BRICS RS-96-14 Danvy & Malmkjær: On the Idempotence of the CPS Transformation

Matematisk Institut
Aarhus Universitet
Trykkeriet

Basic Research in Computer Science

On the Idempotence of the CPS Transformation

Olivier Danvy
Karoline Malmkjær

BRICS Report Series RS-96-14

ISSN 0909-0878 May 1996

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through World Wide Web and anonymous FTP:

http://www.brics.dk/
ftp://ftp.brics.dk/
This document in subdirectory RS/96/14/

On the Syntactic Idempotence of the CPS Transformation

Olivier Danvy and Karoline Malmkjær
BRICS *
Department of Computer Science
Aarhus University †

May 10, 1996 (Revised: August 1996 and December 1996)

Abstract

The CPS (continuation-passing style) transformation on λ-terms has an interpretation in many areas of Computer Science, such as programming languages and type theory. Programming intuition suggests that in effect, it is idempotent, but this does not obviously hold for all existing CPS transformations (Plotkin, Reynolds, Fischer, *etc.*).

We rephrase the call-by-value CPS transformation to make it syntactically idempotent, modulo η-reduction of the newly introduced continuation. Type-wise, iterating this transformation corresponds to refining the polymorphic domain of answers.

*Basic Research in Computer Science,
Centre of the Danish National Research Foundation.
†Ny Munkegade, Building 540, DK-8000 Aarhus C, Denmark.
E-mail: {danvy,karoline}@brics.dk
Phone: (+45) 89 42 33 69.
Fax: (+45) 89 42 32 55.
Home pages: http://www.brics.dk/~{danvy,karoline}

1 Introduction

The CPS transformation stands at an intersection between several areas of Computer Science: on λ-terms, it is used to encode evaluation orders [11, 16, 20, 21], *e.g.*, for compiling programs [1, 17, 24]; on types, seen as propositions, it corresponds to a double-negation translation, *i.e.*, as a mapping from Classical Logic into Intuitionistic Logic [14, 19, 27].

We are interested in the effect of iterating the CPS transformation. In the absence of control operators [2, 4, 9], programming intuition suggests that nothing should be gained: since continuations represent new evaluation contexts, and CPS terms only contain tail-calls, CPS-transforming a CPS term cannot possibly introduce much of a continuation. Therefore we wonder whether the CPS transformation is, or could be made, syntactically idempotent: what is the result of CPS transforming a CPS-transformed program? Does it give the same CPS-transformed program?

One property of the λ-calculus hints that the CPS transformation could actually be made syntactically idempotent: η-reduction. The intuition is that a tail-recursive CPS term e ought to be CPS transformed into a term $\lambda k.e\,@\,k$, where "@" denotes infix application, and which can be safely η-reduced to e in a CPS context.

This intuition, however, does not hold for any known CPS transformation, and after a brush up on CPS in Section 2, we see how and why in Section 3. The problem with the CPS transformation is that it introduces too many continuations — a drawback for compilers that CPS-transform their input, when this input is already in CPS. In Section 4, we see how a strategic mix of currying and uncurrying leads us towards idempotence, which we obtain in Section 5.

In Section 6, we investigate the type analogue of syntactic idempotence. In Section 7, we consider other continuation-passing styles. Section 8 concludes.

2 CPS: Values First vs. Continuations First

Broadly speaking, there are two classes of continuation-passing styles: those that put values first ($\lambda x.\lambda k....$), and those that put continuations first ($\lambda k.\lambda x....$). The "values first" class was investigated by Plotkin, Reynolds, and others [20, 21]. The "continuations first" class was investigated by Fischer, Fradet and Le Métayer, Sabry and Felleisen, and others [11, 12, 23].

Denotational Semantics can put some light on these two classes. Suppose that continuations are defined as mappings from values to answers. Then putting values first amounts to defining (call-by-value) functions as mappings from values to continuations to answers. Conversely, putting continuations first amounts to considering functions as continuation transformers [13, 25].

Values first	Continuations first
$\mathrm{Cont} = \mathrm{Val} \to \mathrm{Ans}$	$\mathrm{Cont} = \mathrm{Val} \to \mathrm{Ans}$
$\mathrm{Fun} = \mathrm{Val} \to \mathrm{Cont} \to \mathrm{Ans}$	$\mathrm{Fun} = \mathrm{Cont} \to \mathrm{Val} \to \mathrm{Ans}$
	$= \mathrm{Cont} \to \mathrm{Cont}$

Since continuations can only be η-reduced when they are placed last, in the rest of this paper, we consider the "values first" class. Specifically, we focus on Plotkin's CPS transformation for call-by-value, where subterms are evaluated from left to right [20].

Figure 1 specifies our source language: it is the pure λ-calculus. Following Reynolds [21], we distinguish between trivial terms, whose reduction always terminate, and serious terms, whose reduction may diverge. Figures 2 and 3 display Plotkin's CPS transformation and the syntactic characterization of its output, after administrative reductions [5, 20, 23, 24]. Both BNF's were used in earlier work on the direct-style transformation, the inverse of the CPS transformation [3, 8].

3 Idempotence: No

Let us CPS-transform the identity function $\lambda x.x$ in an empty context, for simplicity. The result reads as follows.

$$\lambda x.\lambda k_1.k_1\,@\,x$$

CPS-transforming this result reads as follows.

$$\lambda x.\lambda k_2.k_2\,@\,(\lambda k_1.\lambda k_2.(k_1\,@\,x)\,@\,k_2)$$

We may η-reduce the inner occurrence of k_2, but the outer one remains, defeating idempotence.

The nature of this failing suggests to consider an uncurried CPS transformation — *i.e.*, a CPS transformation generating uncurried λ-terms (Figures

$$
\begin{array}{rll}
r \in \mathrm{DRoot} & & \text{— domain of DS } \lambda\text{-terms} \\
e \in \mathrm{DExp} & & \text{— domain of DS serious expressions} \\
t \in \mathrm{DTriv} & & \text{— domain of DS trivial expressions} \\
i \in \mathrm{Ide} & & \text{— domain of identifiers}
\end{array}
$$

$$
\begin{array}{l}
r ::= e \\
e ::= t \mid e_0 \mathbin{@} e_1 \\
t ::= i \mid \lambda i.r
\end{array}
$$

Figure 1: Syntax of pure direct-style (DS) λ-terms

$$
\begin{array}{rcl}
\mathcal{C}_c^{\mathrm{DRoot}} & : & \mathrm{DRoot} \to \mathrm{CRoot}_c \\
\mathcal{C}_c^{\mathrm{DRoot}}\,[\![e]\!] & = & \mathcal{C}_c^{\mathrm{DExp}}\,[\![e]\!] \\
\\
\mathcal{C}_c^{\mathrm{DExp}}\,[\![t]\!] & = & \lambda\kappa.\kappa \mathbin{@} \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t]\!] \\
\mathcal{C}_c^{\mathrm{DExp}}\,[\![e_0 \mathbin{@} e_1]\!] & = & \lambda\kappa.\mathcal{C}_c^{\mathrm{DExp}}\,[\![e_0]\!] \mathbin{@} \lambda t_0.\mathcal{C}_c^{\mathrm{DExp}}\,[\![e_1]\!] \mathbin{@} \lambda t_1.(t_0 \mathbin{@} t_1) \mathbin{@} \kappa \\
\\
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![i]\!] & = & i \\
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![\lambda i.r]\!] & = & \lambda i.\mathcal{C}_c^{\mathrm{DRoot}}\,[\![r]\!]
\end{array}
$$

Figure 2: Curried CPS transformation on pure DS terms

$$
\begin{array}{rll}
r \in \mathrm{CRoot}_c & & \text{— domain of curried CPS } \lambda\text{-terms} \\
e \in \mathrm{CExp}_c & & \text{— domain of curried CPS serious expressions} \\
t \in \mathrm{CTriv}_c & & \text{— domain of curried CPS trivial expressions} \\
i, k, v \in \mathrm{Ide} & & \text{— domain of identifiers}
\end{array}
$$

$$
\begin{array}{l}
r ::= \lambda k.e \\
e ::= k \mathbin{@} t \mid (t_0 \mathbin{@} t_1) \mathbin{@} \lambda v.e \\
t ::= i \mid \lambda i.r \mid v
\end{array}
$$

Figure 3: Syntax of curried CPS λ-terms

$$
\begin{array}{rcl}
\mathcal{C}_u^{\mathrm{DRoot}} & : & \mathrm{DRoot} \to \mathrm{CRoot}_u \\
\mathcal{C}_u^{\mathrm{DRoot}}\,[\![e]\!] & = & \mathcal{C}_u^{\mathrm{DExp}}\,[\![e]\!] \\
\\
\mathcal{C}_u^{\mathrm{DExp}}\,[\![t]\!] & = & \lambda\kappa.\kappa \mathbin{@} \mathcal{C}_u^{\mathrm{DTriv}}\,[\![t]\!] \\
\mathcal{C}_u^{\mathrm{DExp}}\,[\![e_0 \mathbin{@} e_1]\!] & = & \lambda\kappa.\mathcal{C}_u^{\mathrm{DExp}}\,[\![e_0]\!] \mathbin{@} \lambda t_0.\mathcal{C}_u^{\mathrm{DExp}}\,[\![e_1]\!] \mathbin{@} \lambda t_1.t_0 \mathbin{@} \langle t_1, \kappa \rangle \\
\\
\mathcal{C}_u^{\mathrm{DTriv}}\,[\![i]\!] & = & i \\
\mathcal{C}_u^{\mathrm{DTriv}}\,[\![\lambda i.r]\!] & = & \lambda\langle i, \kappa \rangle.\mathcal{C}_u^{\mathrm{DRoot}}\,[\![r]\!] \mathbin{@} \kappa
\end{array}
$$

Figure 4: Uncurried CPS transformation on pure DS terms

$$
\begin{array}{rll}
r \in \mathrm{CRoot}_u & & \text{— domain of uncurried CPS } \lambda\text{-terms} \\
e \in \mathrm{CExp}_u & & \text{— domain of uncurried CPS serious expressions} \\
t \in \mathrm{CTriv}_u & & \text{— domain of uncurried CPS trivial expressions} \\
i, k, v \in \mathrm{Ide} & & \text{— domain of identifiers}
\end{array}
$$

$$
\begin{array}{l}
r ::= \lambda k.e \\
e ::= k \mathbin{@} t \mid t_0 \mathbin{@} \langle t_1, \lambda v.e \rangle \\
t ::= i \mid \lambda\langle i, k \rangle.e \mid v
\end{array}
$$

Figure 5: Syntax of uncurried CPS λ-terms

$$
\begin{array}{l}
e ::= \ldots \mid e_0 \mathbin{@} \langle e_1, e_2 \rangle \\
t ::= \ldots \mid \lambda\langle i_1, i_2 \rangle.r
\end{array}
$$

Figure 6: Extension of Figure 1 to uncurried DS λ-terms

$$\begin{aligned}
\mathcal{C}_c^{\mathrm{DExp}}\,[\![e_0 \mathbin{@} \langle e_1, e_2\rangle]\!] &= \lambda\kappa.\,\mathcal{C}_c^{\mathrm{DExp}}\,[\![e_0]\!] \mathbin{@} \lambda t_0.\\
&\qquad \mathcal{C}_c^{\mathrm{DExp}}\,[\![e_1]\!] \mathbin{@} \lambda t_1.\\
&\qquad \mathcal{C}_c^{\mathrm{DExp}}\,[\![e_2]\!] \mathbin{@} \lambda t_2.\\
&\qquad (t_0 \mathbin{@} \langle t_1, t_2\rangle) \mathbin{@} \kappa\\
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![\lambda\langle i_1, i_2\rangle.r]\!] &= \lambda\langle i_1, i_2\rangle.\mathcal{C}_c^{\mathrm{DRoot}}\,[\![r]\!]
\end{aligned}$$

Figure 7: Extension of Figure 2 to uncurried DS λ-terms

4 and 5). The corresponding extension of the λ-calculus is displayed in Figure 6. The corresponding extension of the CPS transformation is displayed in Figure 7.

The uncurried CPS counterpart of the identity function $\lambda x.x$ thus reads as follows.

$$\lambda\langle x, k_1\rangle.k_1 \mathbin{@} x$$

CPS-transforming this result reads as follows.

$$\lambda\langle x,\, k_1,\, k_2\rangle.k_1 \mathbin{@} \langle x, k_2\rangle$$

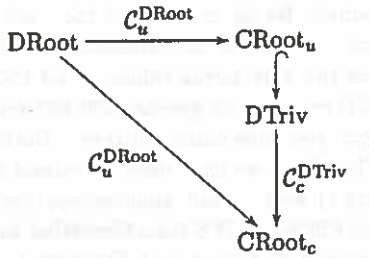
This term contains only one occurrence of k_2, but we cannot η-reduce it precisely because the term is uncurried.

The nature of this second failing suggests to mix curried and uncurried CPS, as investigated in Section 4.

4 Idempotence: Sort Of

Let us compose the curried CPS transformation (Figure 2) with the uncurried CPS transformation (Figure 4). To this end, and as was done implicitly in Section 3, we need to embed the output of the uncurried CPS transformation into the λ-calculus as extended in Figure 6. This embedding is the obvious one. The CPS transformation yields a trivial term, which compels us to compose $\mathcal{C}_c^{\mathrm{DTriv}}$ with $\mathcal{C}_u^{\mathrm{DRoot}}$.

Theorem 1 *$\mathcal{C}_c^{\mathrm{DTriv}}$ is the identity function over call-by-value uncurried CPS terms.*



Proof. We map $\mathcal{C}_c^{\mathrm{DTriv}}$ (Figures 2 and 7) over the output BNF of $\mathcal{C}_u^{\mathrm{DRoot}}$ (Figure 5).

$$\begin{aligned}
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![\lambda k.r]\!] &= \lambda k.\mathcal{C}_c^{\mathrm{DRoot}}\,[\![r]\!]\\
\mathcal{C}_c^{\mathrm{DRoot}}\,[\![e]\!] &= \mathcal{C}_c^{\mathrm{DExp}}\,[\![e]\!]\\
\mathcal{C}_c^{\mathrm{DExp}}\,[\![k \mathbin{@} t]\!] &= \lambda\kappa.\mathcal{C}_c^{\mathrm{DExp}}\,[\![k]\!] \mathbin{@} \lambda t_0.\mathcal{C}_c^{\mathrm{DExp}}\,[\![t]\!] \mathbin{@} \lambda t_1.(t_0 \mathbin{@} t_1) \mathbin{@} \kappa\\
&= \lambda\kappa.(k \mathbin{@} \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t]\!]) \mathbin{@} \kappa\\
&\qquad \text{after two administrative reductions}\\
&=_\eta\ k \mathbin{@} \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t]\!]\\
\mathcal{C}_c^{\mathrm{DExp}}\,[\![t_0 \mathbin{@} \langle t_1, \lambda v.e\rangle]\!] &= \lambda\kappa.\,\mathcal{C}_c^{\mathrm{DExp}}\,[\![t_0]\!] \mathbin{@} \lambda t_0'.\\
&\qquad \mathcal{C}_c^{\mathrm{DExp}}\,[\![t_1]\!] \mathbin{@} \lambda t_1'.\\
&\qquad \mathcal{C}_c^{\mathrm{DExp}}\,[\![\lambda v.e]\!] \mathbin{@} \lambda t_2'.\\
&\qquad (t_0' \mathbin{@} \langle t_1', t_2'\rangle) \mathbin{@} \kappa\\
&= \lambda\kappa.(\mathcal{C}_c^{\mathrm{DTriv}}\,[\![t_0]\!] \mathbin{@} \langle \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t_1]\!], \lambda v.\mathcal{C}_c^{\mathrm{DExp}}\,[\![e]\!]\rangle) \mathbin{@} \kappa\\
&\qquad \text{after three administrative reductions}\\
&=_\eta\ \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t_0]\!] \mathbin{@} \langle \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t_1]\!], \lambda v.\mathcal{C}_c^{\mathrm{DExp}}\,[\![e]\!]\rangle\\
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![x]\!] &= x\\
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![\lambda\langle i_1, i_2\rangle.r]\!] &= \lambda\langle i_1, i_2\rangle.\mathcal{C}^{\mathrm{DRoot}}\,[\![r]\!]
\end{aligned}$$

...which is the identity transformation. □

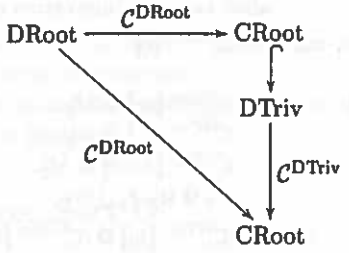
5 Idempotence: Yes

Leaving currying behind, let us go back to the pure λ-calculus and draw lessons from Theorem 1. The uncurried transformation introduced too many continuations because the λ declaring values, in a CPS term, is trivial in the sense of Reynolds [21] — or more precisely, it is total, since β-reducing it yields a λ-abstraction, and thus cannot diverge. Such total functions need no continuations. Therefore, we only need to extend the original syntax of the λ-calculus (Figure 1) with "total" annotations (decorating λ and @ with a hat), and to extend Plotkin's CPS transformation to cater for them. This possibility was mentioned in Danvy and Filinski's work [5, Sect. 4.3] and used in a call-by-name setting by Danvy and Hatcliff [7].

The corresponding extension of the λ-calculus is displayed in Figure 8.[1] The corresponding extension of the CPS transformation is displayed in Figure 9. The BNF of its output is displayed in Figure 10. Again, the embedding of such output terms in the extended λ-calculus is the obvious one.

We are now in position to state our main theorem.

Theorem 2 *$\mathcal{C}^{\mathrm{DTriv}}$ is the identity function over call-by-value CPS terms.*



Proof. Follows the proof of Theorem 1 *mutatis mutandis.* □

Corollary 1 (Idempotence) *$\mathcal{C}^{\mathrm{DRoot}}$ is idempotent.*

[1] This extension of the λ-calculus to trivial DS λ-terms comes with the closure condition that serious (resp. trivial) λ-abstractions can only and exclusively occur in serious (resp. trivial) applications, during reduction.

$$t ::= \ldots \mid \widehat{\lambda}\, i\,.\,t \mid t_0 \,\widehat{@}\, t_1$$

Figure 8: Extension of Figure 1 to trivial DS λ-terms

$$\begin{array}{rcl}
\mathcal{C}^{\mathrm{DRoot}} & : & \mathrm{DRoot} \to \mathrm{CRoot} \\
\mathcal{C}^{\mathrm{DRoot}}\,[\![e]\!] & = & \mathcal{C}^{\mathrm{DExp}}\,[\![e]\!] \\
\\
\mathcal{C}^{\mathrm{DExp}}\,[\![t]\!] & = & \lambda\kappa.\kappa\,@\,\mathcal{C}^{\mathrm{DTriv}}\,[\![t]\!] \\
\mathcal{C}^{\mathrm{DExp}}\,[\![e_0\,@\,e_1]\!] & = & \lambda\kappa.\mathcal{C}^{\mathrm{DExp}}\,[\![e_0]\!]\,@\,\lambda t_0.\mathcal{C}^{\mathrm{DExp}}\,[\![e_1]\!]\,@\,\lambda t_1.(t_0\,\widehat{@}\,t_1)\,@\,\kappa \\
\\
\mathcal{C}^{\mathrm{DTriv}}\,[\![i]\!] & = & i \\
\mathcal{C}^{\mathrm{DTriv}}\,[\![\lambda i.r]\!] & = & \widehat{\lambda}\, i\,.\,\mathcal{C}^{\mathrm{DRoot}}\,[\![r]\!] \\
\mathcal{C}^{\mathrm{DTriv}}\,[\![\widehat{\lambda}\, i\,.\,t]\!] & = & \widehat{\lambda}\, i\,.\,\mathcal{C}^{\mathrm{DTriv}}\,[\![t]\!] \\
\mathcal{C}^{\mathrm{DTriv}}\,[\![t_0\,\widehat{@}\,t_1]\!] & = & \mathcal{C}^{\mathrm{DTriv}}\,[\![t_0]\!]\,\widehat{@}\,\mathcal{C}^{\mathrm{DTriv}}\,[\![t_1]\!]
\end{array}$$

Figure 9: Extended call-by-value CPS transformation on pure DS terms

$$\begin{array}{rll}
r \in \mathrm{CRoot} & & \text{— domain of CPS } \lambda\text{-terms} \\
e \in \mathrm{CExp} & & \text{— domain of CPS serious expressions} \\
t \in \mathrm{CTriv} & & \text{— domain of CPS trivial expressions} \\
i, k, v \in \mathrm{Ide} & & \text{— domain of identifiers}
\end{array}$$

$$\begin{array}{rcl}
r & ::= & \lambda k.e \\
e & ::= & k\,@\,t \mid (t_0\,\widehat{@}\,t_1)\,@\,\lambda v.e \\
t & ::= & i \mid \widehat{\lambda}\, i\,.\,r \mid v
\end{array}$$

Figure 10: Syntax of call-by-value CPS λ-terms with totality annotations

6 Iterating the CPS Transformation over Types

The BNF of DS types reads as follows. (b denotes a base type.)

$$t \ ::= \ b \mid t_1 \to t_2$$

The call-by-value CPS transformation over types reads as follows [14, 15, 16, 19].

$$\begin{aligned} \mathcal{C}(\!|t|\!) &= (\mathcal{C}\langle t\rangle \to \mathrm{Ans}) \to \mathrm{Ans} = \neg_{\mathrm{Ans}}\neg_{\mathrm{Ans}}\mathcal{C}\langle t\rangle \\ \mathcal{C}\langle b\rangle &= b \\ \mathcal{C}\langle t_1 \to t_2\rangle &= \mathcal{C}\langle t_1\rangle \to \mathcal{C}(\!|t_2|\!) \end{aligned}$$

for some domain of answers Ans.

We follow Hatcliff and Danvy's notation [16], and thus distinguish between the type of serious terms (handled by $\mathcal{C}(\!|\cdot|\!)$) and the type of trivial terms (handled by $\mathcal{C}\langle\cdot\rangle$). For conciseness, we also abbreviate $t \to \mathrm{Ans}$ by $\neg_{\mathrm{Ans}}t$.

As pointed out in Section 5, however, one function space in the CPS transformation on types is trivial. To match the extended transform on terms, let us annotate it and extend the CPS transformation over trivial function spaces [7].

$$\begin{aligned} \mathcal{C}(\!|t|\!) &= (\mathcal{C}\langle t\rangle \to \mathrm{Ans}) \to \mathrm{Ans} \\ \mathcal{C}\langle b\rangle &= b \\ \mathcal{C}\langle t_1 \to t_2\rangle &= \mathcal{C}\langle t_1\rangle \to \mathcal{C}(\!|t_2|\!) \\ \mathcal{C}\langle t_1 \to t_2\rangle &= \mathcal{C}\langle t_1\rangle \to \mathcal{C}\langle t_2\rangle \end{aligned}$$

We are now equipped to iterate the CPS transformation, starting from the pure λ-calculus.

Since the result of the CPS transformation is trivial, we consider $\mathcal{C}\langle\cdot\rangle \circ \mathcal{C}(\!|\cdot|\!)$. Calculating the first case yields:

$$\begin{aligned} \mathcal{C}\langle\mathcal{C}(\!|b|\!)\rangle &= \mathcal{C}\langle(\mathcal{C}\langle b\rangle \to \mathrm{Ans}) \to \mathrm{Ans}\rangle \\ &= \mathcal{C}\langle\mathcal{C}\langle b\rangle \to \mathrm{Ans}\rangle \to \mathcal{C}(\!|\mathrm{Ans}|\!) \\ &= (\mathcal{C}\langle\mathcal{C}\langle b\rangle\rangle \to \mathcal{C}(\!|\mathrm{Ans}|\!)) \to \mathcal{C}(\!|\mathrm{Ans}|\!) \\ &= \neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\mathcal{C}\langle\mathcal{C}\langle b\rangle\rangle \end{aligned}$$

Calculating the second case yields:

$$\begin{aligned} \mathcal{C}\langle\mathcal{C}(\!|t_1 \to t_2|\!)\rangle &= (\mathcal{C}\langle\mathcal{C}\langle t_1 \to t_2\rangle\rangle \to \mathcal{C}(\!|\mathrm{Ans}|\!)) \to \mathcal{C}(\!|\mathrm{Ans}|\!) \\ &= \neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\mathcal{C}\langle\mathcal{C}\langle t_1 \to t_2\rangle\rangle \end{aligned}$$

and

$$\begin{aligned} \mathcal{C}\langle\mathcal{C}\langle t_1 \to t_2\rangle\rangle &= \mathcal{C}\langle\mathcal{C}\langle t_1\rangle \to \mathcal{C}(\!|t_2|\!)\rangle \\ &= \mathcal{C}\langle\mathcal{C}\langle t_1\rangle\rangle \to \mathcal{C}\langle\mathcal{C}(\!|t_2|\!)\rangle \\ &= \mathcal{C}\langle\mathcal{C}\langle t_1\rangle\rangle \to (\mathcal{C}\langle\mathcal{C}\langle t_2\rangle\rangle \to \mathcal{C}(\!|\mathrm{Ans}|\!)) \to \mathcal{C}(\!|\mathrm{Ans}|\!) \\ &= \mathcal{C}\langle\mathcal{C}\langle t_1\rangle\rangle \to \neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\mathcal{C}\langle\mathcal{C}\langle t_2\rangle\rangle \end{aligned}$$

The three results

$$\begin{aligned} \mathcal{C}\langle\mathcal{C}(\!|b|\!)\rangle &= \neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\mathcal{C}\langle\mathcal{C}\langle b\rangle\rangle \\ \mathcal{C}\langle\mathcal{C}(\!|t_1 \to t_2|\!)\rangle &= \neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}(\mathcal{C}\langle\mathcal{C}\langle t_1\rangle\rangle \to \neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\mathcal{C}\langle\mathcal{C}\langle t_2\rangle\rangle) \\ \mathcal{C}\langle\mathcal{C}\langle t_1 \to t_2\rangle\rangle &= \mathcal{C}\langle\mathcal{C}\langle t_1\rangle\rangle \to \neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\neg_{\mathcal{C}(\!|\mathrm{Ans}|\!)}\mathcal{C}\langle\mathcal{C}\langle t_2\rangle\rangle \end{aligned}$$

can be compared with

$$\begin{aligned} \mathcal{C}(\!|b|\!) &= \neg_{\mathrm{Ans}}\neg_{\mathrm{Ans}}\mathcal{C}\langle b\rangle \\ \mathcal{C}(\!|t_1 \to t_2|\!) &= \neg_{\mathrm{Ans}}\neg_{\mathrm{Ans}}(\mathcal{C}\langle t_1\rangle \to \neg_{\mathrm{Ans}}\neg_{\mathrm{Ans}}\mathcal{C}\langle t_2\rangle) \\ \mathcal{C}\langle t_1 \to t_2\rangle &= \mathcal{C}\langle t_1\rangle \to \neg_{\mathrm{Ans}}\neg_{\mathrm{Ans}}\mathcal{C}\langle t_2\rangle \end{aligned}$$

Iterating the CPS transformation on types thus amounts to refining the domain of answers with its CPS counterpart. This was implicitly the case in Danvy and Filinski's earlier work on iterating the CPS transformation [4], which to the best of our knowledge is the only other work considering the question. This work, however, investigates a family of control operators shift_n and reset_n that follows the hierarchy of iterating the CPS transformation. Given a n-level term (*i.e.*, a term containing occurrences of shift_n and reset_n), $n+1$ CPS transformations are necessary to obtain a purely functional, continuation-passing term, over which the CPS transformation of Section 5 is idempotent.

We have been asked whether a detour via Moggi's monadic metalanguage could help [16, 18]. In the present case, the answer is no because, for example, applying the continuation-monad constructor to a continuation monad does not yield a monad [28]. Therefore, idempotence does not make any sense.

7 Other Continuation-Passing Styles

One may wonder whether other CPS transformations (call-by-name [20], Reynolds-style [22], or after evaluation-order analysis [6, 7]) are idempotent. The answer is generally no, even if one adds triviality annotations.

However, CPS terms are evaluation-order independent [20, 21], and thus in particular they can be evaluated — *or CPS-transformed* — using call-by-value. It appears that a strategy similar to the one of Section 5 leads to idempotence for all of these CPS transformations.

We reproduce it here for call-by-name. Figure 11 displays the syntax of the call-by-name λ-calculus. (NB: under call-by-name, identifiers are serious terms.) Figures 12 and 13 display Plotkin's CPS transformation and the syntactic characterization of its output, after administrative reductions.

Theorem 3 *C^{DTriv} is the identity function over call-by-name CPS terms.*

8 Conclusion

Programming intuition suggests that in the absence of control operators, CPS-transforming a tail-recursive (*i.e.*, iterative) program ought to yield a program with insignificant continuations. We have formalized this insignificance syntactically though η-reduction and idempotence, namely: CPS transforming a CPS-transformed program yields a program where all the new continuations can be η-reduced, leading back the original CPS-transformed program. Type-wise, idempotence is reflected in the polymorphic domain of answers, that can be variously refined.

To this end, we had to use a notion of "trivial function space" that need not be CPS transformed. This notion was suggested earlier by Danvy and Filinski [5, Section 4.3] in a call-by-value setting and used by Danvy and Hatcliff in a call-by-name setting [7]. It can be bypassed by directly considering CPS transformations with multiple continuations (see Figures 14 and 15) instead of composing the CPS transformation. In these CPS transformations with multiple continuations, by construction, all outer continuations can be η-reduced [4].

The approach we have taken here is essentially syntactic. To be meaningful, it should be doubled with a semantical study such as Filinski's in his PhD thesis [10], with an eye on the logical interpretation of idempotence.

$$
\begin{array}{lll}
r \in \mathrm{DRoot} & & \text{— domain of DS } \lambda\text{-terms} \\
e \in \mathrm{DExp} & & \text{— domain of DS serious expressions} \\
t \in \mathrm{DTriv} & & \text{— domain of DS trivial expressions} \\
i \in \mathrm{Ide} & & \text{— domain of identifiers}
\end{array}
$$

$$
\begin{array}{l}
r ::= e \\
e ::= t \mid i \mid e \mathbin{@} r \\
t ::= \lambda i.r \mid \widehat{\lambda}\, i.t \mid t \mathbin{\widehat{@}} r
\end{array}
$$

Figure 11: Extended syntax of pure direct-style (DS) λ-terms — call-by-name

$$
\begin{array}{rcl}
C^{\mathrm{DRoot}} & : & \mathrm{DRoot} \to \mathrm{CRoot} \\
C^{\mathrm{DRoot}}\,[\![e]\!] & = & C^{\mathrm{DExp}}\,[\![e]\!] \\
\\
C^{\mathrm{DExp}}\,[\![t]\!] & = & \lambda\kappa.\kappa \mathbin{@} C^{\mathrm{DTriv}}\,[\![t]\!] \\
C^{\mathrm{DExp}}\,[\![i]\!] & = & \lambda\kappa.i \mathbin{@} \kappa \\
C^{\mathrm{DExp}}\,[\![e \mathbin{@} r]\!] & = & \lambda\kappa.C^{\mathrm{DExp}}\,[\![e]\!] \mathbin{@} \lambda t.(t \mathbin{\widehat{@}} C^{\mathrm{DRoot}}\,[\![r]\!]) \mathbin{@} \kappa \\
\\
C^{\mathrm{DTriv}}\,[\![\lambda i.r]\!] & = & \widehat{\lambda}\, i.\, C^{\mathrm{DRoot}}\,[\![r]\!] \\
C^{\mathrm{DTriv}}\,[\![\widehat{\lambda}\, i.t]\!] & = & \widehat{\lambda}\, i.\, C^{\mathrm{DTriv}}\,[\![t]\!] \\
C^{\mathrm{DTriv}}\,[\![t \mathbin{\widehat{@}} r]\!] & = & C^{\mathrm{DTriv}}\,[\![t]\!] \mathbin{\widehat{@}} C^{\mathrm{DRoot}}\,[\![r]\!]
\end{array}
$$

Figure 12: Extended call-by-name CPS transformation on pure DS terms

$$
\begin{array}{lll}
r \in \mathrm{CRoot} & & \text{— domain of CPS } \lambda\text{-terms} \\
e \in \mathrm{CExp} & & \text{— domain of CPS serious expressions} \\
t \in \mathrm{CTriv} & & \text{— domain of CPS trivial expressions} \\
i, k, v \in \mathrm{Ide} & & \text{— domain of identifiers}
\end{array}
$$

$$
\begin{array}{l}
r ::= \lambda k.e \\
e ::= k \mathbin{@} t \mid (t \mathbin{\widehat{@}} r) \mathbin{@} \lambda v.e \mid i \mathbin{@} \lambda v.e \\
t ::= \widehat{\lambda}\, i.r \mid v
\end{array}
$$

Figure 13: Syntax of call-by-name CPS λ-terms with totality annotations

$$
\begin{array}{rcl}
\mathcal{C}_c^{\mathrm{DRoot}}\,[\![e]\!] & = & \lambda\kappa_1.\lambda\kappa_2.\lambda\kappa_3.((\mathcal{C}_c^{\mathrm{DExp}}\,[\![e]\!]\,@\,\kappa_1)\,@\,\kappa_2)\,@\,\kappa_3 \\
\\
\mathcal{C}_c^{\mathrm{DExp}}\,[\![t]\!] & = & \lambda\kappa_1.\lambda\kappa_2.\lambda\kappa_3.((\kappa_1\,@\,\mathcal{C}_c^{\mathrm{DTriv}}\,[\![t]\!])\,@\,\kappa_2)\,@\,\kappa_3 \\
\mathcal{C}_c^{\mathrm{DExp}}\,[\![e_0\,@\,e_1]\!] & = & \lambda\kappa_1.\lambda\kappa_2.\lambda\kappa_3.((\mathcal{C}_c^{\mathrm{DExp}}\,[\![e_0]\!]\,@\,A_0)\,@\,\kappa_2)\,@\,\kappa_3 \\
\text{where } A_0 & = & \lambda t_0.\lambda\kappa_2'.\lambda\kappa_3'.((\mathcal{C}_c^{\mathrm{DExp}}\,[\![e_1]\!]\,@\,A_1)\,@\,\kappa_2')\,@\,\kappa_3' \\
\text{where } A_1 & = & \lambda t_1.\lambda\kappa_2''.\lambda\kappa_3''.(((t_0\,@\,t_1)\,@\,\kappa_1)\,@\,\kappa_2'')\,@\,\kappa_3'' \\
\\
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![i]\!] & = & i \\
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![\lambda i.r]\!] & = & \lambda i.\mathcal{C}_c^{\mathrm{DRoot}}\,[\![r]\!]
\end{array}
$$

Figure 14: Triple CPS transformation on call-by-value λ-terms

$$
\begin{array}{rcl}
\mathcal{C}^{\mathrm{DRoot}}\,[\![e]\!] & = & \lambda\kappa_1.\lambda\kappa_2.\lambda\kappa_3.((\mathcal{C}^{\mathrm{DExp}}\,[\![e]\!]\,@\,\kappa_1)\,@\,\kappa_2)\,@\,\kappa_3 \\
\\
\mathcal{C}^{\mathrm{DExp}}\,[\![t]\!] & = & \lambda\kappa_1.\lambda\kappa_2.\lambda\kappa_3.((\kappa_1\,@\,\mathcal{C}^{\mathrm{DTriv}}\,[\![t]\!])\,@\,\kappa_2)\,@\,\kappa_3 \\
\mathcal{C}^{\mathrm{DExp}}\,[\![i]\!] & = & \lambda\kappa_1.\lambda\kappa_2.\lambda\kappa_3.((i\,@\,\kappa_1)\,@\,\kappa_2)\,@\,\kappa_3 \\
\mathcal{C}^{\mathrm{DExp}}\,[\![e\,@\,r]\!] & = & \lambda\kappa_1.\lambda\kappa_2.\lambda\kappa_3.((\mathcal{C}^{\mathrm{DExp}}\,[\![e]\!]\,@\,A)\,@\,\kappa_2)\,@\,\kappa_3 \\
\text{where } A & = & \lambda t.\lambda\kappa_2'.\lambda\kappa_3'.(((t\,@\,B)\,@\,\kappa_1)\,@\,\kappa_2')\,@\,\kappa_3' \\
\text{where } B & = & \lambda\kappa_1''.\lambda\kappa_2''.\lambda\kappa_3''.((\mathcal{C}^{\mathrm{DRoot}}\,[\![r]\!]\,@\,\kappa_1'')\,@\,\kappa_2'')\,@\,\kappa_3'' \\
\\
\mathcal{C}^{\mathrm{DTriv}}\,[\![\lambda i.r]\!] & = & \lambda i.\mathcal{C}^{\mathrm{DRoot}}\,[\![r]\!]
\end{array}
$$

Figure 15: Triple CPS transformation on call-by-name λ-terms

Acknowledgements

Thanks to Andrzej Filinski, John Hatcliff and Ian Stark for discussions. The diagrams of Sections 4 and 5 were drawn with Kristoffer Rose's XY-pic package.

References

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

[2] William Clinger, Daniel P. Friedman, and Mitchell Wand. A scheme for a higher-level semantic algebra. In John Reynolds and Maurice Nivat, editors, *Algebraic Methods in Semantics*, pages 237–250. Cambridge University Press, 1985.

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

[4] Olivier Danvy and Andrzej Filinski. Abstracting control. In Mitchell Wand, editor, *Proceedings of the 1990 ACM Conference on Lisp and Functional Programming*, pages 151–160, Nice, France, June 1990. ACM Press.

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

[6] Olivier Danvy and John Hatcliff. CPS transformation after strictness analysis. *ACM Letters on Programming Languages and Systems*, 1(3):195–212, 1993.

[7] Olivier Danvy and John Hatcliff. On the transformation between direct and continuation semantics. In Stephen Brookes, Michael Main, Austin Melton, Michael Mislove, and David Schmidt, editors, *Proceedings of the 9th Conference on Mathematical Foundations of Programming Semantics*, number 802 in Lecture Notes in Computer Science, pages 627–648, New Orleans, Louisiana, April 1993.

[8] 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.

[9] Matthias Felleisen. The theory and practice of first-class prompts. In Jeanne Ferrante and Peter Mager, editors, *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 180–190, San Diego, California, January 1988.

[10] Andrzej Filinski. *Controlling Effects.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1996.

[11] Michael J. Fischer. Lambda-calculus schemata. In Talcott [26], pages 259–288. An earlier version appeared in an ACM Conference on Proving Assertions about Programs, SIGPLAN Notices, Vol. 7, No. 1, January 1972.

[12] Pascal Fradet and Daniel Le Métayer. Compilation of functional languages by program transformation. *ACM Transactions on Programming Languages and Systems*, 13:21–51, 1991.

[13] Michael Gordon. *The Denotational Description of Programming Languages.* Springer-Verlag, 1979.

[14] Timothy G. Griffin. A formulae-as-types notion of control. In Paul Hudak, editor, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, pages 47–58, San Francisco, California, January 1990. ACM Press.

[15] Bob Harper and Mark Lillibridge. Polymorphic type assignment and CPS conversion. In Talcott [26].

[16] 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.

[17] David Kranz, Richard Kesley, Jonathan Rees, Paul Hudak, Jonathan Philbin, and Norman Adams. Orbit: An optimizing compiler for Scheme. In *Proceedings of the ACM SIGPLAN'86 Symposium on Compiler Construction*, pages 219–233, Palo Alto, California, June 1986.

[18] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[19] Chetan R. Murthy. *Extracting Constructive Content from Classical Proofs.* PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, 1990.

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

[21] John C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proceedings of 25th ACM National Conference*, pages 717–740, Boston, Massachusetts, 1972.

[22] John C. Reynolds. On the relation between direct and continuation semantics. In Jacques Loeckx, editor, *2nd Colloquium on Automata, Languages and Programming*, number 14 in Lecture Notes in Computer Science, pages 141–156, Saarbrücken, West Germany, July 1974.

[23] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In Talcott [26], pages 289–360.

[24] Guy L. Steele Jr. Rabbit: A compiler for Scheme. Technical Report AI-TR-474, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts, May 1978.

[25] Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, 1977.

[26] Carolyn L. Talcott, editor. *Special issue on continuations (Part I)*, LISP and Symbolic Computation, Vol. 6, Nos. 3/4. Kluwer Academic Publishers, December 1993.

[27] Dirk van Dalen. *Logic and Structure (second edition).* Universitext. Springer-Verlag, 1985.

[28] Philip Wadler. Monads and composable continuations. In Carolyn L. Talcott, editor, *Special issue on continuations (Part II)*, LISP and Symbolic Computation, Vol. 7, No. 1, pages 39–55. Kluwer Academic Publishers, January 1994.

Recent Publications in the BRICS Report Series

RS-96-14 Olivier Danvy and Karoline Malmkjær. *On the Idempotence of the CPS Transformation*. May 1996. 17 pp.

RS-96-13 Olivier Danvy and René Vestergaard. *Semantics-Based Compiling: A Case Study in Type-Directed Partial Evaluation*. May 1996. 28 pp. Appears in Kuchen and Swierstra, editors, *8th International Symposium on Programming Languages, Implementations, Logics, and Programs*, PLILP '96 Proceedings, LNCS 1140, 1996, pages 182–??

RS-96-12 Lars Arge, Darren Erik Vengroff, and Jeffrey Scott Vitter. *External-Memory Algorithms for Processing Line Segments in Geographic Information Systems*. May 1996. 34 pp. A shorter version of this paper appears in Spirakis, editor, *Algorithms - ESA '95: Third Annual European Symposium Proceedings*, LNCS 979, 1995, pages 295–310.

RS-96-11 Devdatt P. Dubhashi, David A. Grable, and Alessandro Panconesi. *Near-Optimal, Distributed Edge Colouring via the Nibble Method*. May 1996. 17 pp. Appears in Spirakis, editor, *Algorithms - ESA '95: Third Annual European Symposium Proceedings*, LNCS 979, 1995, pages 448–459. Invited to be published in a special issue of *Theoretical Computer Science* devoted to the proceedings of ESA '95.

RS-96-10 Torben Brauner and Valeria de Paiva. *Cut-Elimination for Full Intuitionistic Linear Logic*. April 1996. 27 pp. Also available as Technical Report 395, Computer Laboratory, University of Cambridge.

RS-96-9 Thore Husfeldt, Theis Rauhe, and Søren Skyum. *Lower Bounds for Dynamic Transitive Closure, Planar Point Location, and Parentheses Matching*. April 1996. 11 pp. Appears in *Nordic Journal of Computing*, 3(4):323–336, December 1996. Also in Karlson and Lingas, editors, *Algorithm Theory: 5th Scandinavian Workshop*, SWAT '96 Proceedings, LNCS 1097, 1996, pages 198–211.