On the Idempotence of the CPS Transformation

Olivier Danvy and Karoline Malmkjær

BRICS *

Department of Computer Science

Aarhus University †

{danvy,karoline}@brics.dk

May 1996

Abstract

The CPS (continuation-passing style) transformation on λ-terms has an interpretation both in programming languages, type theory, proof theory, and logic. Programming intuition suggests that it is idempotent, but this does not directly hold for all existing CPS transformations (Plotkin, Reynolds, Fischer, *etc.*). We rephrase it to make it syntactically idempotent, modulo η-reduction of the newly introduced continuation.

*Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

†Building 540, Ny Munkegade, DK-8000 Aarhus C, Denmark. 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 [10, 14, 17, 18], *e.g.*, for compiling programs [1, 15, 20]; on types, seen as propositions, it corresponds to a double-negation translation, *i.e.*, as a mapping from Classical Logic into Intuitionistic Logic [13, 16, 23].

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 can't possibly introduce much of a continuation.

One property of the λ-calculus even hints that the CPS transformation could actually be made 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. Section 6 concludes.

2 CPS: Values First vs. Continuations First

Broadly speaking, there are two classes of continuation-passing styles: those that put values first ($\lambda x.\lambda k.\ldots$), and those that put continuations first ($\lambda k.\lambda x.\ldots$). The "values first" class was investigated by Plotkin, Reynolds, and others [17, 18]. The "continuations first" class was investigated by Fischer, Fradet and Le Métayer, Sabry and Felleisen, and others [10, 11, 19].

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 [12, 21].

$$
\begin{array}{rclcrcl}
 & \text{Values first} & & \qquad & & \text{Continuations first} & \\
\mathrm{Cont} & = & \mathrm{Val} \to \mathrm{Ans} & & \mathrm{Cont} & = & \mathrm{Val} \to \mathrm{Ans} \\
\mathrm{Fun} & = & \mathrm{Val} \to \mathrm{Cont} \to \mathrm{Ans} & & \mathrm{Fun} & = & \mathrm{Cont} \to \mathrm{Val} \to \mathrm{Ans} \\
 & & & & & = & \mathrm{Cont} \to \mathrm{Cont}
\end{array}
$$

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 [17].

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

$$
\begin{array}{rcl}
r & ::= & e \\
e & ::= & t \mid e_0 \,@\, e_1 \\
t & ::= & i \mid \lambda i.r
\end{array}
$$

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

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

Figure 2: Curried CPS transformation on pure DS terms

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

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

Figure 3: Syntax of curried CPS λ-terms

Figure 1 specifies our source language: it is the pure λ-calculus. Following Reynolds [18], 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, 17, 19, 20]. 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 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.

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

CPS-transforming this result reads as follows.

$$\lambda\langle x,\ k_1,\ k_2\rangle.k_1 @ \langle x, k_2\rangle$$

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 $\mathcal{C}_u^{\mathrm{DRoot}}$.

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

Figure 4: Uncurried CPS transformation on pure DS terms

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

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

Figure 5: Syntax of uncurried CPS λ-terms

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

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

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

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

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

$$\begin{tikzcd}
\mathrm{DRoot} \arrow[r, "\mathcal{C}_u^{\mathrm{DRoot}}"] \arrow[ddr, "\mathcal{C}_u^{\mathrm{DRoot}}"'] & \mathrm{CRoot}_u \arrow[d, hook] \\
& \mathrm{DTriv} \arrow[d, "\mathcal{C}_c^{\mathrm{DTriv}}"] \\
& \mathrm{CRoot}_c
\end{tikzcd}$$

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

$$\begin{aligned}
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![\lambda k.r]\!] &= \lambda k.\mathcal{C}_c^{\mathrm{DRoot}}\,[\![r]\!] \\[1ex]
\mathcal{C}_c^{\mathrm{DRoot}}\,[\![e]\!] &= \mathcal{C}_c^{\mathrm{DExp}}\,[\![e]\!] \\[1ex]
\mathcal{C}_c^{\mathrm{DExp}}\,[\![k \mathbin{@} t]\!] &= \lambda\kappa.\mathcal{C}_c^{\mathrm{DExp}}\,[\![k]\!] \mathbin{@} \lambda t_0.\mathcal{C}_c^{\mathrm{DExp}}\,[\![t]\!] \mathbin{@} \lambda t_1.(t_0 \mathbin{@} t_1) \mathbin{@} \kappa \\
&= \lambda\kappa.(k \mathbin{@} \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t]\!]) \mathbin{@} \kappa \\
&=_\eta k \mathbin{@} \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t]\!] \\[1ex]
\mathcal{C}_c^{\mathrm{DExp}}\,[\![t_0 \mathbin{@} \langle t_1, \lambda v.e\rangle]\!] &= \lambda\kappa.\,\mathcal{C}_c^{\mathrm{DExp}}\,[\![t_0]\!] \mathbin{@} \lambda t_0'. \\
&\qquad \mathcal{C}_c^{\mathrm{DExp}}\,[\![t_1]\!] \mathbin{@} \lambda t_1'. \\
&\qquad \mathcal{C}_c^{\mathrm{DExp}}\,[\![\lambda v.e]\!] \mathbin{@} \lambda t_2'. \\
&\qquad (t_0' \mathbin{@} \langle t_1', t_2'\rangle) \mathbin{@} \kappa \\
&=_\beta (\mathcal{C}_c^{\mathrm{DTriv}}\,[\![t_0]\!] \mathbin{@} \langle \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t_1]\!], \lambda v.\mathcal{C}_c^{\mathrm{DExp}}\,[\![e]\!]\rangle) \mathbin{@} \kappa \\
&=_\eta \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t_0]\!] \mathbin{@} \langle \mathcal{C}_c^{\mathrm{DTriv}}\,[\![t_1]\!], \lambda v.\mathcal{C}_c^{\mathrm{DExp}}\,[\![e]\!]\rangle \\[1ex]
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![x]\!] &= x \\
\mathcal{C}_c^{\mathrm{DTriv}}\,[\![\lambda\langle i_1, i_2\rangle.r]\!] &= \lambda\langle i_1, i_2\rangle.\mathcal{C}^{\mathrm{DRoot}}\,[\![r]\!]
\end{aligned}$$

...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 [18] — 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 (overlining λ and @ with a hat), and to extend Plotkin's CPS transformation to cater for them. This possibility was

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

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

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

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

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

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

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

mentioned in Danvy and Filinski's work [5, Section 4.3] and carried out in a call-by-name setting by Danvy and Hatcliff [6].

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 CPS terms.*

$$\begin{tikzcd}
\mathrm{DRoot} \arrow[r, "\mathcal{C}^{\mathrm{DRoot}}"] \arrow[ddr, "\mathcal{C}^{\mathrm{DRoot}}"'] & \mathrm{CRoot} \arrow[d, hook] \\
& \mathrm{DTriv} \arrow[d, "\mathcal{C}^{\mathrm{DTriv}}"] \\
& \mathrm{CRoot}
\end{tikzcd}$$

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

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

6 Conclusion

Programming intuition suggests that the CPS transformation be idempotent, but existing CPS transformations do not satisfy this property. We have refined Plotkin's call-by-value CPS transformation with the concept of totality — as was suggested earlier by Danvy and Filinski [5, Section 4.3] and explored earlier on by Danvy and Hatcliff in a call-by-name setting [6] — and have obtained a syntactically idempotent CPS transformation. We explore its type counterpart in an extended version of this note [7].

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.

[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.

[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. Special issue on ESOP'92, the Fourth European Symposium on Programming, Rennes, France, February 1992.

[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. 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.

[7] Olivier Danvy and Karoline Malmkjær. On the idempotence of the CPS transformation. Technical report BRICS-RS-96-14, DAIMI, Computer Science Department, Aarhus University, Aarhus, Denmark, May 1996.

[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] Michael J. Fischer. Lambda-calculus schemata. In Talcott [22], pages 259–288. An earlier version appeared in an ACM Conference on Proving Assertions about Programs, SIGPLAN Notices, Vol. 7, No. 1, January 1972.

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

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

[13] 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.

[14] 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.

[15] 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.

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

[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. In *Proceedings of 25th ACM National Conference*, pages 717–740, Boston, Massachusetts, 1972.

[19] Amr Sabry and Matthias Felleisen. Reasoning about programs in continuation-passing style. In Talcott [22], pages 289–360.

[20] 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.

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

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

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