S}_2 k.kM = M$ does not hold. We have found that the correct extension of the level-1 equation is the axiom $\mathcal{S}$-elim. By putting $i =$

$\mathcal{S}_i k.\langle M\rangle_i = \mathcal{S}_i k.M$	$\mathcal{S}$-reset
$\mathcal{S}_i k.k\langle M\rangle_{i-1} = \langle M\rangle_{i-1}$	$\mathcal{S}$-elim, if $k \notin \mathrm{FV}(M)$
$\langle E^{j-1}[\mathcal{S}_j k.M]\rangle_i = \langle M\{k := \lambda x.\ \langle E^{j-1}[x]\rangle_j\}\rangle_i$	$\mathcal{S}$-lift, if $k \notin \mathrm{FV}(E^{j-1})$ and $x \notin \mathrm{FV}(kE^{j-1})$

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

$\mathcal{C}_i k.\langle M\rangle_i = \mathcal{C}_i k.M$	$\mathcal{C}$-reset
$\mathcal{C}_i k.k\langle M\rangle_{i-1} = \langle M\rangle_{i-1}$	$\mathcal{C}$-elim, if $k \notin \mathrm{FV}(M)$
$\langle E^{j-1}[\mathcal{C}_j k.M]\rangle_i = \langle M\{k := \lambda x.\ \mathcal{A}_j\langle E^{j-1}[x]\rangle_j\}\rangle_i$	$\mathcal{C}$-lift, if $k \notin \mathrm{FV}(E^{j-1})$ and $x \notin \mathrm{FV}(kE^{j-1})$

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

1 in $\mathcal{S}$-elim, we obtain the level-1 $\mathcal{S}$-elim axiom. The last axiom $\mathcal{S}$-lift is a direct equational formalization of the corresponding reduction rule given in Section 3.2. The specific axioms for $\mathcal{C}_i$ can be explained similarly.

We think that these axioms are natural and simple.

They are natural since many axioms have been obtained by replacing reduction by equality in the operational rules, and the rest can be understood by the operational semantics. Moreover if we put all i's and j's to be 1 in $\lambda\mathcal{S}_n$, we can obtain the axioms for level-1 shift and reset [23][6].

They are simple since many existing works proposed a larger number of more complex axioms for axiomatizing simpler control operators than we use in our axiomatization. For instance, Sabry and Felleisen's axiomatization for callcc has 10 axioms [29], and our axiomatization for level-2 delimited continuation operator has 14 axioms [22]. We think that simplicity of our axioms partly comes from the conceptual simplicity of the definition of shift and reset in terms of CPS translation.

Soundness of these axioms with respect to the extended CPS translation can be proved by calculating both sides of the equations. 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 the

[6] Strictly speaking, there is a slight difference, for the shift operator was formulated as a constant in [23], which can be defined as $\lambda x.\mathcal{S}_1 k.xk$ in $\lambda\mathcal{S}_n$.

level-2 delimited continuation operator $\mathcal{C}_2$ [22], such as:

$$\begin{aligned}
\langle\langle M\rangle_i\rangle_l &= \langle M\rangle_{max(l,i)} && \text{reset-reset}\\
\langle(\lambda x.M)\langle N\rangle_{i-1}\rangle_i &= \langle(\lambda x.\langle M\rangle_j)\langle N\rangle_{i-1}\rangle_i && \text{reset-lift-2, } j\le i\\
\langle\mathcal{C}_i k.M\rangle_j &= \mathcal{C}_i k.M && \text{reset-}\mathcal{C}\text{, } j<i\\
\langle\mathcal{C}_j k.M\rangle_i &= \langle M\{k:=\mathcal{A}_j\}\rangle_i && \mathcal{C}\text{-top, } j\le i\\
(\lambda x.\mathcal{C}_1 k.M)N &= \mathcal{C}_1 k.(\lambda x.M)N && \text{let-}\mathcal{C}_1\text{, } k\neq x
\end{aligned}$$

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

From now on, we will mainly investigate $\lambda\mathcal{C}_n$. After proving its soundness and completeness, we will come back to $\lambda\mathcal{S}_n$. The use of $\lambda\mathcal{C}_n$ in our proof is not essential, and we can directly prove the completeness for shift as well. The use of $\lambda\mathcal{C}_n$ is motivated by the fact that $\mathcal{C}_i$ has a slightly simpler CPS translation, and its expressive power is the same as that of $\mathcal{S}_i$.

5.2. Derivable Equations

THEOREM 2. *The equations reset-$\mathcal{C}$, $\mathcal{C}$-top, reset-reset, reset-lift-2, and let-$\mathcal{C}_1$ as well as the following equations are derivable in $\lambda\mathcal{C}_n$ where $k_1,\cdots,k_i,x$ are fresh variables, and k is not bound in the context C in $\mathcal{C}$-abort.*

$$\begin{aligned}
E^j[\mathcal{A}_i\langle M\rangle_{i-1}] &= \mathcal{A}_i\langle M\rangle_{i-1} && \mathcal{A}\text{-abort, } j<i\\
\mathcal{C}_i k.C[E^j[kV]] &= \mathcal{C}_i k.C[kV] && \mathcal{C}\text{-abort, } j<i\\
E^j[\mathcal{C}_i k.M] &= \overline{\mathcal{C}k_{1,i}}.\ M\{k:=N\} && \textit{telescope, } j<i
\end{aligned}$$

where N is $\lambda x.\ k_i\langle k_{i-1}\cdots\langle k_1(E^j[x])\rangle_1\cdots\rangle_{i-1}$.

Proof. ($\mathcal{C}$-top) Putting $E^{j-1} \equiv [\,]$ and $i = j$ in $\mathcal{C}$-lift and using reset-value and η_v, we have $\mathcal{C}$-top.
(reset-reset) By putting $M \equiv x$ in reset-lift, we have $\langle\langle(\lambda x.x)N\rangle_i\rangle_j = (\lambda x.\langle x\rangle_j)\langle N\rangle_i$. Using reset-value and β_Ω (in the form of $(\lambda x.x)L = L$ for any L), we obtain $\langle\langle N\rangle_i\rangle_j = \langle N\rangle_i$ for $j \le i$, which constitutes a half of reset-reset.

We will prove the remaining half of reset-reset, namely, $\langle\langle N\rangle_j\rangle_i = \langle N\rangle_i$ for $j \le i$. By putting $E^{j-1} \equiv (\lambda x.N)[\,]$ and $M \equiv kx$ in $\mathcal{C}$-lift, we have $\langle(\lambda x.N)(\mathcal{C}_j k.kx)\rangle_i = \langle(\lambda y.\mathcal{A}_j\langle(\lambda x.N)y\rangle_j)x\rangle_i$. Using reset-value, $\mathcal{C}$-elim and β_v, the lefthand side is equal to $\langle N\rangle_i$, and using β_v,

the righthand side is equal to $\langle \mathcal{A}_j \langle N \rangle_j \rangle_i$. We can further rewrite the righthand side as:

$$\begin{aligned}
\langle \mathcal{A}_j \langle N \rangle_j \rangle_i &= \langle \mathcal{C}_j k.\ k \langle N \rangle_j \rangle_i \quad \text{by } \mathcal{C}\text{-top} \\
&= \langle \mathcal{C}_j k.\ k \langle \langle N \rangle_j \rangle_{j-1} \rangle_i \quad \text{by the half of reset-reset} \\
&= \langle \langle \langle N \rangle_j \rangle_{j-1} \rangle_i \quad \text{by } \mathcal{C}\text{-elim} \\
&= \langle \langle N \rangle_j \rangle_i \quad \text{by the half of reset-reset}
\end{aligned}$$

where we used the already proved half of reset-reset. Hence we have $\langle N \rangle_i = \langle \langle N \rangle_j \rangle_i$ for $j \leq i$, proving the remaining half of the reset.
(reset-lift-2) We have the following derivation for $0 \leq l \leq i$.

$$\begin{aligned}
\langle (\lambda x. \langle M \rangle_l) \langle N \rangle_{i-1} \rangle_i &= \langle (\lambda x. \langle M \rangle_l)(\mathcal{C}_i k. k \langle N \rangle_{i-1}) \rangle_i \quad \text{by } \mathcal{C}\text{-elim} \\
&= \langle (\lambda y. \mathcal{A}_i \langle (\lambda x. \langle M \rangle_l) y \rangle_i) \langle N \rangle_{i-1} \rangle_i \quad \text{by } \mathcal{C}\text{-lift} \\
&= \langle (\lambda x. \mathcal{A}_i \langle M \rangle_i) \langle N \rangle_{i-1} \rangle_i \quad \text{by } \beta_v, \text{ reset-reset}
\end{aligned}$$

The final result does not depend on l, and by putting $l = 0$ and $l = j$, we can obtain reset-lift-2.

For the remaining equations in Theorem 2, we prepare two key equations. First, we derive by $\mathcal{C}$-elim and $\mathcal{C}$-reset:

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

Iterating this process i-times we obtain:

$$N = \overline{\mathcal{C} k_{1,i}}. \langle k_i \langle k_{i-1} \cdots \langle k_1 N \rangle_1 \cdots \rangle_{i-1} \rangle_i \quad \text{(key eq.-1)}$$

For $j < i$, by putting $N \equiv E^j[\mathcal{C}_i k. M]$ in the key eq.-1 and using $\mathcal{C}$-lift and reset-reset, we obtain the following equation ($k_1, \cdots, k_i, x$ are fresh variables):

$$\begin{aligned}
E^j[\mathcal{C}_i k. M] &= \overline{\mathcal{C} k_{1,i}}. \langle M \{ k := \lambda x. \mathcal{A}_i \langle k_i \langle \cdots \langle k_1 (E^j[x]) \rangle_1 \cdots \rangle_{i-1} \rangle_i \} \rangle_i \\
&\qquad \text{(key eq.-2, for } j < i)
\end{aligned}$$

(reset-$\mathcal{C}$) By putting $E^j \equiv [\ ]$ and $E^j \equiv \langle [\ ] \rangle_j$ in the key eq.-2, we have that the righthand sides of these two cases are equal by reset-value, hence we obtain $\mathcal{C}_i k. M = \langle \mathcal{C}_i k. M \rangle_j$.
($\mathcal{A}$-abort) We first note that, if $j < i$, 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 $k \notin \mathrm{FV}(M)$, then the right-hand side of the key eq.-2 does not contain E^j, hence $E^j[\mathcal{C}_i \_. M] = \mathcal{C}_i \_. M$. Then we can compute as follows (for $j < i$):

$$\begin{aligned}
E^j[\mathcal{A}_i \langle M \rangle_{i-1}] &\equiv E^j[(\lambda x. \mathcal{C}_i _. x) \langle M \rangle_{i-1}] \\
&= (\lambda x. E^j[\mathcal{C}_i _. x]) \langle M \rangle_{i-1} \qquad \text{by the above equation} \\
&= (\lambda x. \mathcal{C}_i _. x) \langle M \rangle_{i-1} \qquad \text{by the key eq.-2} \\
&\equiv \mathcal{A}_i \langle M \rangle_{i-1}
\end{aligned}$$

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

$$\begin{aligned}
& \mathcal{C}_i k.C[E^j[kV]] \\
= \ & \overline{\mathcal{C}k_{1,i}}.\langle (C[E^j[kV]])\{k := \lambda x.\mathcal{A}_i\langle k_i\langle \cdots \langle k_1 x\rangle_1 \cdots\rangle_{i-1}\rangle_i\}\rangle_i \\
& \quad \text{by the key eq.-2} \\
= \ & \overline{\mathcal{C}k_{1,i}}.\langle C[E^j[\mathcal{A}_i\langle k_i\langle \cdots \langle k_1 V\rangle_1 \cdots\rangle_{i-1}\rangle_i]]\rangle_i \quad \text{by } \beta_v \\
= \ & \overline{\mathcal{C}k_{1,i}}.\langle C[\mathcal{A}_i\langle k_i\langle \cdots \langle k_1 V\rangle_1 \cdots\rangle_{i-1}\rangle_i]\rangle_i \quad \text{by } \mathcal{A}\text{-abort}
\end{aligned}$$

Since the final result does not contain E^j, we obtain $\mathcal{C}$-abort.
(telescope) We first prove $\mathcal{A}_i\langle M\rangle_{i-1} = \mathcal{A}_i\langle M\rangle_i$. In the following derivation, $k_1, \cdots, k_i$ are fresh variables.

$$\begin{aligned}
\mathcal{A}_i\langle M\rangle_{i-1} \ &= \ \overline{\mathcal{C}k_{1,i}}.\langle k_i\langle \cdots \langle k_1(\mathcal{A}_i\langle M\rangle_{i-1})\rangle_1 \cdots\rangle_{i-1}\rangle_i \\
& \qquad \text{by the key eq.-1} \\
&= \ \overline{\mathcal{C}k_{1,i}}.\langle \mathcal{A}_i\langle M\rangle_{i-1}\rangle_i \quad \text{by } \mathcal{A}\text{-abort} \\
&= \ \overline{\mathcal{C}k_{1,i}}.\langle M\rangle_i
\end{aligned}$$

The last equation has been derived in the proof of reset-reset. Similarly we have $\mathcal{A}_i\langle M\rangle_i = \overline{\mathcal{C}k_{1,i}}.M$, and hence $\mathcal{A}_i\langle M\rangle_i = \mathcal{A}_i\langle M\rangle_{i-1}$.

The righthand side of the key eq.-2 has a subterm $\mathcal{A}_i\langle k_i\langle \cdots\rangle_{i-1}\rangle_i$, which is equal to $\mathcal{A}_i\langle k_i\langle \cdots\rangle_{i-1}\rangle_{i-1}$ by the equation above. Since $\mathcal{A}_i\langle [\ ]\rangle_{i-1}$ is a level-$(i-1)$ evaluation context, we can apply $\mathcal{C}$-abort to eliminate it from the righthand side of the key eq.-2 (note that k_i is bound by $\mathcal{C}_i k_i$ in the the key eq.-2), and finally obtain the telescope axiom[7]:

$$E^j[\mathcal{C}_i k.M] = \overline{\mathcal{C}k_{1,i}}.\ M\{k := \lambda x.k_i\langle k_{i-1} \cdots \langle k_1(E^j[x])\rangle_1 \cdots\rangle_{i-1}\}$$

We also have $\mathcal{A}_i\langle M\rangle_{i-1} = \mathcal{C}_{i\_}.M$ from the above derivation and telescope.
(let-$\mathcal{C}_1$) This equation is not needed in this article, and its verification is left for the reader.

This finishes the proof of Theorem 2.

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 $\lambda\mathcal{T}_n$. In this section we give a proof for these results.

[7] This is a generalized version of Murthy's telescope axiom [26].

6.1. Proof Methods for Completeness

We first review the proof methods for this kind of completeness proposed in the literature, and then give an overview of our proof method.

A traditional way to prove completeness is to find an inverse $\_^\dagger$ of the CPS translation that satisfies the following two properties:

(1) For any source term M, we can prove $[\![M]\!]^\dagger = M$ in the source calculus.

(2) For any target terms T and T', if $T = T'$ in the target calculus, we can prove $T^\dagger = T'^\dagger$ in the source calculus.

In fact, Sabry and Felleisen used this strategy to prove the completeness of their axioms for the calculus with `callcc`[29], and we used it for the calculus with level-1 shift and reset [23].

Sabry proposed a new proof method to prove the completeness of the calculus with level-1 shift and lazy reset [28]. Let us illustrate his method by an example.

In the standard CPS translation, a source term of the form MN is CPS translated to:

$$[\![MN]\!] \stackrel{def}{=} \lambda k.[\![M]\!](\lambda m.[\![N]\!](\lambda n.mnk)))$$

Sabry's key observation is that, as long as the equality of the target calculus is concerned, we do not have to use the ordinary lambda terms to represent the CPS translation. Instead, we can represent it as:

$$[\![MN]\!]' = \mathtt{get}\ k.\mathtt{send}(\lambda m.\mathtt{send}(\lambda n.\mathtt{send}(k, mn), [\![N]\!]'), [\![M]\!]')$$

with two functions `get` and `send` that satisfy the following two equations corresponding to the β and η equations:

$$\begin{aligned} \mathtt{send}(K, \mathtt{get}\ k.P) &= P\{k := K\} \\ \mathtt{get}\ k.\mathtt{send}(k, T) &= T \quad \text{if } k \notin \mathrm{FV}(T). \end{aligned}$$

Note that, if $[\![M]\!] = [\![N]\!]$ under $\beta\eta$-equality, then we have $[\![M]\!]' = [\![N]\!]'$ under $\beta\eta$-equality with the two $\beta\eta$-like equations for `send`/`get` (this property will be referred as Property (i)).

The generalized CPS translation is an *abstract* CPS translation that can be interpreted in several ways:

- An interpretation in the target calculus is defined by $\mathtt{send}(K, T) \stackrel{def}{=} TK$ and $\mathtt{get}\ k.P \stackrel{def}{=} \lambda k.P$, then $[\![M]\!]'$ is identical to the standard CPS translation $[\![M]\!]$.

– We can interpret it in the source calculus by defining $\texttt{send}(K, T) \stackrel{def}{=} KT$ and $\texttt{get}\ k.P \stackrel{def}{=} \texttt{callcc}(\lambda k.P)$. The notable feature of this implementation is that the generalized CPS translation is the identity on source terms, namely, $[\![M]\!]' = M$ under the direct-style axioms (Property (ii)).

The completeness for the standard CPS translation follows from Properties (i) and (ii).

One may say that Sabry's new method is merely a reformulation of the traditional one, since the non-trivial implementation corresponds to an inverse translation in the traditional method, and Properties (i) and (ii) correspond to Properties (2) and (1), resp. However, we think that his method is valuable, since finding a suitable inverse translation from a huge search space is often very difficult, as was the case for our axiomatization for level-1 shift and reset, and his method is a good guide to this exploration.

Instead of directly following his method, we will use the type structure of the target calculus to reflect Sabry's proof method. The use of types has two benefits: first, we can capture the essence of Sabry's use of abstract datatypes. Second, it makes explicit hidden details in Sabry's proof. As we saw, Sabry replaced *some* occurrences of λ and application by `get` and `send`, leaving others intact, but that this selective replacement is harmless was not formally expressed nor proved in his paper. Since our source calculus with layered control operators is much more complex than his source calculus, we need a way to represent this property explicitly. The subject reduction property of the type system ensures the non-interference between them.

There is a subtle, but important reason for us not to follow Sabry's method directly (see below), hence we will use a traditional proof method which defines an inverse translation (called a direct-style translation) and prove the two properties (1) and (2) (Theorems 5 and 3, resp.). These two properties subsume the completeness of $\lambda\mathcal{S}_n$.

6.2. Direct-Style Translation

We first define a direct-style (DS) translation $\_^\dagger$ from the target calculus $\lambda\mathcal{T}_n$ to the source calculus $\lambda\mathcal{C}_n$. Later, we will prove that it is an inverse of the extended CPS translation.

The following eight clauses correspond to the type inference rules in Section 4.4. Note that, in $\lambda\mathcal{T}_n$, the index i can be up to $n+1$, that is,

$1 \leq i \leq n+1$.

$$(WW')^\dagger \stackrel{def}{=} W^\dagger W'^\dagger \qquad (\lambda k_i.T_i)^\dagger \stackrel{def}{=} \begin{cases} \mathcal{C}_i k_i.T_i^\dagger, & \text{if } i \leq n \\ T_i^\dagger, & \text{o.w.} \end{cases}$$

$$(T_{i-1}K_i)^\dagger \stackrel{def}{=} K_i^\dagger \langle T_{i-1}^\dagger \rangle_{i-1} \qquad (K_i W)^\dagger \stackrel{def}{=} K_i^\dagger W^\dagger$$

$$k_i^\dagger \stackrel{def}{=} \begin{cases} k_i, & \text{if } i \leq n \\ \lambda x.x, & \text{o.w.} \end{cases} \qquad (\lambda x.T_i)^\dagger \stackrel{def}{=} \begin{cases} \lambda x.\langle T_i^\dagger \rangle_i, & \text{if } i \leq n \\ \lambda x.T_i^\dagger, & \text{o.w.} \end{cases}$$

$$x^\dagger \stackrel{def}{=} x \qquad (\lambda x.T_0)^\dagger \stackrel{def}{=} \lambda x.T_0^\dagger$$

where x, k_i are variables of type $\texttt{Value}$ and $\texttt{Cont}_i$, resp, and W, W', T_i, K_i are terms of $\texttt{Value}$, $\texttt{Value}$, $\texttt{Exp}_i$, and $\texttt{Cont}_i$, resp.

Let us briefly explain how we obtained the DS translation above. As in Sabry's method, we generalize the following four terms in the target calculus as:

$$\begin{aligned} \lambda k_i.T_i &\text{ is replaced by } \texttt{get}_i^1\ k_i.\ T_i \\ T_{i-1}K_i &\text{ is replaced by } \texttt{send}_i^1(K_i, T_{i-1}) \\ \lambda x.T_i &\text{ is replaced by } \texttt{get}_i^2\ x.\ T_i \\ K_i W &\text{ is replaced by } \texttt{send}_i^2(K_i, W) \end{aligned}$$

for $1 \leq i \leq n+1$. The pair of functions $(\texttt{send}_i^1, \texttt{get}_i^1)$ are interface functions of the type $\texttt{Exp}_{i-1} = \texttt{Cont}_i \to \texttt{Exp}_i$, and $(\texttt{send}_i^2, \texttt{get}_i^2)$ are those of the type $\texttt{Cont}_i = \texttt{Value} \to \texttt{Exp}_i$. Each of these pairs must satisfy $\beta\eta$-like equations as before. We do not generalize other terms such as terms of type $\texttt{Value} = \texttt{Value} \to \texttt{Exp}_0$.

We then interpret these terms in the source calculus. Here we ignore the case $i = n+1$ for simplicity. One can see that the following definitions can be obtained (after a few trial and errors[8]):

$$\begin{aligned} \texttt{get}_i^1\ k_i.\ T_i &\stackrel{def}{=} \mathcal{C}_i k_i.T_i \\ \texttt{send}_i^1(K_i, T_{i-1}) &\stackrel{def}{=} K_i \langle T_{i-1} \rangle_{i-1} \\ \texttt{get}_i^2\ x.\ T_i &\stackrel{def}{=} \lambda x.\langle T_i \rangle_i \\ \texttt{send}_i^2(K_i, W) &\stackrel{def}{=} K_i W \end{aligned}$$

As one can see in the definition of $\texttt{get}_i^1$ (and $(\lambda k_i.T_i)^\dagger$), an abstraction of a continuation variable is sent back to a control operator $\mathcal{C}_i$. In other words, the level-i continuation is explicitly grabbed by $\mathcal{C}_i$ in the result

[8] There are other candidates, but since each pair of $\texttt{send}/\texttt{get}$ must satisfy $\beta\eta$-like equalities locally, our search space is rather limited.

of the direct-style translation, which Sabry called *continuation grabbing style*.

The DS translation is a simple reformulation of this interpretation except that we have to treat the case $i = n+1$ separately. It is necessary to avoid having the level-$(n+1)$ control operators in the image of the DS translation. Note that we have level-$(n+1)$ continuation variables in the target calculus, while our source calculus have level $\leq n$ control operators.

For this purpose, the DS translation simply *forgets* about the level-$n+1$ continuation variables, in that the abstraction by k_{n+1} is omitted in $(\lambda k_i.T_i)^\dagger$, and the variable k_{n+1} is mapped to the identity function $\lambda x.x$. The forgetful translation works well thanks to Property-# of the target calculus. Since the level-$(n+1)$ continuation variable is linear, and moreover every subterm of a target term has at most one free occurrence of such variables, we do not have to remember the variable through the inverse translation.

This trick appeared in our completeness proof for level-1 shift and reset [23], and in this article we generalize it to higher levels. A similar technique appeared in Berdine, O'Hearn, Reddy, and Thielecke's work where they investigated linearity in the target calculus and the inverse translation based on it [4].

Since the trick does not fit in Sabry's proof method, it seems difficult to present our proof as an instance of Sabry's method, and it is the reason why we chose the traditional proof method.

We also remark that, we can define an equivalent DS translation in terms of shift and reset instead of $\mathcal{C}_i$ and reset. However, the results in terms of shift are longer than those by $\mathcal{C}_i$, and this is the (only) reason why we chose to develop the theory for $\mathcal{C}_i$ rather than directly working on the theory for shift.

In the next section, we prove that the DS translation enjoys Properties (1) and (2) mentioned at the beginning of Section 6.1.

6.3. Completeness Proof for $\lambda\mathcal{C}_n$

We first prove Property (2), namely, $\_^\dagger$ respects $\beta\eta$-equality in the target calculus.

THEOREM 3. *Let T_0 and T_0' be terms of type $\mathtt{Exp}_0$ in $\lambda\mathcal{T}_n$. If $\lambda\mathcal{T}_n \vdash T_0 = T_0'$, then $\lambda\mathcal{C}_n \vdash {T_0}^\dagger = {T_0'}^\dagger$.*

This theorem is subsumed by the following general theorem by putting $i = 0$ in the clause (1).

THEOREM 4. *Let $0 \leq i \leq n$ and $1 \leq j \leq n$, T_i and T'_i be terms of type $\mathtt{Exp}_i$, K_j and K'_j be terms of type $\mathtt{Cont}_j$, and W and W' be terms of type* `Value` *in $\lambda\mathcal{T}_n$, and let M be a term in $\lambda\mathcal{C}_n$. We have:*

(1) If $\lambda\mathcal{T}_n \vdash T_i = T'_i$, then $\lambda\mathcal{C}_n \vdash \langle {T_i}^\dagger \rangle_i = \langle {T'_i}^\dagger \rangle_i$.

(2) If $\lambda\mathcal{T}_n \vdash K_j = K'_j$, then $\lambda\mathcal{C}_n \vdash \langle {K_j}^\dagger \langle M \rangle_{j-1} \rangle_j = \langle {K'_j}^\dagger \langle M \rangle_{j-1} \rangle_j$.

(3) If $\lambda\mathcal{T}_n \vdash W = W'$, then $\lambda\mathcal{C}_n \vdash W^\dagger = W'^\dagger$.

Let T_{n+1} and T'_{n+1} be terms of type $\mathtt{Exp}_{n+1}$, K_{n+1} and K'_{n+1} be terms of type $\mathtt{Cont}_{n+1}$ in $\lambda\mathcal{T}_n$, and let M be a term in $\lambda\mathcal{C}_n$. We have:

(1') If $\lambda\mathcal{T}_n \vdash T_{n+1} = T'_{n+1}$, then $\lambda\mathcal{C}_n \vdash {T_{n+1}}^\dagger = {T'_{n+1}}^\dagger$.

(2') If $\lambda\mathcal{T}_n \vdash K_{n+1} = K'_{n+1}$, then $\lambda\mathcal{C}_n \vdash {K_{n+1}}^\dagger \langle M \rangle_n = {K'_{n+1}}^\dagger \langle M \rangle_n$.

In order to prove this theorem, we need three lemmas about substitution. The first one is for an ordinary variable (Lemma 1), and the other two are for a continuation variable. As in the definition of the DS translation, we divide the substitution property for a continuation variable k_l into two cases, namely, $1 \leq l \leq n$ and $l = n + 1$, which correspond to Lemmas 2 and 3, resp.

LEMMA 1. *For $0 \leq i \leq n + 1$ and $1 \leq j \leq n + 1$, let X be a term of type $\mathtt{Exp}_i$, $\mathtt{Cont}_j$, or* `Value`*, and W be a term of type* `Value` *in $\lambda\mathcal{T}_n$. Then we have:*

$$\lambda\mathcal{C}_n \vdash (X\{x := W\})^\dagger \;=\; X^\dagger\{x := W^\dagger\}$$

This lemma is proved by straightforward induction on X.

LEMMA 2. *For $0 \leq i \leq n$, $1 \leq j \leq n$, and $1 \leq l \leq n$, let T_i, T_{n+1}, K_j, K_{n+1}, W and K'_l be terms of type $\mathtt{Exp}_i$, $\mathtt{Exp}_{n+1}$, $\mathtt{Cont}_j$, $\mathtt{Cont}_{n+1}$,* `Value` *and $\mathtt{Cont}_l$, and k_l be a continuation variable of type $\mathtt{Cont}_l$, and M be a term in $\lambda\mathcal{C}_n$. Then we have:*

$$\begin{array}{lll}
\lambda\mathcal{C}_n & \vdash & \langle (T_i\{k_l := K'_l\})^\dagger \rangle_i = \langle {T_i}^\dagger\{k_l := L\} \rangle_i \\
\lambda\mathcal{C}_n & \vdash & (T_{n+1}\{k_l := K'_l\})^\dagger = {T_{n+1}}^\dagger\{k_l := L\} \\
\lambda\mathcal{C}_n & \vdash & \langle (K_j\{k_l := K'_l\})^\dagger \langle M \rangle_{j-1} \rangle_j = \langle ({K_j}^\dagger\{k_l := L\}) \langle M \rangle_{j-1} \rangle_j \\
\lambda\mathcal{C}_n & \vdash & (K_{n+1}\{k_l := K'_l\})^\dagger \langle M \rangle_n = ({K_{n+1}}^\dagger\{k_l := L\}) \langle M \rangle_n \\
\lambda\mathcal{C}_n & \vdash & (W\{k_l := K'_l\})^\dagger = W^\dagger\{k_l := L\} \\
 & & \textit{where } L = \lambda x.\mathcal{A}_l \langle {K'_l}^\dagger x \rangle_l
\end{array}$$

Proof. These equations can be proved by simultaneous induction on the structure of T_i, T_{n+1}, K_j, K_{n+1} and W. We prove only the third

equation when $j = l$ and $K_j \equiv k_l$:

$$\begin{aligned}
\text{lefthand side} &= \langle {K_j}^\dagger \langle M \rangle_{j-1} \rangle_j \\
\text{righthand side} &= \langle (\lambda x.\mathcal{A}_j \langle {K_j}^\dagger x \rangle_j) \langle M \rangle_{j-1} \rangle_j \\
&= \langle (\lambda x.\langle \mathcal{A}_j \langle {K_j}^\dagger x \rangle_j \rangle_j) \langle M \rangle_{j-1} \rangle_j \quad \text{by reset-lift-2} \\
&= \langle (\lambda x.\langle {K_j}^\dagger x \rangle_j) \langle M \rangle_{j-1} \rangle_j \quad \text{by (*)} \\
&= \langle (\lambda x.{K_j}^\dagger x) \langle M \rangle_{j-1} \rangle_j \quad \text{by reset-lift-2} \\
&= \langle {K_j}^\dagger \langle M \rangle_{j-1} \rangle_j \quad \text{by } \beta_\Omega
\end{aligned}$$

In the derivation of (*), we used $\langle \mathcal{A}_j \langle N \rangle_j \rangle_j = \langle N \rangle_j$ which was proved during the proof of reset-reset in Theorem 2.

For the fourth equation when K_{n+1} is a continuation variable, it cannot be k_l since their levels are different, hence the substitution is vacuous and we can easily prove the equation.

The other cases of Lemma 2 can be proved similarly.

The next lemma is for the substitution of the level-$(n+1)$ continuation variable k_{n+1}, where we only have to treat T_{n+1} and K_{n+1}, since by Property-#, the continuation variable k_{n+1} occurs freely in these terms only.

LEMMA 3. *Let T_{n+1}, K_{n+1} and K'_{n+1} be terms of type $\mathtt{Exp}_{n+1}\ \mathtt{Cont}_{n+1}$ and $\mathtt{Cont}_{n+1}$, resp., k_{n+1} be a continuation variable of type $\mathtt{Cont}_{n+1}$, and M be a term in $\lambda\mathcal{C}_n$. Then we have:*

$$\begin{aligned}
\lambda\mathcal{C}_n &\vdash (T_{n+1}\{k_{n+1} := K'_{n+1}\})^\dagger = {K'_{n+1}}^\dagger \langle {T_{n+1}}^\dagger \rangle_n \\
\lambda\mathcal{C}_n &\vdash (K_{n+1}\{k_{n+1} := K'_{n+1}\})^\dagger \langle M \rangle_n = {K'_{n+1}}^\dagger \langle {K_{n+1}}^\dagger \langle M \rangle_n \rangle_n
\end{aligned}$$

Proof. This lemma is proved by induction on T_{n+1} and K_{n+1}.

We only prove the second equation when $K_{n+1} = \lambda x.T'_{n+1}$ for some T'_{n+1} of type $\mathtt{Exp}_{n+1}$ and leave other cases to the reader:

$$\begin{aligned}
\text{lefthand side} &\equiv (\lambda x.(T'_{n+1}\{k_{n+1} := K'_{n+1}\})^\dagger) \langle M \rangle_n \\
&= (\lambda x.{K'_{n+1}}^\dagger \langle {T_{n+1}}^\dagger \rangle_n) \langle M \rangle_n \quad \text{by induction hypothesis} \\
\text{righthand side} &\equiv {K'_{n+1}}^\dagger \langle (\lambda x.{T_{n+1}}^\dagger) \langle M \rangle_n \rangle_n
\end{aligned}$$

On the other hand, we can prove $(\lambda x.L\langle N \rangle_n)\langle M \rangle_n = L\langle (\lambda x.N)\langle M \rangle_n \rangle_n$ (for $x \notin \mathrm{FV}(L)$) using β-lift in Section 5.1 (which is in turn proved by β_Ω) and reset-lift. Consequently, we have that both sides of the second equation are provably equal in $\lambda\mathcal{C}_n$.

Other cases can be proved similarly.

Proof of Theorem 4. This is proved by simultaneous induction on the structure of T_i, T_{n+1}, K_j, K_{n+1} and W.

Suppose $\lambda\mathcal{T}_n \vdash T_i = T'_i$, $\lambda\mathcal{T}_n \vdash T_{n+1} = T'_{n+1}$, $\lambda\mathcal{T}_n \vdash K_j = K'_j$, $\lambda\mathcal{T}_n \vdash K_{n+1} = K'_{n+1}$, and $\lambda\mathcal{T}_n \vdash W = W'$. It suffices to prove the theorem when T'_i, T'_{n+1}, K'_j, K'_{n+1} and W' are obtained by one-step-reduction from the corresponding terms. There are several cases to consider.

(Case 1: reduction of a subterm) This is the case when we reduce a proper subterm of T_i etc. By careful inspection of the shape of the result of the DS translation, we can show that the statement of Theorem 4 is sufficient to make the whole induction go through.

For instance, suppose $T_i = \lambda k_i.T''$ for $1 \leq i \leq n$, and T'_i is obtained by one-step reduction of a redex in T'', then we have: $\langle {T_i}^\dagger \rangle_i \equiv \langle \mathcal{C}_i k_i.{T''}^\dagger \rangle_i = \langle \mathcal{C}_i k_i.\langle {T''}^\dagger \rangle_i \rangle_i$ under $\lambda\mathcal{C}_n$, hence we can apply the induction hypothesis on T''. Also all we need about a term K_j of type $\mathtt{Cont}_j$ is the case when it appears in a subterm $\langle {K_j}^\dagger \langle M \rangle_{i-1} \rangle_j$, so the induction hypothesis suffices.

(Case 2: the whole term is a β-redex or an η-redex which is being contracted) There are six cases to consider.

(Case 2-β-1) $1 \leq i \leq n$, $T_i \equiv (\lambda k_i.T)K_i$, and $T'_i \equiv T\{k_i := K_i\}$.

We can calculate as follows:

$$\begin{aligned}
\langle {T_i}^\dagger \rangle_i &\equiv \langle {K_i}^\dagger \langle \mathcal{C}_i k_i.T^\dagger \rangle_{i-1} \rangle_i \\
&= \langle T^\dagger\{k_i := \lambda x.\mathcal{A}_i \langle {K_i}^\dagger x \rangle_i\} \rangle_i \quad \text{by } \mathcal{C}\text{-lift,reset-value} \\
&= \langle {T'_i}^\dagger \rangle_i \quad \text{by Lemma 2}
\end{aligned}$$

(Case 2-β-2) $T_{n+1} \equiv (\lambda k_{n+1}.T)K_{n+1}$, and $T'_{n+1} \equiv T\{k_{n+1} := K_{n+1}\}$.

We can calculate as ${T_{n+1}}^\dagger \equiv {K_{n+1}}^\dagger \langle T^\dagger \rangle_n = {T'_{n+1}}^\dagger$ by Lemma 3.

(Case 2-β-3) $T_i \equiv (\lambda x.T)W$ and $T'_i \equiv T\{x := W\}$ for $0 \leq i \leq n+1$.

This case can be proved using Lemma 1.

(Case 2-η-1) $T_i \equiv \lambda k_{i+1}.T'_i k_{i+1}$, $0 \leq i \leq n$, and $k_{i+1} \notin \mathrm{FV}(T'_i)$.

If $i \leq n-1$, then we have $\langle {T_i}^\dagger \rangle_i \equiv \langle \mathcal{C}_{i+1} k_{i+1}.k_{i+1} \langle {T'_i}^\dagger \rangle_i \rangle_i = \langle \langle {T'_i}^\dagger \rangle_i \rangle_i = \langle {T'_i}^\dagger \rangle_i$.

If $i = n$, then we have $\langle {T_n}^\dagger \rangle_n \equiv \langle (\lambda x.x) \langle {T'_n}^\dagger \rangle_n \rangle_n = \langle {T'_n}^\dagger \rangle_n$

(Case 2-η-2) $K_i \equiv \lambda x.K'_i x$, $1 \leq i \leq n+1$, and $x \notin \mathrm{FV}(K'_i)$.

Take any term M in $\lambda\mathcal{C}_n$.

If $i \leq n$, we have $\langle {K_i}^\dagger \langle M \rangle_{i-1} \rangle_i \equiv \langle (\lambda x.\langle {K'_i}^\dagger x \rangle_i) \langle M \rangle_{i-1} \rangle_i$, and by reset-lift-2 and η_v, it is equal to $\langle {K'_i}^\dagger \langle M \rangle_{i-1} \rangle_i$.

If $i = n+1$, we have ${K_{n+1}}^\dagger \langle M \rangle_n \equiv (\lambda x.{K'_{n+1}}^\dagger x) \langle M \rangle_n = {K'_{n+1}}^\dagger \langle M \rangle_n$.

(Case 2-η-3) $W \equiv \lambda x.W'x$ and $x \notin \mathrm{FV}(W')$.

This case is easily proved, completing the proof of Theorem 4.

The next theorem corresponds to Property (1), that is, $\_^\dagger$ is a (left) inverse of the extended CPS translation.

THEOREM 5. *If M is a term in $\lambda\mathcal{C}_n$, then $\lambda\mathcal{C}_n \vdash [\![M]\!]^\dagger = M$.*

Proof. We will prove $\lambda\mathcal{C}_n \vdash [\![M]\!]^\dagger = M$ and $\lambda\mathcal{C}_n \vdash V^{*\dagger} = V$ by simultaneous induction on a term M and a value V in $\lambda\mathcal{C}_n$. We list proofs of a few interesting cases.
(Case: $M = N_1N_2$)

$$\begin{aligned}
[\![N_1N_2]\!]^\dagger &= \mathcal{C}_1k_1.(\lambda m.\langle(\lambda n.\langle k_1(mn)\rangle_1)[\![N_2]\!]^\dagger\rangle_1)[\![N_1]\!]^\dagger \\
&= \mathcal{C}_1k_1.(\lambda m.\langle(\lambda n.\langle k_1(mn)\rangle_1)N_2\rangle_1)N_1 \quad \text{by induction hypothesis} \\
&= \mathcal{C}_1k_1.\langle\lambda m.\langle\langle k_1(mN_2)\rangle_1\rangle_1N_1\rangle_1 \quad \text{by reset-lift-2,}\beta_\Omega\text{,}\mathcal{C}\text{-reset} \\
&= \mathcal{C}_1k_1.\langle k_1(N_1N_2)\rangle_1 \quad \text{by reset-lift-2,}\beta_\Omega\text{,}\mathcal{C}\text{-reset} \\
&= N_1N_2 \quad \text{by } \mathcal{C}\text{-reset,}\mathcal{C}\text{-elim}
\end{aligned}$$

(Case: $M = \langle N\rangle_i$) We can easily show ${\theta_i}^\dagger = \lambda x.x$.
When $1 \le i \le n-1$, we have:

$$\begin{aligned}
[\![\langle N\rangle_i]\!]^\dagger &= \overline{\mathcal{C}k_{1,i+1}}.(\lambda x.\langle k_{i+1}\langle k_i\cdots\langle k_1x\rangle_1\cdots\rangle_i\rangle_{i+1}) \\
&\qquad \langle(\lambda x.x)\langle(\lambda x.x)\cdots(\lambda x.x)\langle[\![N]\!]^\dagger\rangle_1\cdots\rangle_{i-1}\rangle_i \\
&= \mathcal{C}_{i+1}k_{i+1}.(\lambda x.\langle k_{i+1}x\rangle_{i+1})\langle[\![N]\!]^\dagger\rangle_i \quad \text{by telescope, } \beta_\Omega\text{, reset-reset} \\
&= \langle[\![N]\!]^\dagger\rangle_i \quad \text{by reset-lift-2, } \eta_v\text{, } \mathcal{C}\text{-elim} \\
&= \langle N\rangle_i \quad \text{by induction hypothesis}
\end{aligned}$$

When $i = n$, we have:

$$\begin{aligned}
[\![\langle N\rangle_n]\!]^\dagger &= \overline{\mathcal{C}k_{1,n}}.(\lambda x.(\lambda x.x)\langle k_n\cdots\langle k_1x\rangle_1\cdots\rangle_n) \\
&\qquad \langle(\lambda x.x)\langle(\lambda x.x)\cdots(\lambda x.x)\langle[\![N]\!]^\dagger\rangle_1\cdots\rangle_{n-1}\rangle_n
\end{aligned}$$

and we can derive $[\![\langle N\rangle_n]\!]^\dagger = \langle N\rangle_n$ in the same way as the previous derivation.
(Case: $M = \mathcal{C}_ic.N$) Let $L = \lambda x\overline{k'_{1,i}}.\theta_0x\overline{k_{1,i}}$. Then L is of type `Value`, and we have

$$\begin{aligned}
L^\dagger &= \lambda x.\overline{\mathcal{C}k'_{1,i}}.k_i\langle k_{i-1}\langle\cdots\langle k_1\langle(\lambda y.y)x\rangle_1\rangle_2\cdots\rangle_{i-2}\rangle_{i-1} \\
&= \lambda x.\overline{\mathcal{C}k'_{1,i}}.(\lambda x.k_i\langle k_{i-1}\langle\cdots\langle k_1\langle x\rangle_1\rangle_2\cdots\rangle_{i-2}\rangle_{i-1})x \quad \text{by } \beta_\Omega\text{, } \beta_v
\end{aligned}$$

Then we have:

$$\begin{aligned}
[\![\mathcal{C}_i c.N]\!]^\dagger &= \overline{\mathcal{C}k_{1,i}}.(\lambda x.x)\langle(\lambda x.x)\langle\cdots\langle(\lambda x.x)([\![N]\!]\{c := L\})^\dagger\rangle_1\cdots\rangle_{i-2}\rangle_{i-1} \\
&= \overline{\mathcal{C}k_{1,i}}.([\![N]\!]\{c := L\})^\dagger \quad \text{by } \beta_\Omega\text{, reset-reset, } \mathcal{C}\text{-reset} \\
&= \overline{\mathcal{C}k_{1,i}}.[\![N]\!]^\dagger\{c := L^\dagger\} \quad \text{by Lemma 1} \\
&= \mathcal{C}_i k.[\![N]\!]^\dagger\{c := \lambda x.\mathcal{C}_i k'.kx\} \quad \text{by telescope twice} \\
&= \mathcal{C}_i k.[\![N]\!]^\dagger\{c := \lambda x.\mathcal{A}_i\langle kx\rangle_{i-1}\} \quad \text{by (*)} \\
&= \mathcal{C}_i k.[\![N]\!]^\dagger\{c := k\} \quad \text{by } \mathcal{C}\text{-abort, } \eta_v \\
&= \mathcal{C}_i c.N \quad \text{by induction hypothesis}
\end{aligned}$$

For the derivation of (*), we used $\mathcal{C}_{i\_}.M = \mathcal{A}_i\langle M\rangle_{i-1}$, which has been already derived in the derivation of telescope (in the proof of Theorem 2).

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

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

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

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 $\lambda\mathcal{T}_n \vdash [\![M]\!] = [\![N]\!]$. Since $[\![M]\!]$ and $[\![N]\!]$ are of type $\mathtt{Exp}_0$, we have $\lambda\mathcal{C}_n \vdash [\![M]\!]^\dagger = [\![N]\!]^\dagger$ by Theorem 3. Using Theorem 5, we conclude that $\lambda\mathcal{C}_n \vdash M = N$.

6.4. Completeness of $\lambda\mathcal{S}_n$

We can also obtain soundness and completeness of $\lambda\mathcal{S}_n$.

THEOREM 7 (Soundness & Completeness). *Let M and N be terms in $\lambda\mathcal{S}_n$. Then we have:*

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

Proof. Soundness can be proved in the same way as for $\lambda\mathcal{C}_n$. 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. Namely,

$$\phi(\mathcal{S}_i k.M) \;=\; \mathcal{C}_i k'.\phi(M)\{k := \lambda x.\langle k'x\rangle_i\}$$

where k' is fresh. ϕ is homomorphic on other term constructors. Similarly let ψ be a translation from $\lambda\mathcal{C}_n$ to $\lambda\mathcal{S}_n$ defined by:

$$\psi(\mathcal{C}_i k.N) \;=\; \mathcal{S}_i k'.\psi(N)\{k := \lambda x.\mathcal{S}_{i\_}.k'x\}$$

where k' is fresh. It suffices to prove the following three properties for terms M and M' in $\lambda\mathcal{S}_n$ and terms N and N' in $\lambda\mathcal{C}_n$:

1. $\lambda\mathcal{T}_n \vdash [\![\phi(M)]\!] = [\![M]\!]$.
2. $\lambda\mathcal{C}_n \vdash N = N'$ implies $\lambda\mathcal{S}_n \vdash \psi(N) = \psi(N')$.
3. $\lambda\mathcal{S}_n \vdash \psi(\phi(M)) = M$

The first property can be proved by calculation.

For the second property, we need to show that each of three specific axioms in $\lambda\mathcal{C}_n$ is mapped to derivable equations in $\lambda\mathcal{S}_n$ by ψ. We only show the case for $\mathcal{C}$-lift, which can be shown as follows (we write E' for $\psi(E^{j-1})$ and M' for $\psi(M)$):

$$\begin{aligned}\psi(\langle E^{j-1}[\mathcal{C}_j k.M]\rangle_i) &\equiv \langle E'[\mathcal{S}_j c.M'\{k := \lambda x.\mathcal{S}j\_.cx\}]\rangle_i \\ &= \langle M'\{k := \lambda x.\mathcal{S}j\_.\langle E'[x]\rangle_j\}]\rangle_i \quad \text{by } \mathcal{S}\text{-lift}\end{aligned}$$

$$\psi(\langle M\{k := \lambda x.\mathcal{A}_j\langle E^{j-1}[x]\rangle_j\}\rangle_i) \equiv \langle M'\{k := \lambda x.(\lambda x.\mathcal{S}_{j\_}.x)\langle E'[x]\rangle_j\}\rangle_i)$$

hence the remaining task is to prove $\mathcal{S}_{j\_}.\langle N\rangle_j = (\lambda x.\mathcal{S}_{j\_}.x)\langle N\rangle_j$ for any N in $\lambda\mathcal{S}_n$, which can be proved similarly as $\mathcal{C}_{j\_}.\langle N\rangle_j = (\lambda x.\mathcal{C}_{j\_}.x)\langle N\rangle_j$.

To prove the third property, we have

$$\begin{aligned}\psi(\phi(\mathcal{S}_i k.M)) &\equiv \mathcal{S}_i k'.\psi(\phi(M))\{k := \lambda x.\langle \mathcal{S}_{i\_}.k'x\rangle_i\} \\ &= \mathcal{S}_i k'.\psi(\phi(M))\{k := \lambda x.k'x\} \quad \text{by } (*) \\ &= \mathcal{S}_i k.\psi(\phi(M)) \quad \text{by } \eta_v\end{aligned}$$

For the derivation (*), we need telescope in terms of shift in $\lambda\mathcal{S}_n$, which is proved similarly as in Theorem 2 for $\lambda\mathcal{C}_n$.

This completes the proof of Theorem 7.

6.5. Type System for the Source Calculus

So far we have been studying type-free source calculi only. Introducing a type structure into source calculi is an important problem, as most modern programming languages have a built-in type system to enhance the reliability of programs. Another motivation of introducing types is that, in the presence of appropriate types, we can avoid the full η_v-equality ($\lambda x.Vx = V$ for $x \notin \mathrm{FV}(V)$), which is inconsistent with the presence of basic values such as natural numbers.

We can introduce a simply typed structure to the source calculus $\lambda\mathcal{S}_n$ with a set of basic types such as $\mathtt{Nat}$, the type for natural numbers. For control operators, the simplest choice of typing would be:

$$\frac{\Gamma \vdash M : \tau_0}{\Gamma \vdash \langle M \rangle_i : \tau_0} \qquad \frac{\Gamma, k : \sigma \to \tau_0 \vdash M : \tau_0}{\Gamma \vdash \mathcal{S}_i k.M : \sigma}$$

where τ_0 is a fixed type, and σ is an arbitrary type. We assume that all shift and reset control operators use the same type for τ_0. Then we can prove that the extended CPS translation preserves typability if we add the type information to the classes $\mathtt{Exp}_0$ and $\mathtt{Value}$ in the target calculus. All the axioms and the equations in the proof of soundness and completeness are typable, and our proof goes through for the simply typed case.

The simply type system above suffices for Filinski's implementation of shift and reset [13], but it is too restrictive. Danvy and Filinski [8] and Murthy [26] have proposed more liberal type systems for shift and reset, both of which essentially assign types induced by the CPS translation to source terms. Since the work presented in this article is orthogonal to the type systems in the source calculus, it seems straightforward to extend our axioms to the source calculi with these type systems.

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 4.4 we have $\mathtt{Exp}_0 = (\mathtt{Value} \to \mathtt{Ans}) \to \mathtt{Ans}$. If we take $n = 2$, then

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

Hence the type $\mathtt{Ans}$ 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. It seems an interesting challenge to investigate the logical contents of iterated CPS translations (or equivalently, iterated double-negation translations).

Recently Ariola, Herbelin and Sabry [1] gave a type-theoretic study on control operators for delimited continuations. Their work is conceptually close to our earlier work [21] in that their purpose is to obtain a logical view of these control operators through the type system. They mainly studied direct-style calculi independently to a CPS translation.

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 [26], 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, which can derive, for instance, a generalized version of telescope axiom.

In another line of work, Danvy and Yang [11], Murthy [26], and Biernacka, Biernacki and Danvy [5] studied an operational aspect of the control operators in the CPS hierarchy, and derived an abstract machine from the CPS translation, and also a set of reduction rules for shift and reset. Their reduction rules are essentially the same as the ones we gave as an operational semantics.

In the study of delimited continuations, there is a question about which control operators are the best among many proposed ones [12, 19, 20, 21, 1]. Even recently, this question is the target of hot discussion [31, 6]. It is difficult to answer this question if we only look at the operational semantics. However, if we take the CPS translation as our essential tool to write applications as well as studying foundations, then shift and reset (and their relatives such as $\mathcal{C}_i$ and $\mathcal{D}_i$) are the only control operators defined solely by (the iteration of) the standard CPS translation among the proposed control operators. In recent years, it has been shown that for the case of level-1 control operators for delimited continuations, shift and reset are the standard choice, for only shift and reset have found many applications in divergent areas.

For the case of higher level control operators where different delimiters (reset) exist in one program, things have not been settled, and there are many conflicting views about their necessity and expressiveness. In this article, we have shown that the higher level shift and reset can be axiomatized by a small set of concise axioms, whose sizes (as terms) are almost the same as those for level-1 shift and reset. Thus, we

believe that we can reinforce the claim that shift and reset are one of the better choices for higher level control operators too. Also the existence of several application programs and the correspondence between the CPS hierarchy and layered monads [15] seem to give a positive answer to this question. Recently Asai has studied partial evaluators using shift and reset and its correctness using our axiomatization for level-1 shift and reset [2, 3]. If the source language contains shift and reset, then one may need higher level shift and reset, and our direct-style axiomatization in this article will be needed to guarantee its correctness, since CPS translating the whole partial evaluator may result in a rather complicated program which is difficult to reason about.

Future Work: There are several possibilities for future work on the topics of this article.

First, while we have built an equational foundation for the control operators in the CPS hierarchy there remains a question about the application of our axioms. It is difficult to use them for automatic verification because they require a degree of insight. Nevertheless, we hope to use the axioms to prove the correctness of program translations. In Section 5.1, we briefly mentioned how to use the axioms reset-lift to show an equation which is useful in the let-insertion technique, but a thorough investigation on this direction is yet to be done.

Secondly, it is interesting to see which reduction rules are sound and complete with respect to the CPS translation rather than equations. This question is theoretically interesting as well as practically important in connection with operational semantics and abstract machines of shift and reset. Still it is a hard question even for the case of callcc.

Finally and most importantly, in order to answer the basic question about which higher level control operator is the best one, we need to have more substantial applications using higher level shift and reset.

Acknowledgments: The author would like to thank Olivier Danvy, Masahito Hasegawa, Kenichi Asai, Dariusz Biernacki, Malgorzata Biernacka, and Amr Sabry for constructive comments and discussions. He also thanks anonymous referees of CSL'04 and HOSC for many insightful comments and criticisms, and Noam Zeilberger for proofreading.

This work was supported in part by Japan Science and Technology Agency, and by Grant-in-Aid for Scientific Research No. 16500004 from Japan Society for the Promotion of Science.

References

1. Ariola, Z. M., H. Herbelin, and A. Sabry: 2004, 'A Type-Theoretic Foundation of Continuations and Prompts'. *Proc. ACM SIGPLAN International Conference on Functional Programming* pp. 40–53.

2. Asai, K.: 2002, 'Online Partial Evaluation for Shift and Reset'. In: *Proc. ACM Workshop on Partial Evaluation and Semantics-Based Program Manipulation*. pp. 19–30.
3. Asai, K.: 2004, 'Offline Partial Evaluation for Shift and Reset'. In: *Proc. 2004 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*. pp. 3–14.
4. Berdine, J., P. O'Hearn, U. Reddy, and H. Thielecke: 2002, 'Linear Continuation Passing'. *Higher-Order and Symbolic Computation* pp. 181–208.
5. Biernacka, M., D. Biernacki, and O. Danvy: 2004, '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*. pp. 25–34.
6. Biernacki, D. and O. Danvy: 2005, 'On the Dynamic Extent of Delimited Continuations'. Technical Report BRICS-RS-05-2, BRICS, University of Aarhus, Denmark.
7. Danvy, O.: 1996, 'Type-Directed Partial Evaluation'. In: *Proc. 23rd Symposium on Principles of Programming Languages*. pp. 242–257.
8. Danvy, O. and A. Filinski: 1989, 'A Functional Abstraction of Typed Contexts'. Technical Report 89/12, DIKU, University of Copenhagen.
9. Danvy, O. and A. Filinski: 1990, 'Abstracting Control'. In: *Proc. 1990 ACM Conference on Lisp and Functional Programming*. pp. 151–160.
10. Danvy, O. and A. Filinski: 1992, 'Representing Control: a Study of the CPS Transformation'. *Mathematical Structures in Computer Science* **2**(4), 361–391.
11. Danvy, O. and Z. Yang: 1999, 'An Operational Investigation of the CPS Hierarchy'. In: *Proc. 8th European Symposium on Programming*. pp. 224–242.
12. Felleisen, M.: 1988, 'The Theory and Practice of First-Class Prompts'. In: *Proc. 15th Symposium on Principles of Programming Languages*. pp. 180–190.
13. Filinski, A.: 1994, 'Representing Monads'. In: *Proc. 21st Symposium on Principles of Programming Languages*. pp. 446–457.
14. Filinski, A.: 1996, 'Controlling Effects'. Ph.D. thesis, Carnegie Mellon University, available as Technical Report CMU-CS-96-119.
15. Filinski, A.: 1999, 'Representing Layered Monads'. In: *Proc. 26th Symposium on Principles of Programming Languages*. pp. 175–188.
16. Flanagan, C., A. Sabry, B. F. Duba, and M. Felleisen: 1993, 'The Essence of Compiling with Continuations'. *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation* pp. 237–247.
17. Friedman, H.: 1978, 'Classically and Intuitionistically Provably Recursive Functions'. In: *Lecture Notes in Mathematics 699*. pp. 21–28.
18. Griffin, T.: 1990, 'A Formulae-as-Types Notion of Control'. In: *Proc. 17th Symposium on Principles of Programming Languages*. pp. 47–58.
19. Gunter, C. A., D. Rémy, and J. G. Riecke: 1995, 'A Generalization of Exceptions and Control in ML-Like Languages'. In: *Proc. Functional Programming and Computer Architecture*. pp. 12–23.
20. Hieb, R., R. Dybvig, and C. W. Anderson: 1993, 'Subcontinuations'. *Lisp and Symbolic Computation* **6**, 453–484.
21. Kameyama, Y.: 2000, 'A Type-Theoretic Study on Partial Continuations'. In: *Proc. IFIP International Conference on Theoretical Computer Science*. pp. 489–504.
22. Kameyama, Y.: 2004, 'Axiomatizing Higher Level Delimited Continuations'. In: *Proc. 3rd ACM SIGPLAN Workshop on Continuations, Technical Report 545, Computer Science Department, Indiana University*. pp. 49–53.

23. Kameyama, Y. and M. Hasegawa: 2003, 'A Sound and Complete Axiomatization of Delimited Continuations'. In: *Proc. ACM SIGPLAN International Conference on Functional Programming*. pp. 177–188.
24. Lawall, J. and O. Danvy: 1994, 'Continuation-Based Partial Evaluation'. In: *Proc. 1994 ACM Conference on LISP and Functional Programming*. pp. 227–238.
25. Moggi, E.: 1989, 'Computational Lambda-Calculus and Monads'. In: *Proc. 4th Symposium on Logic in Computer Science*. pp. 14–28.
26. Murthy, C.: 1992, '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*. pp. 49–72.
27. Plotkin, G. D.: 1975, 'Call-by-Name, Call-by-Value, and the λ-Calculus'. *Theoretical Computer Science* **1**, 125–159.
28. Sabry, A.: 1996, 'Note on Axiomatizing the Semantics of Control Operators'. Technical Report CIS-TR-96-03, Dept. of Computer Science, University of Oregon.
29. Sabry, A. and M. Felleisen: 1993, 'Reasoning about Programs in Continuation-Passing Style'. *Lisp and Symbolic Computation* **6**(3-4), 289–360.
30. Sekiguchi, T., T. Sakamoto, and A. Yonezawa: 2001, 'Portable Implementation of Continuation Operators in Imperative Languages by Exception Handling'. In: *Advances in Exception Handling Techniques*. pp. 217–233.
31. Shan, C.: 2004, 'Shift to Control'. *Proc. 5th Workshop on Scheme and Functional Programming* pp. 99–107.
32. Thielecke, H.: 2003, 'From Control Effects to Typed Continuation Passing'. In: *Proc. 30th Symposium on Principles of Programming Languages*. pp. 139–149.