BRICS RS-04-39 Danvy & Nielsen: CPS Transformation of Beta-Redexes

BRICS

Basic Research in Computer Science

CPS Transformation of Beta-Redexes

Olivier Danvy
Lasse R. Nielsen

BRICS Report Series **RS-04-39**

ISSN 0909-0878 **December 2004**

See back inner page for a list of recent BRICS Report Series publications.
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 the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/04/39/`

CPS Transformation of Beta-Redexes *

Olivier Danvy and Lasse R. Nielsen
BRICS †
Department of Computer Science
University of Aarhus ‡

December 2004

Abstract

The extra compaction of the most compacting CPS transformation in existence, which is due to Sabry and Felleisen, is generally attributed to (1) making continuations occur first in CPS terms and (2) classifying more redexes as administrative. We show that this extra compaction is actually independent of the relative positions of values and continuations and furthermore that it is solely due to a context-sensitive transformation of beta-redexes. We stage the more compact CPS transformation into a first-order uncurrying phase and a context-insensitive CPS transformation. We also define a context-insensitive CPS transformation that provides the extra compaction. This CPS transformation operates in one pass and is dependently typed.

Keywords

Functional programming, program derivation, continuation-passing style (CPS), Plotkin, Fischer, one-pass CPS transformation, two-level lambda-calculus, generalized reduction, dependent types.

---

*Extended version of an article to appear in Information Processing Letters. A preliminary version was presented at the Third ACM SIGPLAN Workshop on Continuations (CW'01), January 16, 2001, London, UK [6].

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

‡IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark
E-mail: {`danvy`,`lrn`}`@brics.dk`

Contents

List of Figures

1 Introduction

1.1 Continuation-passing style (CPS)

The meaning of a λ-term, in general, depends on its evaluation order. Evaluation-order independence was one of the motivations for continuations [20, 26], and continuation-passing style was developed as an evaluation-order independent λ-encoding of λ-terms [8, 19]. In this λ-encoding, each evaluation context is represented by a λ-abstraction, called a *continuation*, and each λ-abstraction is passed a continuation in addition to its usual argument. All intermediate results are sent to a continuation and thus all calls are tail-calls. This λ-encoding gives rise to a variety of continuation-passing styles, whose structure is a subject of study in itself [12, 21, 25].

1.2 The CPS transformation

The format of CPS λ-terms was soon noticed to be of interest for the compiler writer [24], which in turn fostered interest in automating the transformation of λ-terms into CPS. Over the last twenty years, a wide body of CPS transformations has thus been developed for various purposes, e.g., to compile and to analyze programs, and to generate compilers [1, 10, 14, 23, 24, 27].

The naïve λ-encoding into CPS, however, generates a quite impressive inflation of lambdas, most of which form *administrative redexes* that can be safely reduced. Administrative reductions yield CPS terms corresponding to what one could write by hand. It has therefore become a challenge to eliminate as many administrative redexes as possible, at CPS-transformation time. (Contracting other β-redexes would correspond to simplifying the source term, which falls out of the scope of the CPS transformation.)

1.3 Sabry and Felleisen's optimization

In their article "Reasoning about Programs in Continuation-Passing Style" [22], Sabry and Felleisen present a CPS transformation that yields more compact terms than existing CPS transformations. For example [22, Footnote 6], CPS-transforming

$$((\lambda x.\lambda y.x)\, a)\, b$$

where a and b are (free) variables, yields the term

$$\lambda k.((\lambda x.((\lambda y.k\, x)\, b))\, a)$$

whereas earlier transformations, such as Steele's [24] or Danvy and Filinski's [5], yield the more voluminous term

$$\lambda k.((\lambda x.\lambda k_1.(k_1\, (\lambda y.\lambda k_2.k_2\, x)))\, a\, (\lambda m.m\, b\, k)).$$

Sabry and Felleisen's optimization relies on using Fischer's CPS (where continuations occur first, as in $\lambda k.\lambda x.e$), whereas earlier transformations use Plotkin's CPS (where values occur first, as in $\lambda x.\lambda k.e$).[1]

[1] As traditional, the reference to Fischer is a little bit stretched since Fischer's domain of discourse was uncurried Lisp functions [8]. But we also follow the tradition here.

1.4 This article

Section 2 reviews administrative reductions in the CPS transformation and characterizes Sabry and Felleisen's optimization, independently of the relative positions of values and continuations in CPS terms (i.e., both for Fischer's and Plotkin's CPS). Section 3 constructs a similarly compact CPS transformation by composing an uncurrying phase and an ordinary CPS transformation. Section 4 integrates the optimization in a context-insensitive, one-pass CPS transformation. Section 5 concludes.

2 Administrative reductions in the CPS transformation

2.1 Context-insensitive administrative reductions

Appel, Danvy and Filinski, and Wand each independently developed a "one-pass" CPS transformation for call by value [1, 5, 27]. This CPS transformation relies on a context-free characterization of administrative reductions, i.e., a characterization that is independent of any source term. This one-pass transformation, shown below for Plotkin's CPS, is formulated with a static, context-free distinction between (translation-time) administrative reductions and (run-time) reductions, using a two-level λ-calculus [5, 18].

$$
\begin{array}{rcl}
\llbracket \cdot \rrbracket_{\mathrm{p}} & : & \Lambda \to (\Lambda \to \Lambda) \to \Lambda \\
\llbracket x \rrbracket_{\mathrm{p}} & = & \overline{\lambda}\kappa.\kappa \mathbin{\overline{@}} x \\
\llbracket \lambda x.e \rrbracket_{\mathrm{p}} & = & \overline{\lambda}\kappa.\kappa \mathbin{\overline{@}} (\underline{\lambda} x.\underline{\lambda} k.\llbracket e \rrbracket'_{\mathrm{p}} \mathbin{\overline{@}} k) \\
\llbracket e_0\, e_1 \rrbracket_{\mathrm{p}} & = & \overline{\lambda}\kappa.\llbracket e_0 \rrbracket_{\mathrm{p}} \mathbin{\overline{@}} (\overline{\lambda} t_0.\llbracket e_1 \rrbracket_{\mathrm{p}} \mathbin{\overline{@}} (\overline{\lambda} t_1.(t_0 \mathbin{\underline{@}} t_1) \mathbin{\underline{@}} (\underline{\lambda} v.\kappa \mathbin{\overline{@}} v))) \\
\llbracket x \rrbracket'_{\mathrm{p}} & = & \overline{\lambda} k.k \mathbin{\underline{@}} x \\
\llbracket \lambda x.e \rrbracket'_{\mathrm{p}} & = & \overline{\lambda} k.k \mathbin{\underline{@}} (\underline{\lambda} x.\underline{\lambda} k.\llbracket e \rrbracket'_{\mathrm{p}} \mathbin{\overline{@}} k) \\
\llbracket e_0\, e_1 \rrbracket'_{\mathrm{p}} & = & \overline{\lambda} k.\llbracket e_0 \rrbracket_{\mathrm{p}} \mathbin{\overline{@}} (\overline{\lambda} t_0.\llbracket e_1 \rrbracket_{\mathrm{p}} \mathbin{\overline{@}} (\overline{\lambda} t_1.(t_0 \mathbin{\underline{@}} t_1) \mathbin{\underline{@}} k))
\end{array}
$$

"$\underline{\lambda}$" and "$\underline{@}$" denote hygienic abstract-syntax constructors and "$\overline{\lambda}$" and "$\overline{@}$" denote translation-time abstractions and (infix) applications, respectively.

A λ-term $e : \Lambda$ is CPS-transformed with $\underline{\lambda} k.\llbracket e \rrbracket'_{\mathrm{p}} \mathbin{\overline{@}} k$. (The reader is directed to [5, Section 2] for a construction of this one-pass CPS transformation based on control-flow analysis [23].)

The corresponding one-pass transformation for Fischer's CPS is as follows.

$$
\begin{array}{rcl}
\llbracket \cdot \rrbracket_{\mathrm{f}} & : & \Lambda \to (\Lambda \to \Lambda) \to \Lambda \\
\llbracket x \rrbracket_{\mathrm{f}} & = & \overline{\lambda}\kappa.\kappa \mathbin{\overline{@}} x \\
\llbracket \lambda x.e \rrbracket_{\mathrm{f}} & = & \overline{\lambda}\kappa.\kappa \mathbin{\overline{@}} (\underline{\lambda} k.\underline{\lambda} x.\llbracket e \rrbracket'_{\mathrm{f}} \mathbin{\overline{@}} k) \\
\llbracket e_0\, e_1 \rrbracket_{\mathrm{f}} & = & \overline{\lambda}\kappa.\llbracket e_0 \rrbracket_{\mathrm{f}} \mathbin{\overline{@}} (\overline{\lambda} t_0.\llbracket e_1 \rrbracket_{\mathrm{f}} \mathbin{\overline{@}} (\overline{\lambda} t_1.(t_0 \mathbin{\underline{@}} (\underline{\lambda} v.\kappa \mathbin{\overline{@}} v)) \mathbin{\underline{@}} t_1)) \\
\llbracket x \rrbracket'_{\mathrm{f}} & = & \overline{\lambda} k.k \mathbin{\underline{@}} x \\
\llbracket \lambda x.e \rrbracket'_{\mathrm{f}} & = & \overline{\lambda} k.k \mathbin{\underline{@}} (\underline{\lambda} k.\underline{\lambda} x.\llbracket e \rrbracket'_{\mathrm{f}} \mathbin{\overline{@}} k) \\
\llbracket e_0\, e_1 \rrbracket'_{\mathrm{f}} & = & \overline{\lambda} k.\llbracket e_0 \rrbracket_{\mathrm{f}} \mathbin{\overline{@}} (\overline{\lambda} t_0.\llbracket e_1 \rrbracket_{\mathrm{f}} \mathbin{\overline{@}} (\overline{\lambda} t_1.(t_0 \mathbin{\underline{@}} k) \mathbin{\underline{@}} t_1))
\end{array}
$$

A λ-term $e : \Lambda$ is CPS-transformed with $\underline{\lambda} k.\llbracket e \rrbracket'_{\mathrm{f}} \mathbin{\overline{@}} k$.

2.2 Context-sensitive administrative reductions

Sabry and Felleisen proposed a three-pass CPS transformation that (1) tags all the "new" lambdas introduced by the CPS transformation, (2) repeatedly reduces the β-redexes with a tagged lambda, and (3) untags the remaining tagged lambdas:

$$\begin{array}{rcl} [\![x]\!] & = & \overline{\lambda}k.k\,x \\ [\![\lambda x.e]\!] & = & \overline{\lambda}k.k\,(\overline{\lambda}k.\lambda x.[\![e]\!]\,k) \\ [\![e_0\,e_1]\!] & = & \overline{\lambda}k.[\![e_0]\!]\,(\overline{\lambda}t_0.[\![e_1]\!]\,(\overline{\lambda}t_1.t_0\,k\,t_1)) \end{array}$$

A λ-term e is CPS-transformed with $[\![e]\!]$.

An administrative reduction amounts to reducing a β-redex where the λ-abstraction is tagged.

In contrast to the Fischer-style one-pass CPS transformation of Section 2.1, this three-pass transformation (a) does not use @ for applications and is more implicit by not underlining abstract-syntax constructors; (b) is a first-order rewriting system whereas the one-pass transformation is a higher-order one; and (c) in addition, contains one more overlined λ-abstraction, namely the one declaring the continuation of a λ-abstraction. The extra overline makes administrative reductions context-sensitive, as illustrated below:

$$\begin{array}{rcl} [\![\lambda x.((\lambda y.y)\,x)]\!] & = & \overline{\lambda}k.k\,(\overline{\lambda}k.\lambda x.(\overline{\lambda}k.k\,(\overline{\lambda}k.\lambda y.(\overline{\lambda}k.k\,y)\,k))\,\overline{\lambda}t_0.(\overline{\lambda}k.k\,x)\,\overline{\lambda}t_1.t_0\,k\,t_1) \\ & \longrightarrow_{\overline{\beta}^+} & \overline{\lambda}k.k\,(\overline{\lambda}k.\lambda x.(\overline{\lambda}k.\lambda y.k\,y)\,k\,x) \\ & \longrightarrow_{\overline{\beta}} & \overline{\lambda}k.k\,(\overline{\lambda}k.\lambda x.(\lambda y.k\,y)\,x) \end{array}$$

The term $\overline{\lambda}k.\lambda x....$ arises from the transformation of $\lambda x....$ and cannot be administratively reduced. The term $\overline{\lambda}k.\lambda y....$ arises from the transformation of $\lambda y....$ and can be administratively reduced.

In contrast, in a context-insensitive one-pass CPS transformation, all overlined λ-abstractions are guaranteed to occur in an overlined application (and thus there is no need for post-erasure). A context-sensitive CPS transformation thus can perform more administrative reductions than a context-insensitive one.

Furthermore, we can precisely locate the (single) extra gain: for source β-redexes. Given a source β-redex, one can actually substitute the continuation of the application for the continuation of the abstraction:

$$(\lambda x.e[c/k])\,t_1$$

thereby enabling further administrative reductions inside e.

This reduction is not accounted for in a (say, Plotkin-style) one-pass CPS transformation, since in the particular case where t_0 denotes $\underline{\lambda}x.\underline{\lambda}k.e$, one does not simplify

$$(t_0\,\underline{@}\,t_1)\,\underline{@}\,c$$

into

$$(\underline{\lambda}x.e[c/k])\,\underline{@}\,t_1.$$

The reduction thus yields more compact CPS counterparts of source β-redexes, in that the translated λ-abstractions are not explicitly passed any continuation when they occur in a β-redex.[2]

On the other hand, a similar phenomenon already occurs for let expressions, as reviewed next.

2.3 CPS transformation of let expressions

The CPS transformation of let expressions reads as follows:

$$[\![\text{let } x = e' \text{ in } e]\!] \quad = \quad \overline{\lambda}\kappa.[\![e']\!]\overline{\lambda}t'.\underline{\text{let}}\; x = t' \;\underline{\text{in}}\; [\![e]\!]\kappa$$

In words, e is in tail-position in the let expression, and is CPS-transformed with respect to the same κ as the let expression. This technique is instrumental in binding-time analysis [3] and continuation-based partial evaluation [15].

Seeing let expressions as syntactic sugar for β-redexes, it appears clearly that the context-sensitive administrative reduction includes the standard let optimization, independently of whether continuations are put first or last. This administrative reduction, however, yields more.

2.4 CPS transformation of nested β-redexes

Extra mileage is obtained for (curried) λ-abstractions that are fully applied. CPS-transforming the curried application of a "n-ary" λ-abstraction to n arguments relocates the continuation of the application to the body of the λ-abstraction:

$$\begin{array}{l} [\![(\lambda x_1 .\ldots \lambda x_n.e)\, e_1 \ldots e_n]\!] \\ = \overline{\lambda}\kappa.[\![e_1]\!] \mathbin{\overline{@}} (\overline{\lambda}t_1 .\ldots [\![e_n]\!] \mathbin{\overline{@}} (\overline{\lambda}t_n.(\underline{\lambda}x_n .\ldots (\underline{\lambda}x_1.[\![e]\!] \mathbin{\overline{@}} \kappa) \mathbin{\underline{@}} t_1 \ldots) \mathbin{\underline{@}} t_n) \ldots) \\ [\![(\lambda x_1 .\ldots \lambda x_n.e)\, e_1 \ldots e_n]\!]' \\ = \overline{\lambda}k.[\![e_1]\!] \mathbin{\overline{@}} (\overline{\lambda}t_1 .\ldots [\![e_n]\!] \mathbin{\overline{@}} (\overline{\lambda}t_n.(\underline{\lambda}x_n .\ldots (\underline{\lambda}x_1.[\![e]\!]' \mathbin{\overline{@}} k) \mathbin{\underline{@}} t_1 \ldots) \mathbin{\underline{@}} t_n) \ldots) \end{array}$$

This extra mileage is independent of whether continuations are put first or last.

As a net effect, a term such as

$$(\lambda f.\lambda g.\lambda x.f\; x\; (g\; x))\; (a\; b)\; c\; (d\; e)$$

where a, b, c, d, and e are variables, is CPS transformed into (letting continuations occur last)

$$\lambda k.a\; b\; (\lambda f.(\lambda g.d\; e\; (\lambda x.f\; x\; (\lambda v_1.g\; x\; (\lambda v_2.v_1\; v_2\; k))))\; c).$$

Observe how the λ-abstractions $\lambda f....$ and $\lambda x....$ end up as the continuations of the applications $(a\, b)$ and $(d\, e)$, and how the application of $\lambda g....$ to c survives in the CPS term.

Letting continuations occur first would yield a similar term:

$$\lambda k.a\; (\lambda f.(\lambda g.d\; (\lambda x.f\; (\lambda v_1.g\; (\lambda v_2.v_1\; k\; v_2)\; x)\; x)\; e)\; c)\; b.$$

[2] As Shivers puts it and as can be read off their type, the translated λ-abstractions are 'promoted to continuations' [23].

2.5 Summary and conclusion

A CPS transformation with context-sensitive administrative reductions yields more compact CPS terms because it exposes more administrative redexes. The extra administrative reductions affect nested β-redexes corresponding to fully applied curried λ-abstractions, and reduce continuation-passing by promoting the inner λ-abstractions to continuations. These extra administrative reductions can be carried out independently of whether continuations occur first or last in CPS terms.

The extra compaction of Sabry and Felleisen's CPS transformation is therefore independent of the relative positions of values and continuations. Furthermore, it is solely due to a context-sensitive transformation of beta-redexes.

3 Staging the more compact CPS transformation

Sabry and Felleisen [22, Definition 7, page 306] identify a reduction β_{lift} moving the context of a β-redex into the body of the corresponding λ-abstraction:[3]

$$E[(\lambda x.e)e'] \longrightarrow \begin{array}{l}(\lambda x.E[e])e' \\ \text{where } E \neq [\,] \text{ and } x \notin FV(E)\end{array} \qquad (\beta_{\mathit{lift}})$$

They also pointed out that CPS-transforming a term e and mapping the result back to direct style yields a term in β_{lift}-normal form.

But a term in β_{lift}-normal form does not give rise to the extra context-sensitive administrative reduction of Section 2. Therefore, the extra power of the context-sensitive CPS transformation is solely due to β_{lift}.

The more compact CPS transformation can thus be staged as follows:

1. a phase uncurrying (and appropriately renaming, if need be) all β-redexes $(\lambda x_1. \ldots \lambda x_n.e)\, e_1 \ldots e_n$ into nested let expressions
$$\begin{array}{l} \text{let } x_1 = e_1 \\ \text{in } \;\; \text{let } x_2 = e_2 \\ \qquad \text{in} \ldots \;\; \text{let } x_n = e_n \\ \qquad\qquad\quad \text{in } e \end{array}$$

2. an ordinary, context-insensitive CPS transformation (either à la Plotkin or à la Fischer) handling let expressions.

The benefit of this staging, we believe, is three-fold: (a) it clarifies the extra compaction; (b) it extends a context-insensitive, one-pass CPS transformation; and (c) it suggests how to obtain even more compact terms. Indeed, in the same fashion as control-flow analysis can be used to locate the application sites of curried λ-abstractions in order to uncurry them [1, 11], the CPS transformation can benefit from control-flow information to promote more functions to continuations.

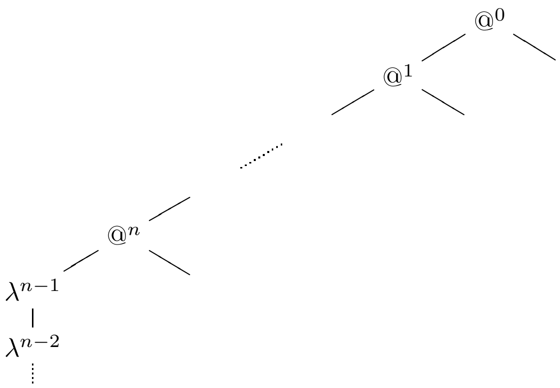
[3]The transitive closure of β_{lift} is a generalized reduction in the sense of Bloo, Kamareddine, and Nederpelt [2].

$$
\begin{array}{rcl}
\Psi_0\, v & = & v \;:\; \tau_0 \\
 & & \qquad \text{where } \tau_0 = \Lambda. \\
\Psi_{n+1}\, v & = & \overline{\lambda} t.\overline{\lambda}\kappa.(v \,\underline{@}\, t)\,\underline{@}\,(\underline{\lambda} v'.\kappa\,\overline{@}\,(\Psi_n\, v')) \\
 & : & \tau_{n+1} \\
 & & \text{where } \tau_{n+1} = \Lambda \rightarrow (\tau_n \rightarrow \Lambda) \rightarrow \Lambda. \\
[\![\cdot]\!]^n & : & \Lambda \rightarrow (\tau_n \rightarrow \Lambda) \rightarrow \Lambda \\
[\![x]\!]^n & = & \overline{\lambda}\kappa.\kappa\,\overline{@}\,(\Psi_n\, x) \\
[\![\lambda x.e]\!]^0 & = & \overline{\lambda}\kappa.\kappa\,\overline{@}\,(\underline{\lambda} x.\underline{\lambda} k.[\![e]\!]^0\,\overline{@}\,(\overline{\lambda} t.k\,\underline{@}\,t)) \\
[\![\lambda x.e]\!]^{n+1} & = & \overline{\lambda}\kappa.\kappa\,\overline{@}\,(\overline{\lambda} t.\overline{\lambda}\kappa'.(\underline{\lambda} x.[\![e]\!]^n\,\overline{@}\,\kappa')\,\underline{@}\,t) \\
[\![e_0\, e_1]\!]^n & = & \overline{\lambda}\kappa.[\![e_0]\!]^{n+1}\,\overline{@}\,(\overline{\lambda} t_0.[\![e_1]\!]^0\,\overline{@}\,(\overline{\lambda} t_1.(t_0\,\overline{@}\,t_1)\,\overline{@}\,\kappa))
\end{array}
$$

Figure 1: A family of one-pass, call-by-value CPS transformations à la Plotkin

4 More compact CPS transformations in one pass

Promoting functions into continuations compromises context independence in the CPS transformation, since how to CPS-transform a λ-abstraction depends on whether it occurs in a β-redex or not. Fortunately, it does so in a very regular way, which makes it possible to derive a *family* of one-pass CPS transformations indexed by positions in the current context.



Indexing the transformation functions with the lexical position of their argument yields the one-pass CPS transformation à la Plotkin (i.e., with continuations last) of Figure 1. A λ-term $e : \Lambda$ is CPS-transformed with

$$\underline{\lambda} k.[\![e]\!]^0\,\overline{@}\,(\overline{\lambda} t.k\,\underline{@}\,t).$$

Similarly, a one-pass CPS transformation à la Fischer (i.e., with continuations first) is displayed in Figure 2.

$$
\begin{array}{rcl}
\Phi_0\, v & = & v \;:\; \tau_0 \\
& & \qquad \text{where } \tau_0 = \Lambda. \\
\Phi_{n+1}\, v & = & \overline{\lambda}\kappa.v \,\underline{@}\, (\underline{\lambda} v'.\kappa \,\overline{@}\, (\Phi_n\, v')) \\
& : & \tau_{n+1} \\
& & \text{where } \tau_{n+1} = (\tau_n \to \Lambda) \to \Lambda. \\
[\![\cdot]\!]^n & : & \Lambda \to (\tau_n \to \Lambda) \to \Lambda \\
[\![x]\!]^n & = & \overline{\lambda}\kappa.\kappa \,\overline{@}\, (\Phi_n\, x) \\
[\![\lambda x.e]\!]^0 & = & \overline{\lambda}\kappa.\kappa \,\overline{@}\, (\underline{\lambda} k.\underline{\lambda} x.[\![e]\!]^0 \,\overline{@}\, (\overline{\lambda} t.k \,\underline{@}\, t)) \\
[\![\lambda x.e]\!]^{n+1} & = & \overline{\lambda}\kappa.\kappa \,\overline{@}\, (\overline{\lambda}\kappa'.\underline{\lambda} x.[\![e]\!]^n \,\overline{@}\, \kappa') \\
[\![e_0\, e_1]\!]^n & = & \overline{\lambda}\kappa.[\![e_0]\!]^{n+1} \,\overline{@}\, (\overline{\lambda} t_0.[\![e_1]\!]^0 \,\overline{@}\, (\overline{\lambda} t_1.(t_0 \,\overline{@}\, \kappa) \,\underline{@}\, t_1))
\end{array}
$$

Figure 2: A family of one-pass, call-by-value CPS transformations à la Fischer

$[\![\cdot]\!]^0$ is applied to the root of a term (i.e., to the body of a λ-abstraction or to the expression in position of argument in an application). For $n > 0$, $[\![\cdot]\!]^n$ is applied to an expression in position of function in an application; n is the depth of the expression since the closest root, as in the picture above. The Ψ (resp. Φ) function coerces a syntactic object into a translation-time one.

The transformation based on these families of functions can be proven correct by a simulation theorem similar to Plotkin's [19]. The correctness criterion is a relation between the transformation of the result of an expression and the result of the transformation of it, i.e., (noting contextual equivalence with $\sim$)

$$e \longrightarrow^* v \quad \text{implies} \quad [\![e]\!]^0\, \lambda a.a \longrightarrow^* v' \quad \text{and} \quad v' \sim [\![v]\!]^0\, \lambda a.a$$

as well as preservation of non-termination and of getting stuck [7, 17].

Reflecting the context dependence of both CPS transformations, the two-level specifications in Figures 1 and 2 are not themselves simply typed. Instead, they are dependently typed and define two families of simply typed two-level specifications. Each of these families produces simply-typed two-level λ-terms, that can be statically (i.e., administratively) reduced in one pass. The ML signature of the first elements of each family are displayed in Figure 3, page 9. (If one uses Scheme, one can simply treat the indices as arguments.)

5 Conclusion and issues

In their study of CPS programs [22], Sabry and Felleisen needed a CPS transformation that would perform more administrative reductions than the ones already available [1, 5, 10, 27]. We have identified the extra power of this CPS transformation: a context-sensitive administrative reduction enabling a more effective treatment of β-redexes which corresponds to Bloo, Kamareddine, and Nederpelt's notion of generalized reduction. This treatment turns out to be independent of the relative positions of values and continuations. The resulting three-pass CPS transformation can be stated as a two-pass process involving (1) a first-order uncurrying phase and (2) a one-pass CPS transformation with context-insensitive administrative reductions. We have also presented two one-pass CPS transformations embodying the extra compaction and generalizing the corresponding one-pass CPS transformations à la Plotkin and à la Fischer. They can be adapted *mutatis mutandis* for encoding λ-terms into monadic normal form [4, 13, 16] (sometimes called A-normal form [9, Figure 9]), including β_{lift}.

Acknowledgements: The first author is grateful to Matthias Felleisen, Andrzej Filinski, John Hatcliff, and Amr Sabry for discussions and comments on this topic and these transformations in June and July 1993, at CMU. Kristoffer Rose wanted to see the dependent types of Figures 1 and 2 spelled out. Thanks are also due to the CW'01 and IPL reviewers and to Julia Lawall and Olin Shivers for perceptive comments.

This work is partially supported by the ESPRIT Working Group APPSEM II (`http://www.appsem.org`) and by the Danish Natural Science Research Council, Grant no. 21-03-0545.

References

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

[2] Roel Bloo, Fairouz Kamareddine, and Rob Nederpelt. The Barendregt cube with definitions and generalised reduction. *Information and Computation*, 126(2):123–143, 1996.

[3] Daniel Damian and Olivier Danvy. Syntactic accidents in program analysis: On the impact of the CPS transformation. *Journal of Functional Programming*, 13(5):867–904, 2003.

[4] Olivier Danvy. A new one-pass transformation into monadic normal form. In Görel Hedin, editor, *Compiler Construction, 12th International Conference, CC 2003*, number 2622 in Lecture Notes in Computer Science, pages 77–89, Warsaw, Poland, April 2003. Springer-Verlag.

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

[6] Olivier Danvy and Lasse R. Nielsen. CPS transformation of beta-redexes. In Amr Sabry, editor, *Proceedings of the Third ACM SIGPLAN Workshop on Continuations*, Technical report 545, Computer Science Department, Indiana University, pages 35–39, London, England, January 2001. Accepted for publication in Information Processing Letters (2004).

[7] Olivier Danvy and Lasse R. Nielsen. A higher-order colon translation. In Herbert Kuchen and Kazunori Ueda, editors, *Functional and Logic Programming, 5th International Symposium, FLOPS 2001*, number 2024 in Lecture Notes in Computer Science, pages 78–91, Tokyo, Japan, March 2001. Springer-Verlag.

```
signature CPST_PLOTKIN =
sig
  val P_0 : exp -> env -> (exp -> exp) -> exp
  val P_1 : exp -> env -> ((exp -> (exp -> exp) -> exp) -> exp) -> exp
  val P_2 : exp -> env -> ((exp -> ((exp -> (exp -> exp) -> exp) -> exp) -> exp) -> exp) -> exp
  val P_3 : exp -> env -> ((exp -> ((exp -> ((exp -> (exp -> exp) -> exp) -> exp) -> exp) -> exp) -> exp) -> exp) -> exp
  val Psi_0 : exp -> exp
  val Psi_1 : exp -> exp -> (exp -> exp) -> exp
  val Psi_2 : exp -> exp -> ((exp -> (exp -> exp) -> exp) -> exp) -> exp
  exception CPS_Overflow of int
  val t_plotkin : exp -> exp
end

signature CPST_FISCHER =
sig
  val F_0 : exp -> env -> (exp -> exp) -> exp
  val F_1 : exp -> env -> (((exp -> exp) -> exp) -> exp) -> exp
  val F_2 : exp -> env -> (((((exp -> exp) -> exp) -> exp) -> exp) -> exp) -> exp
  val F_3 : exp -> env -> (((((((exp -> exp) -> exp) -> exp) -> exp) -> exp) -> exp) -> exp) -> exp
  val Phi_0 : exp -> exp
  val Phi_1 : exp -> (exp -> exp) -> exp
  val Phi_2 : exp -> (((exp -> exp) -> exp) -> exp) -> exp
  exception CPS_Overflow of int
  val t_fischer : exp -> exp
end
```

Figure 3: Signatures of the ML implementations of Figures 1 (Plotkin) and 2 (Fischer)

[8] Michael J. Fischer. Lambda-calculus schemata. *LISP and Symbolic Computation*, 6(3/4):259–288, 1993. `<http://www.brics.dk/~hosc/vol06/03-fischer.html>` Earlier version available in the proceedings of an ACM Conference on Proving Assertions about Programs, SIGPLAN Notices, Vol. 7, No. 1, January 1972.

[9] 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.

[10] Daniel P. Friedman, Mitchell Wand, and Christopher T. Haynes. *Essentials of Programming Languages*. The MIT Press and McGraw-Hill, 1991.

[11] John Hannan and Patrick Hicks. Higher-order unCurrying. *Higher-Order and Symbolic Computation*, 13(3):179–218, 2000.

[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] 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. ACM Press.

[15] Julia L. Lawall and Olivier Danvy. Continuation-based partial evaluation. In Carolyn L. Talcott, editor, *Proceedings of the 1994 ACM Conference on Lisp and Functional Programming*, LISP Pointers, Vol. VII, No. 3, pages 227–238, Orlando, Florida, June 1994. ACM Press.

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

[17] Lasse R. Nielsen. *A study of defunctionalization and continuation-passing style*. PhD thesis, BRICS PhD School, University of Aarhus, Aarhus, Denmark, July 2001. BRICS DS-01-7.

[18] Flemming Nielson and Hanne Riis Nielson. *Two-Level Functional Languages*, volume 34 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.

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

[20] John C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6(3/4):233–247, 1993.

[21] 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 94-242.

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

[23] Olin Shivers. *Control-Flow Analysis of Higher-Order Languages or Taming Lambda*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, May 1991. Technical Report CMU-CS-91-145.

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

[25] Hayo Thielecke. *Categorical Structure of Continuation Passing Style*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, 1997. ECS-LFCS-97-376.

[26] Christopher P. Wadsworth. Continuations revisited. *Higher-Order and Symbolic Computation*, 13(1/2):131–133, 2000.

[27] 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.

Recent BRICS Report Series Publications

RS-04-39 Olivier Danvy and Lasse R. Nielsen. *CPS Transformation of Beta-Redexes*. December 2004. ii+11 pp. Superseedes an article to appear in *Information Processing Letters* and BRICS report RS-00-35.

RS-04-38 Olin Shivers and Mitchell Wand. *Bottom-Up β-Substitution: Uplinks and λ-DAGs*. December 2004.

RS-04-37 Jørgen Iversen and Peter D. Mosses. *Constructive Action Semantics for Core ML*. December 2004. 68 pp. To appear in a special *Language Definitions and Tool Generation* issue of the journal *IEE Proceedings Software*.

RS-04-36 Mark van den Brand, Jørgen Iversen, and Peter D. Mosses. *An Action Environment*. December 2004. 27 pp. Appears in Hedin and Van Wyk, editors, *Fourth ACM SIGPLAN Workshop on Language Descriptions, Tools and Applications*, LDTA '04, 2004, pages 149–168.

RS-04-35 Jørgen Iversen. *Type Checking Semantic Functions in ASDF*. December 2004.

RS-04-34 Anders Møller and Michael I. Schwartzbach. *The Design Space of Type Checkers for XML Transformation Languages*. December 2004. 21 pp. Appears in Eiter and Libkin, editors, *Database Theory: 10th International Conference*, ICDT '05 Proceedings, LNCS 3363, 2005, pages 17–36.

RS-04-33 Aske Simon Christensen, Christian Kirkegaard, and Anders Møller. *A Runtime System for XML Transformations in Java*. December 2004. 15 pp. Appears in Bellahsene, Milo, Rys, Suciu and Unland, editors, *Database and XML Technologies: Second International XML Database Symposium*, XSym '04 Proceedings, LNCS 3186, 2004, pages 143–157. Supersedes the earlier BRICS report RS-03-29.

RS-04-32 Philipp Gerhardy. *A Quantitative Version of Kirk's Fixed Point Theorem for Asymptotic Contractions*. December 2004. 9 pp.

RS-04-31 Philipp Gerhardy and Ulrich Kohlenbach. *Strongly Uniform Bounds from Semi-Constructive Proofs*. December 2004. 31 pp.