HOOTS99 Preliminary Version

On proving syntactic properties of CPS programs

Olivier Danvy and Belmina Dzafic [1]

BRICS
Department of Computer Science, University of Aarhus
Building 540, Ny Munkegade, DK-8000 Aarhus C, Denmark.
E-mail: {danvy,belmina}@brics.dk

Frank Pfenning [2]

School of Computer Science, Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213-3891, USA.
E-mail: fp@cs.cmu.edu

Abstract

Higher-order program transformations raise new challenges for proving properties of their output, since they resist traditional, first-order proof techniques. In this work, we consider (1) the "one-pass" continuation-passing style (CPS) transformation, which is second-order, and (2) the occurrences of parameters of continuations in its output. To this end, we specify the one-pass CPS transformation relationally and we use the proof technique of logical relations.

1 Introduction

We are interested in two syntactic properties of CPS programs: the occurrences of continuation identifiers and of parameters of continuations. The first occurrence formalizes a folklore property in the continuation community that "one k is enough" [10]. The second occurrence was informally stated in connection with the direct-style transformation, which is an inverse of the CPS transformation [3]. We address the second occurrence property here.

CPS programs are typically obtained by CPS transformation, and the canonical CPS transformation is due to Plotkin, in the mid-70's [17]. It, however, gives rise to annoying "administrative reductions" that are interleaved with actual reductions. Proving properties of CPS programs such as relating

[1] Basic Research in Computer Science (http://www.brics.dk), Centre of the Danish National Research Foundation.

[2] This work is supported by NSF Grant CCR-9303383.

their reduction steps with the corresponding reduction steps in direct style thus required Plotkin to develop a so-called “colon translation” [17] which has stuck [12,19].

In the late 80’s, however, a new CPS transformation was developed that operates in one pass and performs administrative reductions at transformation time [1,6,23]. This one-pass transformation is higher-order (or more precisely: second-order), and it is not clear how to prove properties about it, which is our goal here.

This work.

We restate the one-pass CPS transformation in relational form and we present a proof technique using logical relations to prove a syntactic property of the output of the CPS transformation. This note grew out of a general study of the two syntactic properties mentioned above [5,7,9].

Overview.

The rest of this note is organized as follows. In Section 2, we present a BNF of the λ-calculus in direct style, the corresponding BNF of the λ-calculus in CPS, and two successive refinements of the CPS transformation: Plotkin’s original specification, the one-pass specification in functional form, and our one-pass specification in relational form. In Section 3, we present the syntactic property of interest, and we prove that it is satisfied by the output of the CPS transformation. Section 4 concludes.

2 Direct style, continuation-passing style, and the CPS transformation

2.1 Direct-style (DS) programs

The BNF of the pure λ-calculus reads as follows. We refer to this λ-calculus as *direct style* (DS) to distinguish it from the *continuation-passing style* (CPS) calculus introduced below.

$r \in \text{DRoot}$	— DS terms	$r ::= e$
$e \in \text{DExp}$	— DS expressions	$e ::= e_0\, e_1 \mid t$
$t \in \text{DTriv}$	— DS trivial expressions	$t ::= x \mid \lambda x.r$
$x \in \text{Ide}$	— identifiers	

The distinction between trivial expressions and (serious) expressions originates in Reynolds’s work [18].

2.2 Continuation-passing style (CPS) programs

The BNF of CPS terms reads as follows. (NB: We distinguish between the original identifiers x coming from the direct-style term, and the fresh identifiers k and v denoting continuations and the arguments of continuations.)

$$[\![e]\!]^{\mathrm{DRoot}} = [\![e]\!]^{\mathrm{DExp}}$$

$$\begin{aligned}
[\![e_0\, e_1]\!]^{\mathrm{DExp}} &= \lambda k.[\![e_0]\!]^{\mathrm{DExp}}\, \lambda v_0.[\![e_1]\!]^{\mathrm{DExp}}\, \lambda v_1.v_0\, v_1\, k \\
&\quad \text{where } k,\ v_0 \text{ and } v_1 \text{ are fresh.} \\
[\![t]\!]^{\mathrm{DExp}} &= \lambda k.k\, [\![t]\!]^{\mathrm{DTriv}} \quad \text{where } k \text{ is fresh.}
\end{aligned}$$

$$\begin{aligned}
[\![x]\!]^{\mathrm{DTriv}} &= x \\
[\![\lambda x.r]\!]^{\mathrm{DTriv}} &= \lambda x.[\![r]\!]^{\mathrm{DRoot}}
\end{aligned}$$

Fig. 1. Plotkin's left-to-right, call-by-value CPS transformation

r	$\in$ CRoot	— CPS terms	$r ::= \lambda k.e$
e	$\in$ CExp	— CPS (serious) expression	$e ::= t_0\, t_1\, c \mid c\, t$
t	$\in$ CTriv	— CPS trivial expression	$t ::= x \mid \lambda x.r \mid v$
c	$\in$ CCont	— CPS continuations	$c ::= \lambda v.e \mid k$
x	$\in$ Ide	— source identifiers	
k	$\in$ Cont	— fresh continuation identifiers	
v	$\in$ Var	— fresh parameters of continuations	

CPS terms are remarkable in that they satisfy the three properties of indifference, simulation, and translation [14,17,20]. Indifference: CPS terms are evaluation-order independent. Simulation: the CPS transformation encodes an evaluation order. Translation: there is an equational correspondence between direct-style and CPS calculi.

2.3 *The CPS transformation*

2.3.1 *From Plotkin's CPS transformation to the one-pass CPS transformation*

Plotkin's original call-by-value CPS transformation is displayed in Figure 1, where it is phrased to match the syntactic domains of Section 2.1 [17]. Using it as a first-order rewriting system, however, gives rise to the notion of *administrative redexes*: redexes solely due to the CPS transformation and not corresponding to an actual reduction step in the original program. The corresponding administrative reductions are annoying because they are interleaved with actual reductions [12,13,17,19,20]:

$$\begin{tikzcd}
\mathrm{DS} \arrow[loop left, "\text{actual reductions}"] \arrow[r, "\text{CPS transformation}"] & \mathrm{CPS} \arrow[loop right, "\text{administrative + actual reductions}"]
\end{tikzcd}$$

Let us consider the following simple example, using Figure 1.

$$[\![\lambda x.x\, x]\!]^{\mathrm{DRoot}} = \lambda k.k\, \lambda x.\lambda k.(\lambda k.k\, x)\, (\lambda v_0.(\lambda k.k\, x)\, (\lambda v_1.v_0\, v_1\, k))$$

$$
\begin{aligned}
[\![\cdot]\!]^{\mathrm{DRoot}} &: \mathrm{DRoot} \to \mathrm{CRoot} \\
[\![e]\!]^{\mathrm{DRoot}} &= \lambda k.[\![e]\!]^{\mathrm{DExp}}([t]\,k\,t) \quad \text{where } k \text{ is fresh.} \\[1em]
[\![\cdot]\!]^{\mathrm{DExp}} &: \mathrm{DExp} \to (\mathrm{CTriv} \to \mathrm{CExp}) \to \mathrm{CExp} \\
[\![e_0\,e_1]\!]^{\mathrm{DExp}} &= [\kappa]\;[\![e_0]\!]^{\mathrm{DExp}}\,[t_0]\;[\![e_1]\!]^{\mathrm{DExp}}\,[t_1]\;t_0\,t_1\,\lambda v.\kappa(v) \\
&\ \text{where } v \text{ is fresh.} \\
[\![t]\!]^{\mathrm{DExp}} &= [\kappa]\;\kappa([\![t]\!]^{\mathrm{DTriv}}) \\[1em]
[\![\cdot]\!]^{\mathrm{DTriv}} &: \mathrm{DTriv} \to \mathrm{CTriv} \\
[\![x]\!]^{\mathrm{DTriv}} &= x \\
[\![\lambda x.r]\!]^{\mathrm{DTriv}} &= \lambda x.[\![r]\!]^{\mathrm{DRoot}}
\end{aligned}
$$

Fig. 2. The one-pass CPS transformation formulated as a function

The CPS-transformed program contains two administrative redexes: the two occurrences of $(\lambda k....)....$ And reducing them yields two more administrative redexes.

It turns out, however, that administrative reductions can be factored out of a CPS program, so that the resulting reductions are actual ones:

$$\begin{tikzcd}
\mathrm{DS} \arrow[loop left, "\text{actual reductions}"] \arrow[r, "\text{CPS transformation}"] & \mathrm{CPS} \arrow[r, "\text{administrative reductions}"] & \mathrm{CPS} \arrow[loop right, "\text{actual reductions}"]
\end{tikzcd}$$

Sabry and Felleisen have documented such an approach [2,19–21].

Furthermore, it turns out that CPS transformation and administrative reductions can be integrated into one, higher-order, rewriting system that directly produces a CPS program without administrative redexes, in one pass [1,6,8,23]:

$$\begin{tikzcd}
\mathrm{DS} \arrow[loop left, "\text{actual reductions}"] \arrow[r, "\text{CPS transformation}", "\substack{+\\ \text{administrative}\\ \text{reductions}}"'] & \mathrm{CPS} \arrow[loop right, "\text{actual reductions}"]
\end{tikzcd}$$

Let us revisit the simple example above, using Figure 2.

$$[\![\lambda x.x\,x]\!]^{\mathrm{DRoot}} = \lambda k.k\,\lambda x.\lambda k.x\,x\,k$$

We consider this higher-order CPS transformation here, as displayed in Figure 2, where it is phrased to match the syntactic domains of Sections 2.1 and 2.2. The one-pass CPS transformation requires meta-level abstractions, written as $[t]\,e$, and the corresponding applications, written as $\kappa(t)$, where κ

$$\frac{\vdash e \,;\, [t]\, k\, t \overset{\text{DExp}}{\longrightarrow} e'}{\vdash e \overset{\text{DRoot}}{\longrightarrow} \lambda k.e'} \qquad \frac{\vdash t \overset{\text{DTriv}}{\longrightarrow} t'}{\vdash t \,;\, \kappa \overset{\text{DExp}}{\longrightarrow} \kappa(t')}$$

$$\frac{\vdash e_1 \,;\, [t_1]\, t_0\, t_1\, \lambda v.\kappa(v) \overset{\text{DExp}}{\longrightarrow} e_1' \qquad \vdash e_0 \,;\, [t_0]\, e_1' \overset{\text{DExp}}{\longrightarrow} e'}{\vdash e_0\, e_1 \,;\, \kappa \overset{\text{DExp}}{\longrightarrow} e'}$$

$$\frac{}{\vdash x \overset{\text{DTriv}}{\longrightarrow} x} \qquad \frac{\vdash r \overset{\text{DRoot}}{\longrightarrow} r'}{\vdash \lambda x.r \overset{\text{DTriv}}{\longrightarrow} \lambda x.r'}$$

Fig. 3. The one-pass CPS transformation formulated as a judgment

ranges over meta-level functions from trivial CPS expressions to CPS expressions. The key type reads

$$[\![\cdot]\!]^{\text{DExp}} : \text{DExp} \to (\text{CTriv} \to \text{CExp}) \to \text{CExp}.$$

2.4 The one-pass CPS transformation in relational form

For the purpose of our work here, Figure 3 re-expresses the one-pass CPS transformation of Figure 2 in relational form. It uses three judgments. A direct-style term r is transformed into a CPS term r' whenever the judgment

$$\vdash r \overset{\text{DRoot}}{\longrightarrow} r'$$

is satisfied. Given a (higher-order) accumulator κ, a direct-style expression e is transformed into a CPS expression e' whenever the judgment

$$\vdash e \,;\, \kappa \overset{\text{DExp}}{\longrightarrow} e'$$

is satisfied. Finally, a direct-style trivial expression t is transformed into a CPS trivial expression t' whenever the judgment

$$\vdash t \overset{\text{DTriv}}{\longrightarrow} t'$$

is satisfied.

These judgments can be interpreted operationally by assuming that r, e and κ, or t are given and r', e', and t' are to be constructed by building a derivation in a bottom-up fashion. The meta-level applications arising in two of the rules are reduced administratively.

NB: In the inference rule for applications, t_0 is "new", i.e., the deduction of the left premise is parametric in t_0. This parameter may, however, occur free in e_1', which means that we can substitute an arbitrary trivial term t for t_0 in this derivation and obtain a derivation of $\vdash e_1 \,;\, [t_1]\, t\, t_1\, \lambda v.\kappa(v) \overset{\text{DExp}}{\longrightarrow} e_1'[t/t_0]$. This property is exploited crucially in the proof of Section 3.

$$\frac{\bullet \models^{\text{CExp}}_{\text{Var}} e}{\models^{\text{CRoot}}_{\text{Var}} \lambda k.e} \qquad \frac{\Xi \models^{\text{CTriv}}_{\text{Var}} t \;;\; \Xi' \qquad \Xi' \models^{\text{CCont}}_{\text{Var}} c}{\Xi \models^{\text{CExp}}_{\text{Var}} c\,t}$$

$$\frac{\Xi \models^{\text{CTriv}}_{\text{Var}} t_1 \;;\; \Xi_1 \qquad \Xi_1 \models^{\text{CTriv}}_{\text{Var}} t_0 \;;\; \Xi_0 \qquad \Xi_0 \models^{\text{CCont}}_{\text{Var}} c}{\Xi \models^{\text{CExp}}_{\text{Var}} t_0\,t_1\,c}$$

$$\frac{}{\Xi \models^{\text{CTriv}}_{\text{Var}} x \;;\; \Xi} \qquad \frac{\models^{\text{CRoot}}_{\text{Var}} r}{\Xi \models^{\text{CTriv}}_{\text{Var}} \lambda x.r \;;\; \Xi} \qquad \frac{}{\Xi, v \models^{\text{CTriv}}_{\text{Var}} v \;;\; \Xi}$$

$$\frac{}{\bullet \models^{\text{CCont}}_{\text{Var}} k} \qquad \frac{\Xi, v \models^{\text{CExp}}_{\text{Var}} e}{\Xi \models^{\text{CCont}}_{\text{Var}} \lambda v.e}$$

Fig. 4. Valid occurrences of parameters of continuations in a CPS term

2.5 Summary and conclusion

We have specified (1) the input language of Plotkin's left-to-right, call-by-value CPS transformation, (2) the corresponding output language, which incidentally is closed under β-reduction, and (3) a one-pass version of Plotkin's CPS transformation in relational form.

3 A syntactic property of CPS programs

The CPS transformation introduces two classes of fresh identifiers: the continuation identifiers k and the parameters of continuations v. We consider the occurrences of v's here.

Figure 4 characterizes the occurrence conditions on the formal parameters of continuations v, which occur in a stack-like fashion [3]. Here we use Ξ to range over stacks of continuation parameters defined below, where $\bullet$ denotes the empty stack.

$$\Xi \;::=\; \bullet \mid \Xi, v$$

Figure 4 should be read as follows. Given a CPS term $\lambda k.e$, the judgment

$$\models^{\text{CRoot}}_{\text{Var}} \lambda k.e$$

is satisfied whenever the parameters of continuations declared in e occur properly in e. Given a CPS expression e occurring in the scope of parameters of continuations properly listed in Ξ, the judgment

$$\Xi \models^{\text{CExp}}_{\text{Var}} e$$

is satisfied whenever the variables in Ξ and all the other parameters of continuations declared in e occur properly in e. Similarly, given a trivial term t

occurring in the scope of parameters of continuations properly listed in Ξ, the judgment

$$\Xi \models^{\text{CTriv}}_{\text{Var}} t \; ; \; \Xi'$$

is satisfied whenever Ξ' is a prefix of Ξ and the remaining variables of Ξ occur properly in t. And finally, given a continuation c occurring in the scope of parameters of continuations properly listed in a list Ξ, the judgment

$$\Xi \models^{\text{CCont}}_{\text{Var}} c$$

is satisfied whenever all the parameters of continuations declared in c and the variables listed in Ξ occur properly in c.

This occurrence condition essentially says that formal parameters of continuations are introduced and used in a stack-like manner.

Let us prove that CPS-transforming a direct-style term r yields a CPS term r' whose continuation identifiers satisfy the occurrence conditions of Figure 4. In other words, we would like to show that

$$\text{if } \; \vdash r \overset{\text{DRoot}}{\longrightarrow} r' \; \text{ then } \; \models^{\text{CRoot}}_{\text{Var}} r'.$$

Clearly, we cannot prove this inductively by itself since properties at the root of a term are defined in terms of the expressions it contains. The critical issue is the property of the higher-order accumulators κ we must prove (in the inductive conclusion) and require (in the inductive hypothesis) for the translation of expressions in Figure 3. In the CPS transformation, a higher-order accumulator is a (meta-level) function from trivial terms to expressions, which suggests the method of logical relations [22]. The idea behind binary logical relations is to consider two functions related if they map related arguments to related results. In unary form: a function is valid if it maps valid arguments to valid results. This kind of definition is pervasive in the application of logical frameworks to meta-theoretic reasoning (e.g., [9,15]). It works smoothly here.

Four notions of validity arise: for root terms, for serious expressions, for trivial expressions, and for accumulators. In their definitions, we must account for the context Ξ in which an expression might occur. For root terms, serious expressions, and trivial expressions, the notion of validity is derived directly from the property we are trying to prove; for accumulators it arises from the considerations of logical relations as motivated above. We also streamline the definitions by considering separately the case of a trivial variable v, since such a variable is never the result of the translation of a trivial direct-style term (see Theorem 3.2 (3)).

Definition 3.1

(1) r' *is* Var-valid *if* $\models^{CRoot}_{Var} r'$.

(2) e' *is* Ξ-Var-valid *if* $\Xi \models^{CExp}_{Var} e'$.

(3) t' *is* Var-valid *if* $\Xi \models^{CTriv}_{Var} t' \; ; \; \Xi$ *for every* Ξ.

(4) κ *is* Ξ-Var-valid *if*

(a) $\Xi, v \models^{CExp}_{Var} \kappa(v)$, *and*

(b) $\Xi \models^{\mathit{CExp}}_{\mathit{Var}} \kappa(t')$*, for any Var-valid* t'*.*

(5) c *is* Ξ*-Var-valid if* $\Xi \models^{\mathit{CCont}}_{\mathit{Var}} c$*.*

This definition is more complex than it may appear at first, since it involves meta-level applications $\kappa(v)$ and $\kappa(t')$ and therefore, implicitly, substitution.

Theorem 3.2

(1) If $\vdash r \overset{\mathit{DRoot}}{\longrightarrow} r'$ *then* r' *is Var-valid.*

(2) If κ *is* Ξ*-Var-valid and* $\vdash e \,;\, \kappa \overset{\mathit{DExp}}{\longrightarrow} e'$ *then* e' *is* Ξ*-Var-valid.*

(3) If $\vdash t \overset{\mathit{DTriv}}{\longrightarrow} t'$ *then* t' *is Var-valid.*

Proof. By mutual induction on the derivations $\mathcal{R}$, $\mathcal{E}$, and $\mathcal{T}$ of $\vdash r \overset{\mathrm{DRoot}}{\longrightarrow} r'$, $\vdash e \,;\, \kappa \overset{\mathrm{DExp}}{\longrightarrow} e'$, and $\vdash t \overset{\mathrm{DTriv}}{\longrightarrow} t'$, respectively.

In a slight abuse of notation, we write $e(t_0)$ and $\mathcal{E}(t_0)$ to indicate the dependence of e or $\mathcal{E}$ on a parameter t_0 and $e(t)$ and $\mathcal{E}(t)$ for the result of substituting t for t_0 in e and $\mathcal{E}$, respectively.

Case $\mathcal{R} = \dfrac{\begin{array}{c}\mathcal{E}\\ \vdash e \,;\, [t]\, k\, t \overset{\mathrm{DExp}}{\longrightarrow} e'\end{array}}{\vdash e \overset{\mathrm{DRoot}}{\longrightarrow} \lambda k.e'}$

Then $\kappa = [t]\, k\, t$ is $\bullet$-Var-valid:

(a) $\dfrac{\dfrac{}{\bullet,\, v \models^{\mathrm{CTriv}}_{\mathrm{Var}} v \,;\, \bullet}}{\bullet,\, v \models^{\mathrm{CExp}}_{\mathrm{Var}} k\, v}$ holds, and

(b) $\dfrac{\bullet \models^{\mathrm{CTriv}}_{\mathrm{Var}} t' \,;\, \bullet}{\bullet \models^{\mathrm{CExp}}_{\mathrm{Var}} k\, t'}$ for any Var-valid t'.

Hence, by induction hypothesis (2) on $\mathcal{E}$, $\bullet \models^{\mathrm{CExp}}_{\mathrm{Var}} e'$, and thus $\models^{\mathrm{CRoot}}_{\mathrm{Var}} \lambda k.e'$.

Case $\mathcal{E} = \dfrac{\begin{array}{c}\mathcal{E}_1(t_0)\\ \vdash e_1 \,;\, [t_1]\, t_0\, t_1\, \lambda v.\kappa(v) \overset{\mathrm{DExp}}{\longrightarrow} e_1'(t_0)\end{array} \qquad \begin{array}{c}\mathcal{E}_0\\ \vdash e_0 \,;\, [t_0]\, e_1'(t_0) \overset{\mathrm{DExp}}{\longrightarrow} e'\end{array}}{\vdash e_0\, e_1 \,;\, \kappa \overset{\mathrm{DExp}}{\longrightarrow} e'}$

Assume κ is Ξ-Var-valid. We need to show that $\kappa_0 = [t_0]\, e_1'(t_0)$ is Ξ-Var-valid, since then $\Xi \models^{\mathrm{CExp}}_{\mathrm{Var}} e'$ by induction hypothesis (2) on $\mathcal{E}_0$. Thus we need to show Properties (a) and (b) of Definition 3.1(4) for κ_0.

(a) We need $\Xi,\, v_0 \models^{\mathrm{CExp}}_{\mathrm{Var}} \kappa_0(v_0)$. Consider

$$\begin{array}{c}\mathcal{E}_1(v_0)\\ \vdash e_1 \,;\, [t_1]\, v_0\, t_1\, \lambda v.\kappa(v) \overset{\mathrm{DExp}}{\longrightarrow} e_1'(v_0)\end{array}$$

We would like to show that

$$\kappa_1 = [t_1]\, v_0\, t_1\, \lambda v.\kappa(v)$$

is $\Xi,\, v_0$-Var-valid, since then $e_1'(v_0) = \kappa_0(v_0)$ is $\Xi,\, v_0$-Var-valid by induction hypothesis (2) on $\mathcal{E}_1(v_0)$. Therefore we need to consider the two cases of Definition 3.1(4).

(a) $\Xi,\, v_0,\, v_1 \models^{\text{CExp}}_{\text{Var}} \kappa_1(v_1)$. We derive this as follows:

$$\dfrac{\dfrac{}{\Xi,\, v_0,\, v_1 \models^{\text{CTriv}}_{\text{Var}} v_1 \;;\; \Xi,\, v_0} \qquad \dfrac{}{\Xi,\, v_0 \models^{\text{CTriv}}_{\text{Var}} v_0 \;;\; \Xi} \qquad \dfrac{\begin{array}{c}\text{since } \kappa \text{ is } \Xi\text{-Var-valid}\\ \Xi,\, v \models^{\text{CExp}}_{\text{Var}} \kappa(v)\end{array}}{\Xi \models^{\text{CCont}}_{\text{Var}} \lambda v.\kappa(v)}}{\Xi,\, v_0,\, v_1 \models^{\text{CExp}}_{\text{Var}} v_0\, v_1\, \lambda v.\kappa(v)}$$

(b) $\Xi,\, v_0 \models^{\text{CExp}}_{\text{Var}} \kappa_1(t'_1)$, where t'_1 is Var-valid. This is established by the derivation

$$\dfrac{\begin{array}{c}\text{since } t'_1 \text{ is Var-valid}\\ \Xi,\, v_0 \models^{\text{CTriv}}_{\text{Var}} t'_1 \;;\; \Xi,\, v_0\end{array} \qquad \dfrac{}{\Xi,\, v_0 \models^{\text{CTriv}}_{\text{Var}} v_0 \;;\; \Xi} \qquad \dfrac{\begin{array}{c}\text{since } \kappa \text{ is } \Xi\text{-Var-valid}\\ \Xi,\, v \models^{\text{CExp}}_{\text{Var}} \kappa(v)\end{array}}{\Xi \models^{\text{CCont}}_{\text{Var}} \lambda v.\kappa(v)}}{\Xi,\, v_0 \models^{\text{CExp}}_{\text{Var}} v_0\, t'_1\, \lambda v.\kappa(v)}$$

Thus κ_1 is $\Xi,\, v_0$-Var-valid. Therefore, by induction hypothesis on $\mathcal{E}_1(v_0)$,

$$\Xi,\, v_0 \models^{\text{CExp}}_{\text{Var}} \kappa_0(v_0).$$

(b) We need $\Xi \models^{\text{CExp}}_{\text{Var}} \kappa_0(t'_0)$ for any Var-valid t'_0. Consider

$$\begin{array}{c}\mathcal{E}_1(t'_0)\\ \vdash e_1 \;;\; [t_1]\, t'_0\, t_1\, \lambda v.\kappa(v) \overset{\text{DExp}}{\longrightarrow} \underbrace{e'_1(t'_0)}_{=\,\kappa_0(t'_0)}\end{array}$$

We would like to show that

$$\kappa_1 = [t_1]\, t'_0\, t_1\, \lambda v.\kappa(v)$$

is Ξ-Var-valid, so we can apply the induction hypothesis to $\mathcal{E}_1(t'_0)$. Again, we need to consider the two clauses of Definition 3.1(4).

(a) $\Xi,\, v_1 \models^{\text{CExp}}_{\text{Var}} \kappa_1(v_1)$. We derive this as follows:

$$\dfrac{\dfrac{}{\Xi,\, v_1 \models^{\text{CTriv}}_{\text{Var}} v_1 \;;\; \Xi} \qquad \begin{array}{c}\text{since } t'_0 \text{ is Var-valid}\\ \Xi \models^{\text{CTriv}}_{\text{Var}} t'_0 \;;\; \Xi\end{array} \qquad \dfrac{\begin{array}{c}\text{since } \kappa \text{ is } \Xi\text{-Var-valid}\\ \Xi,\, v \models^{\text{CExp}}_{\text{Var}} \kappa(v)\end{array}}{\Xi \models^{\text{CCont}}_{\text{Var}} \lambda v.\kappa(v)}}{\Xi,\, v_1 \models^{\text{CExp}}_{\text{Var}} t'_0\, v_1\, \lambda v.\kappa(v)}$$

(b) $\Xi \models^{\text{CExp}}_{\text{Var}} \kappa_1(t'_1)$ for any Var-valid t'_1. We construct:

$$\dfrac{\begin{array}{c}\text{since } t'_1 \text{ is Var-valid}\\ \Xi \models^{\text{CTriv}}_{\text{Var}} t'_1 \;;\; \Xi\end{array} \qquad \begin{array}{c}\text{since } t'_0 \text{ is Var-valid}\\ \Xi \models^{\text{CTriv}}_{\text{Var}} t'_0 \;;\; \Xi\end{array} \qquad \dfrac{\begin{array}{c}\text{since } \kappa \text{ is } \Xi\text{-Var-valid}\\ \Xi,\, v \models^{\text{CExp}}_{\text{Var}} \kappa(v)\end{array}}{\Xi \models^{\text{CCont}}_{\text{Var}} \lambda v.\kappa(v)}}{\Xi \models^{\text{CExp}}_{\text{Var}} t'_0\, t'_1\, \lambda v.\kappa(v)}$$

Hence κ_1 is Ξ-Var-valid and thus $\Xi \models^{\text{CExp}}_{\text{Var}} \underbrace{e'_1(t'_0)}_{=\,\kappa_0(t'_0)}$ by induction hypothesis (2) on $\mathcal{E}_1(t'_0)$.

Thus κ_0 is Ξ-Var-valid. Hence e' is Ξ-Var-valid by induction hypothesis (2) on $\mathcal{E}_0$.

Case $\mathcal{E} = \dfrac{\overset{\mathcal{T}}{\vdash t \overset{\text{DTriv}}{\longrightarrow} t'}}{\vdash t\ ;\ \kappa \overset{\text{DExp}}{\longrightarrow} \kappa(t')}$.

By induction hypothesis (3) on $\mathcal{T}$, t' is Var-valid. Since we assume that κ is Ξ-Var-valid, $\kappa(t')$ is also Ξ-Var-valid by clause 4b in Definition 3.1.

Case $\mathcal{T} = \dfrac{}{\vdash x \overset{\text{DTriv}}{\longrightarrow} x}$.

$\dfrac{}{\Xi \models^{\text{CTriv}}_{\text{Var}} x\ ;\ \Xi}$ is an axiom for any Ξ.

Case $\mathcal{T} = \dfrac{\overset{\mathcal{R}}{\vdash r \overset{\text{DRoot}}{\longrightarrow} r'}}{\vdash \lambda x.r \overset{\text{DTriv}}{\longrightarrow} \lambda x.r'}$.

We construct $\dfrac{\overset{\text{by i.h. (1) on } \mathcal{R}}{\models^{\text{CRoot}}_{\text{Var}} r'}}{\Xi \models^{\text{CTriv}}_{\text{Var}} \lambda x.r'\ ;\ \Xi}$.

□

4 Conclusion

We have characterized an occurrence condition in CPS programs, Var-validity, and we have proven that this condition holds for the output of the one-pass CPS transformation. To this end, we developed a third-order proof technique matching the second-order nature of the one-pass CPS transformation.

Elsewhere [5,9], we investigate another, similar, occurrence condition on continuation identifiers. Using the same technique, we prove that the one-pass CPS transformation yields terms that satisfy this other occurrence condition. We then consider the closure of both occurrence conditions under β-reduction and their application to the direct-style transformation and to stack-based abstract machines for CPS programs.

We have also formalized most of the languages, transformations, properties, and proofs in Elf, a constraint logic-programming language based on the logical framework LF [9,11,16]. This formalization is small but non-trivial. It captures the computational content of the translations and the meta-theoretic reasoning in a declarative, yet executable way. Because Elf is built around the notions of substitution and meta-level function, the formalization is direct and (we find) elegant. It is also unusual in that since it abstracts over continuations, it requires third-order constants for the CPS transformation. This exemplifies a new technique for representing deductive systems in LF, which is interesting in its own right.

We can summarize this new technique as follows: we translate a two-level functional presentation to a relational representation in a logical framework by mapping "static" abstractions and applications directly to meta-level abstractions and applications. This means that static redexes of a two-level functional representation become β-redexes in the logical framework. Statically convertible terms are therefore *definitionally equal*, avoiding explicit treatment of administrative reductions. A direct encoding of the meta-theory of such a representation will be third-order, since we reason about second-order objects.

Examples of two-level functional presentations include all one-pass CPS transformations, state-passing transformations, etc., and more generally the one-pass transformation into monadic style [13]. Type-directed partial evaluation provides another example of two-level functional presentations [4,9].

Finally, and most significantly, the encoding suggested the proof technique. This work thus demonstrates, on a small scale, the value of a logical framework as a conceptual tool in the theoretical study of programming languages.

References

[1] Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, New York, 1992.

[2] Gilles Barthe, John Hatcliff, and Morten Heine Sørensen. Reflections on reflections. In Hugh Glaser, H. Hartel, and Herbert Kuchen, editors, *Ninth International Symposium on Programming Language Implementation and Logic Programming*, number 1292 in Lecture Notes in Computer Science, pages 241–258, Southampton, UK, September 1997. Springer-Verlag.

[3] Olivier Danvy. Back to direct style. *Science of Computer Programming*, 22(3):183–195, 1994.

[4] Olivier Danvy. Type-directed partial evaluation. Lecture Notes BRICS LN-98-3, Department of Computer Science, University of Aarhus, Aarhus, Denmark, December 1998. Extended version.

[5] Olivier Danvy, Belmina Dzafic, and Frank Pfenning. On the occurrence of continuation parameters in CPS programs. Unpublished note, June 1999.

[6] Olivier Danvy and Andrzej Filinski. Representing control, a study of the CPS transformation. *Mathematical Structures in Computer Science*, 2(4):361–391, December 1992.

[7] Olivier Danvy and Frank Pfenning. The occurrence of continuation parameters in CPS terms. Technical report CMU-CS-95-121, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, February 1995.

[8] Olivier Danvy and Kristoffer Høgsbro Rose. Higher-order rewriting and partial evaluation. In Tobias Nipkow, editor, *Rewriting Techniques and Applications*, Lecture Notes in Computer Science, Kyoto, Japan, March 1998. Springer-Verlag. Extended version available as the technical report BRICS-RS-97-46.

[9] Belmina Dzafic. Formalizing program transformations. Master's thesis, DAIMI, Department of Computer Science, University of Aarhus, Aarhus, Denmark, December 1998.

[10] Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. The essence of compiling with continuations. In David W. Wall, editor, *Proceedings of the ACM SIGPLAN'93 Conference on Programming Languages Design and Implementation*, SIGPLAN Notices, Vol. 28, No 6, pages 237–247, Albuquerque, New Mexico, June 1993. ACM Press.

[11] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, 1993. A preliminary version appeared in the proceedings of the First IEEE Symposium on Logic in Computer Science, pages 194–204, June 1987.

[12] John Hatcliff. *The Structure of Continuation-Passing Styles.* PhD thesis, Department of Computing and Information Sciences, Kansas State University, Manhattan, Kansas, June 1994.

[13] John Hatcliff and Olivier Danvy. A generic account of continuation-passing styles. In Hans-J. Boehm, editor, *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Programming Languages*, pages 458–471, Portland, Oregon, January 1994. ACM Press.

[14] John Hatcliff and Olivier Danvy. Thunks and the λ-calculus. *Journal of Functional Programming*, 7(2):303–319, 1997.

[15] Frank Pfenning. A proof of the Church-Rosser theorem and its representation in a logical framework. *Journal of Automated Reasoning.* To appear. A preliminary version is available as Carnegie Mellon Technical Report CMU-CS-92-186, September 1992.

[16] Frank Pfenning. Logic programming in the LF logical framework. In Gérard Huet and Gordon Plotkin, editors, *Logical Frameworks*, pages 149–181. Cambridge University Press, 1991.

[17] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[18] John C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998. Reprinted from the proceedings of the 25th ACM National Conference (1972).

[19] Amr Sabry. *The Formal Relationship between Direct and Continuation-Passing Style Optimizing Compilers: A Synthesis of Two Paradigms.* PhD thesis, Computer Science Department, Rice University, Houston, Texas, August 1994. Technical report TR94-242.

[20] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6(3/4):289–360, December 1993.

[21] Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Transactions on Programming Languages and Systems*, 19(6):916–941, November 1997.

[22] W. W. Tait. Intensional interpretation of functionals of finite type I. *Journal of Symbolic Logic*, 32:198–212, 1967.

[23] Mitchell Wand. Correctness of procedure representations in higher-order assembly language. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Proceedings of the 7th International Conference on Mathematical Foundations of Programming Semantics*, number 598 in Lecture Notes in Computer Science, pages 294–311, Pittsburgh, Pennsylvania, March 1991. Springer-Verlag.