Polymorphic Type Assignment and CPS Conversion

(To appear: ACM SIGPLAN Workshop on Continuations, San Francisco, June 1992)

Robert Harper* Mark Lillibridge†
Carnegie Mellon University
Pittsburgh, PA 15213

April 10, 1992

Abstract

Meyer and Wand established that the type of a term in the simply typed λ-calculus may be related in a straightforward manner to the type of its call-by-value CPS transform. This typing property may be extended to Scheme-like continuation-passing primitives, from which the soundness of these extensions follows. We study the extension of these results to the Damas-Milner polymorphic type assignment system under both the call-by-value and call-by-name interpretations. We obtain CPS transforms for the call-by-value interpretation, provided that the polymorphic let is restricted to values, and for the call-by-name interpretation with no restrictions. We prove that there is no call-by-value CPS transform for the full Damas-Milner language that validates the Meyer-Wand typing property and is equivalent to the standard call-by-value transform up to $\beta\eta$-conversion.

1 Introduction

In their study of the relationship between direct and continuation semantics for the simply typed λ-calculus ($\lambda^{\rightarrow}$), Meyer and Wand note that the type of a term in $\lambda^{\rightarrow}$ may be related in a simple and natural way to the type of its call-by-value continuation passing style (CPS) transform [8]. This result may be extended to the calculus that results from extending $\lambda^{\rightarrow}$ with Scheme-like continuation-passing primitives callcc and throw ($\lambda^{\rightarrow}$ + cont) [1, 3]. Since $\lambda^{\rightarrow}$ under a call-by-value operational semantics is "type safe" in the sense of Milner [9, 2], and since the call-by-value CPS transform faithfully mimics the call-by-value semantics [12], it follows that $\lambda^{\rightarrow}$ + cont under a call-by-value operational semantics is also type safe.

In a subsequent study Duba, Harper, and MacQueen studied the addition of callcc and throw to Standard ML [10]. The extension of the Meyer-Wand transform to $\lambda^{\rightarrow}$ + cont establishes the soundness of the monomorphic fragment of the language, but the soundness of the polymorphic language with continuation-passing primitives was left open. It was subsequently proved by the authors [7] that the full polymorphic language is unsound when extended with callcc and throw. The source of this discrepancy may be traced to the interaction between the polymorphic let construct and the typing rules for callcc. Several *ad hoc* methods for restricting the language to recover soundness have been proposed [6, 14].

In this paper we undertake a systematic study of the interaction between continuations and polymorphism by considering the typing properties of the CPS transform for both the call-by-value and call-by-name variants of the Damas-Milner language [2] and its extension with continuation-passing primitives. We obtain suitable extensions of the Meyer-Wand theorem for the call-by-value CPS transform, provided that the polymorphic let is restricted to values, and for the call-by-name transform, under no restrictions. Finally, we

*This work was sponsored by the Defense Advanced Research Projects Agency, CSTO, under the title "The Fox Project: Advanced Development of Systems Software", ARPA Order No. 8313, issued by ESD/AVS under Contract No. F19628–91–C–0168.

†Supported by a National Science Foundation Graduate Fellowship.

prove that there is no call-by-value CPS transform for the full Damas-Milner language that both satisfies the Meyer-Wand typing property and is equivalent to the usual transform up to $\beta\eta$-conversion. In particular, the standard call-by-value CPS transform fails to preserve typability.

2 Untyped Terms

The language of untyped terms is given by the following grammar:

$$e \quad ::= \quad x \mid \lambda x.e \mid e_1\, e_2 \mid \mathsf{let}\, x\, \mathsf{be}\, e_1\, \mathsf{in}\, e_2 \mid \mathsf{callcc} \mid \mathsf{throw}$$

Here x ranges over a countably infinite set of variables. We include the let construct as a primitive because it is needed in the discussion of polymorphic type assignment. callcc and throw are continuation-passing primitives whose definitions are derived from analogous constructs in Scheme [1] and Standard ML of New Jersey [3].

We consider two CPS transforms for untyped terms, corresponding to the call-by-value and call-by-name operational semantics [12]. Each CPS transform consists of a transformation $|{-}|$ for untyped terms and a transformation $||{-}||$ for untyped values. Exactly what is considered a value depends on which operational semantics is being used. Under call-by-value, variables, λ-abstractions, and constants[1] are considered values. Under call-by-name, only λ-abstractions and constants are considered values. We shall use v as a meta-variable for call-by-value values and w as a meta-variable for call-by-name values.

Definition 2.1 (Call-by-Value CPS Transform)

$$\begin{aligned}
|v|_{cbv} &= \lambda k.k\, ||v||_{cbv} \\
|e_1\, e_2|_{cbv} &= \lambda k.|e_1|_{cbv}\, (\lambda x_1.|e_2|_{cbv}\, (\lambda x_2.x_1\, x_2\, k)) \\
|\mathsf{let}\, x\, \mathsf{be}\, e_1\, \mathsf{in}\, e_2|_{cbv} &= \lambda k.|e_1|_{cbv}\, (\lambda x.|e_2|_{cbv}\, k)
\end{aligned}$$

$$\begin{aligned}
||x||_{cbv} &= x \\
||\lambda x.e||_{cbv} &= \lambda x.|e|_{cbv} \\
||\mathsf{callcc}||_{cbv} &= \lambda f.\lambda k.f\, k\, k \\
||\mathsf{throw}||_{cbv} &= \lambda c.\lambda k.k\, (\lambda x.\lambda l.c\, x)
\end{aligned}$$

Lemma 2.2

1. $||\,[v/x]v'\,||_{cbv} = [\,||v||_{cbv}/x\,]\,||v'||_{cbv}$.
2. $|\,[v/x]e\,|_{cbv} = [\,||v||_{cbv}/x\,]\,|e|_{cbv}$.

We shall also have need of a variant call-by-value CPS transform (cbv') defined on untyped terms satisfying the restriction that all let expressions are of the form $\mathsf{let}\, x\, \mathsf{be}\, v\, \mathsf{in}\, e$. I.e., the let-bound expression is required to be a (call-by-value) value. Because of this restriction, a simpler rule can be given for the let case:

$$|\mathsf{let}\, x\, \mathsf{be}\, v\, \mathsf{in}\, e|_{cbv'} \quad = \quad \lambda k.\mathsf{let}\, x\, \mathsf{be}\, ||v||_{cbv'}\, \mathsf{in}\, (|e|_{cbv'}\, k)$$

This simpler rule for let expressions is the only difference between the two transforms.

Lemma 2.3 *Let v and v' be values obeying the restriction on* let *expressions and e be a term obeying the restriction on* let *expressions. Then*

1. $||\,[v/x]v'\,||_{cbv'} = [\,||v||_{cbv'}/x\,]\,||v'||_{cbv'}$.
2. $|\,[v/x]e\,|_{cbv'} = [\,||v||_{cbv'}/x\,]\,|e|_{cbv'}$.

[1] Note that callcc and throw are considered to be constants.

Definition 2.4 (Call-by-Name CPS Transform)

$$\begin{array}{rcl}
|w|_{cbn} & = & \lambda k.k\,||w||_{cbn} \\
|x|_{cbn} & = & x \\
|e_1\,e_2|_{cbn} & = & \lambda k.|e_1|_{cbn}\,(\lambda x_1.x_1\,|e_2|_{cbn}\,k) \\
|\mathsf{let}\ x\ \mathsf{be}\ e_1\ \mathsf{in}\ e_2|_{cbn} & = & \lambda k.\mathsf{let}\ x\ \mathsf{be}\ |e_1|_{cbn}\ \mathsf{in}\ (|e_2|_{cbn}\,k) \\
\\
||\lambda x.e||_{cbn} & = & \lambda x.|e|_{cbn} \\
||\mathsf{callcc}||_{cbn} & = & \lambda f.\lambda k.f\,(\lambda f'.f'\,(\lambda l.l\,k)\,k) \\
||\mathsf{throw}||_{cbn} & = & \lambda c.\lambda k.k\,(\lambda x.\lambda l.c\,(\lambda c'.x\,(\lambda x'.c'\,x')))
\end{array}$$

Lemma 2.5

1. $||\,[e/x]w\,||_{cbn} = [\,|e|_{cbn}/x\,]\,||w||_{cbn}$.
2. $|\,[e/x]e'\,|_{cbn} = [\,|e|_{cbn}/x\,]\,|e'|_{cbn}$.

The correctness of these transforms may either be established by relating them to an independently-defined operational semantics (as in [12, 4]), or else taken as the definition of call-by-value and call-by-name semantics.

3 Simple Type Assignment

In this section we review Meyer and Wand's typing theorem for the call-by-value CPS transform for the simply-typed λ-calculus ($\lambda^{\rightarrow}$), and present an analogous result for the call-by-name CPS transform.

Definition 3.1 ($\lambda^{\rightarrow}$ Types and Contexts)

$$\begin{array}{lccl}
\textit{types} & \tau & ::= & b \mid \tau_1 \rightarrow \tau_2 \\
\\
\textit{contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}\tau
\end{array}$$

Here b ranges over a countable set of base types. We assume that among the base types there is a distinguished type α, which will be used in what follows to represent the "answer" type of a CPS transform.

Definition 3.2 ($\lambda^{\rightarrow}$ Typing Rules)

$$\Gamma \triangleright x : \Gamma(x) \qquad (\textsc{var})$$

$$\frac{\Gamma, x{:}\tau_1 \triangleright e : \tau_2}{\Gamma \triangleright \lambda x.e : \tau_1 \rightarrow \tau_2}\quad (x \notin \mathrm{dom}(\Gamma)) \qquad (\textsc{abs})$$

$$\frac{\Gamma \triangleright e_1 : \tau_2 \rightarrow \tau \quad \Gamma \triangleright e_2 : \tau_2}{\Gamma \triangleright e_1\,e_2 : \tau} \qquad (\textsc{app})$$

$$\frac{\Gamma \triangleright e_1 : \tau_1 \qquad \Gamma, x{:}\tau_1 \triangleright e_2 : \tau}{\Gamma \triangleright \mathsf{let}\ x\ \mathsf{be}\ e_1\ \mathsf{in}\ e_2 : \tau} \qquad (\textsc{mono-let})$$

The type system $\lambda^{\rightarrow} + \mathsf{cont}$ is defined by adding the type expression $\tau\,\mathsf{cont}$ and the following typing rules for the continuation-passing primitives:

$$\Gamma \triangleright \mathsf{callcc} : (\tau\ \mathsf{cont} \rightarrow \tau) \rightarrow \tau \qquad (\textsc{callcc})$$

$$\Gamma \triangleright \mathsf{throw} : \tau\ \mathsf{cont} \rightarrow \tau \rightarrow \tau' \qquad (\textsc{throw})$$

Definition 3.3 (Call-by-Value Type Transform for $\lambda^{\rightarrow}$)

$$
\begin{array}{rcl}
|\tau|_{cbv} & = & (||\tau||_{cbv} \rightarrow \alpha) \rightarrow \alpha \\
\\
||b||_{cbv} & = & b \\
||\tau_1 \rightarrow \tau_2||_{cbv} & = & ||\tau_1||_{cbv} \rightarrow |\tau_2|_{cbv}
\end{array}
$$

The type transform is extended to contexts by defining $||\Gamma||_{cbv}(x) = ||\Gamma(x)||_{cbv}$ for each $x \in \mathrm{dom}(\Gamma)$.

Theorem 3.4 (Meyer-Wand)

1. *If* $\lambda^{\rightarrow} \vdash \Gamma \rhd v : \tau$*, then* $\lambda^{\rightarrow} \vdash ||\Gamma||_{cbv} \rhd ||v||_{cbv} : ||\tau||_{cbv}$.
2. *If* $\lambda^{\rightarrow} \vdash \Gamma \rhd e : \tau$*, then* $\lambda^{\rightarrow} \vdash ||\Gamma||_{cbv} \rhd |e|_{cbv} : |\tau|_{cbv}$.

The call-by-value type transform is extended to $\lambda^{\rightarrow} + \mathsf{cont}$ by defining $||\tau\,\mathsf{cont}||_{cbv} = ||\tau||_{cbv} \rightarrow \alpha$. It is straightforward to verify that Theorem 3.4 extends to $\lambda^{\rightarrow} + \mathsf{cont}$ in this way [3].

Definition 3.5 (Call-by-Name Type Transform for $\lambda^{\rightarrow}$) [2]

$$
\begin{array}{rcl}
|\tau|_{cbn} & = & (||\tau||_{cbn} \rightarrow \alpha) \rightarrow \alpha \\
\\
||b||_{cbn} & = & b \\
||\tau_1 \rightarrow \tau_2||_{cbn} & = & |\tau_1|_{cbn} \rightarrow |\tau_2|_{cbn}
\end{array}
$$

The type transform is extended to contexts by defining $|\Gamma|_{cbn}(x) = |\Gamma(x)|_{cbn}$ for each $x \in \mathrm{dom}(\Gamma)$.

Theorem 3.6

1. *If* $\lambda^{\rightarrow} \vdash \Gamma \rhd w : \tau$*, then* $\lambda^{\rightarrow} \vdash |\Gamma|_{cbn} \rhd ||w||_{cbn} : ||\tau||_{cbn}$.
2. *If* $\lambda^{\rightarrow} \vdash \Gamma \rhd e : \tau$*, then* $\lambda^{\rightarrow} \vdash |\Gamma|_{cbn} \rhd |e|_{cbn} : |\tau|_{cbn}$.

The call-by-name CPS transform is extended to $\lambda^{\rightarrow} + \mathsf{cont}$ by defining $||\tau\,\mathsf{cont}||_{cbn} = ||\tau||_{cbn} \rightarrow \alpha$, just as for call-by-value. It is straightforward to verify that Theorem 3.6 extends to $\lambda^{\rightarrow} + \mathsf{cont}$ in this way.

4 Polymorphic Type Assignment

In this section we study the extension of the Meyer-Wand typing property to Damas and Milner's polymorphic type assignment system (DM).

The syntax of types and contexts in (DM) is defined by the following grammar:

Definition 4.1 (DM Types and Contexts)

$$
\begin{array}{llcl}
\textit{monotypes} & \tau & ::= & t \mid b \mid \tau_1 \rightarrow \tau_2 \\
\textit{polytypes} & \sigma & ::= & \tau \mid \forall t.\sigma \\
\\
\textit{contexts} & \Gamma & ::= & \cdot \mid \Gamma, x{:}\sigma
\end{array}
$$

Here t ranges over a countably infinite set of type variables. The typing rules of the Damas-Milner system extend those of $\lambda^{\rightarrow}$ as follows:

[2] The term "call-by-name type transform" is something of a misnomer since there exists a by-value CPS transform that validates the by-name typing property [5]. Nevertheless we stick with the suggestive, if somewhat misleading, terminology.

Definition 4.2 (Additional DM Typing Rules)

$$\frac{\Gamma \rhd e : \sigma}{\Gamma \rhd e : \forall t.\sigma} \quad (t \notin FTV(\Gamma)) \tag{GEN}$$

$$\frac{\Gamma \rhd e : \forall t.\sigma}{\Gamma \rhd e : [\tau/t]\sigma} \tag{INST}$$

$$\frac{\Gamma \rhd e_1 : \sigma_1 \quad \Gamma, x{:}\sigma_1 \rhd e_2 : \tau_2}{\Gamma \rhd \mathsf{let}\ x\ \mathsf{be}\ e_1\ \mathsf{in}\ e_2 : \tau_2} \quad (x \notin \mathrm{dom}(\Gamma)) \tag{POLY-LET}$$

The system $\mathsf{DM} + \mathsf{cont}$ is defined by adding the type expression $\tau\,\mathsf{cont}$, as before, and adding the following typing rules:

$$\Gamma \rhd \mathsf{callcc} : \forall t.(t\,\mathsf{cont} \to t) \to t \tag{CALLCC'}$$

$$\Gamma \rhd \mathsf{throw} : \forall s.\forall t.s\,\mathsf{cont} \to s \to t \tag{THROW'}$$

Let σ_{callcc} and σ_{throw} be the polytypes assigned to callcc and throw, respectively.

4.1 Restricted Call-by-Value

Let DM^- denote the sub-system of DM obtained by restricting let expressions so that the bound expression is a call-by-value value. The Meyer-Wand typing theorem may be extended to terms of DM^-, provided that we use the variant call-by-value CPS transform (cbv') given in Section 2.

Definition 4.3 (Call-by-Value Type Transform for DM^-)

$$\begin{array}{rcl}
|\tau|_{cbv} & = & (||\tau||_{cbv} \to \alpha) \to \alpha \\
|\forall t.\sigma|_{cbv} & = & \forall t.|\sigma|_{cbv} \\
\\
||t||_{cbv} & = & t \\
||b||_{cbv} & = & b \\
||\tau_1 \to \tau_2||_{cbv} & = & ||\tau_1||_{cbv} \to |\tau_2|_{cbv} \\
||\forall t.\sigma||_{cbv} & = & \forall t.||\sigma||_{cbv}
\end{array}$$

This definition extends the Meyer-Wand type transform to polymorphic types. In the terminology of Reynolds [13], polymorphic instantiation is given a "trivial" interpretation in that no interesting computation can occur as a result of the specialization of a value of polymorphic type. The definition of $|\forall t.\sigma|_{cbv}$ reflects the fact that in DM^- there is no need of continuations whose domain is a polymorphic type.

Lemma 4.4

1. *$||[\tau/t]\sigma||_{cbv} = [\,||\tau||_{cbv}\,/\,t\,]\,||\sigma||_{cbv}$.*
2. *$|[\tau/t]\sigma|_{cbv} = [\,||\tau||_{cbv}\,/\,t\,]\,|\sigma|_{cbv}$.*

Theorem 4.5

1. *If $\mathsf{DM}^- \vdash \Gamma \rhd v : \sigma$, then $\mathsf{DM}^- \vdash ||\Gamma||_{cbv'} \rhd ||v||_{cbv'} : ||\sigma||_{cbv}$.*
2. *If $\mathsf{DM}^- \vdash \Gamma \rhd e : \sigma$, then $\mathsf{DM}^- \vdash ||\Gamma||_{cbv'} \rhd |e|_{cbv'} : |\sigma|_{cbv}$.*

The proof hinges on the following observations. First, the definitions of the transformations $|-|_{cbv}$ and $||-||_{cbv}$ on polytypes are such that the GEN and INST rules carry over to applications of the same rule. Specifically, if $\Gamma \rhd e : \sigma$ and t does not occur free in Γ, then t does not occur free in $||\Gamma||_{cbv'}$, and hence $||\Gamma||_{cbv'} \rhd |e|_{cbv'} : \forall t.|\sigma|_{cbv}$ is derivable by an application of GEN and the induction hypothesis. A similar argument suffices for the value transform. Uses of INST are handled similarly.

Second, the restriction on let expressions in DM^- combined with the use of the variant transform ensure that let's are carried over to let's, and hence that polymorphic typing is preserved. Specifically, if $\Gamma \triangleright v_1 : \sigma_1$ and $\Gamma, x{:}\sigma_1 \triangleright e_2 : \tau_2$ are both derivable, then by induction $||\Gamma||_{cbv'} \triangleright ||v_1||_{cbv'} : ||\sigma_1||_{cbv}$ and $||\Gamma||_{cbv'}, x{:}||\sigma_1||_{cbv} \triangleright |e_2|_{cbv'} : |\tau_2|_{cbv}$ are derivable, and hence $||\Gamma||_{cbv'} \triangleright \lambda k.\mathsf{let}\, x\, \mathsf{be}\, ||v_1||_{cbv'}\, \mathsf{in}\, |e_2|_{cbv'}\, k : |\tau_2|_{cbv}$ is also derivable.

Theorem 4.5 extends to $\mathsf{DM}^- + \mathsf{cont}$ by defining $||\tau\, \mathsf{cont}||_{cbv} = ||\tau||_{cbv} \rightarrow \alpha$. We need only verify that $||\mathsf{callcc}||_{cbv'}$ and $||\mathsf{throw}||_{cbv'}$, given in Section 2, have types $||\sigma_{\mathsf{callcc}}||_{cbv}$ and $||\sigma_{\mathsf{throw}}||_{cbv}$, respectively. The soundness of $\mathsf{DM}^- + \mathsf{cont}$ under call-by-value follows from the extended theorem. (Same proof as for the soundness of $\lambda^{\rightarrow} + \mathsf{cont}$ under call-by-value.)

4.2 Call-by-Name

Theorem 3.6 (the Meyer-Wand-like typing theorem for call-by-name) can be extended to the unrestricted DM langauge.

Definition 4.6 (Call-by-Name Type Transform for DM)

$$\begin{aligned} |\tau|_{cbn} &= (||\tau||_{cbn} \rightarrow \alpha) \rightarrow \alpha \\ |\forall t.\sigma|_{cbn} &= \forall t.|\sigma|_{cbn} \\ \\ ||t||_{cbn} &= t \\ ||b||_{cbn} &= b \\ ||\tau_1 \rightarrow \tau_2||_{cbn} &= |\tau_1|_{cbn} \rightarrow |\tau_2|_{cbn} \\ ||\forall t.\sigma||_{cbn} &= \forall t.||\sigma||_{cbn} \end{aligned}$$

Lemma 4.7

1. $||\,[\tau/t]\sigma\,||_{cbn} = [\,||\tau||_{cbn}\,/\,t\,]\,||\sigma||_{cbn}$.
2. $|\,[\tau/t]\sigma\,|_{cbn} = [\,||\tau||_{cbn}\,/\,t\,]\,|\sigma|_{cbn}$.

Theorem 4.8

1. *If* $\mathsf{DM} \vdash \Gamma \triangleright w : \sigma$, *then* $\mathsf{DM} \vdash |\Gamma|_{cbn} \triangleright ||w||_{cbn} : ||\sigma||_{cbn}$.
2. *If* $\mathsf{DM} \vdash \Gamma \triangleright e : \sigma$, *then* $\mathsf{DM} \vdash |\Gamma|_{cbn} \triangleright |e|_{cbn} : |\sigma|_{cbn}$.

The proof proceeds along similar lines to that of the call-by-value case. For example, if $\Gamma \triangleright e_1 : \sigma_1$ and $\Gamma, x{:}\sigma_1 \triangleright e_2 : \tau_2$ are derivable, then by induction so are $|\Gamma|_{cbn} \triangleright |e_1|_{cbn} : |\sigma_1|_{cbn}$ and $|\Gamma|_{cbn}, x{:}|\sigma_1|_{cbn} \triangleright |e_2|_{cbn} : |\tau_2|_{cbn}$, and hence so is $|\Gamma|_{cbn} \triangleright \lambda k.\mathsf{let}\, x\, \mathsf{be}\, |e_1|_{cbn}\, \mathsf{in}\, |e_2|_{cbn}\, k : |\tau_2|_{cbn}$, as required.

Theorem 4.8 extends to $\mathsf{DM} + \mathsf{cont}$ by defining $||\tau\, \mathsf{cont}||_{cbn} = ||\tau||_{cbn} \rightarrow \alpha$. We need only verify that $||\mathsf{callcc}||_{cbn}$ and $||\mathsf{throw}||_{cbn}$, given in Section 2, have types $||\sigma_{\mathsf{callcc}}||_{cbn}$ and $||\sigma_{\mathsf{throw}}||_{cbn}$, respectively. The soundness of $\mathsf{DM} + \mathsf{cont}$ under call-by-name operational semantics follows from the extended theorem in a manner similar to that of the call-by-value case for $\mathsf{DM}^- + \mathsf{cont}$.

4.3 Unrestricted Call-by-Value

Having established suitable typing properties for the variant call-by-value transform for DM^- and the call-by-name transform for full DM, it is natural to consider whether there is a call-by-value CPS transform for full DM that satisfies a Meyer-Wand-like typing property. Since cbv' is only defined on terms with restricted let expressions, we can not simply extend Theorem 4.5 to full DM.

Let us consider attempting to extend Theorem 4.5 to full DM by using cbv instead of cbv' as the transform. Consider the induction step for the polymorphic let case. By induction we have

$$\mathsf{DM} \vdash ||\Gamma||_{cbv} \triangleright |e_1|_{cbv} : |\sigma_1|_{cbv}$$

and

$$\mathsf{DM} \vdash ||\Gamma||_{cbv}, x{:}||\sigma_1||_{cbv} \triangleright |e_2|_{cbv} : |\tau_2|_{cbv}.$$

We are to show that

$$\mathsf{DM} \vdash ||\Gamma||_{cbv} \rhd \lambda k.|e_1|_{cbv}\,(\lambda x.|e_2|_{cbv}\,k) : |\tau_2|_{cbv}.$$

Since the call-by-value interpretation of let requires that e_1 be evaluated before e_2, the call-by-value CPS transform of let x be e_1 in e_2 involves a continuation whose argument may, in general, be of polymorphic type. To capture this we must change the definition of $|-|_{cbv}$ so that $|\sigma|_{cbv} = (||\sigma||_{cbv} \to \alpha) \to \alpha$. But this takes us beyond the limits of the Damas-Milner type system since $||\sigma||_{cbv}$ is, in general, a polytype. We therefore consider as target language the extension, DM^+, of DM, in which the distinction between monotypes and polytypes is dropped, leading to full polymorphic type assignment [11]. The decidability of type checking for DM^+ is unknown, but this is not important for our purposes. We shall rely, however, on the fact that the subject reduction property holds for β-reduction in DM^+ [11].

With these changes to the type transformation and the associated enrichment of the target type system, the induction step for general let's works. However, polymorphic generalization becomes problematic. Specifically, if $\mathsf{DM} \vdash \Gamma \rhd e : \sigma$ with $t \notin FTV(\Gamma)$, then by induction $\mathsf{DM}^+ \vdash ||\Gamma||_{cbv} \rhd |e|_{cbv} : |\sigma|_{cbv}$, and $t \notin FTV(||\Gamma||_{cbv})$. We are to show $\mathsf{DM}^+ \vdash ||\Gamma||_{cbv} \rhd |e| : |\forall t.\sigma|_{cbv}$, and there is no evident way to proceed. We can indeed show that $|e|$ has type $\forall t.(||\sigma||_{cbv} \to \alpha) \to \alpha$, but this is not enough. In fact we shall prove that any variant call-by-value CPS transform $|e|$ verifying the Meyer-Wand typing property for DM must not be $\beta\eta$-convertible to $|e|_{cbv}$.

The argument proceeds by way of the extension of DM with continuation passing primitives. Under the call-by-value evaluation strategy, DM + cont is unsound. Specifically, we can find a term e such that e has a type τ, but whose value, when executed, fails to have type τ. In other words, evaluation fails to respect typing. Assuming that we have base types int and bool, and constants[3] $0 :$ int and true : bool, the following term is well-typed with type bool in DM + cont but evaluates under call-by-value to 0:

$$\begin{array}{rcl} e_0 & = & \mathsf{let}\ f\ \mathsf{be}\ \mathsf{callcc}\,(\lambda k.\lambda x.\mathsf{throw}\,k\,\lambda y.x) \\ & & \mathsf{in}\ (\lambda x.\lambda y.y)\,(f\,0)\,(f\,\mathsf{true}) \end{array}$$

Using the typing rules of DM + cont, the let-bound identifier f is assigned the type $\forall t.t \to t$, and hence may be used at types int→int and bool→bool in the body. But the binding for f grabs the continuation associated with the body of the let expression and saves it. Upon evaluation of $f\,0$, the continuation is invoked and f is effectively re-bound to a constant function returning 0. The body is re-entered, $f\,0$ is evaluated once again (without difficulty), but then f true is evaluated, resulting in 0.

It follows that there is no call-by-value CPS transform for DM + cont that preserves typability. Consequently, any call-by-value CPS transform for DM must be of a somewhat different form than the usual one.

Theorem 4.9 (No Call-by-Value CPS Transform) *There is no call-by-value CPS transform $|e|$ for* DM *that simultaneously satisfies the following two conditions:*

1. *Equivalence:* $|e| =_{\beta\eta} |e|_{cbv}$.
2. *Typing: If* $\mathsf{DM} \vdash \Gamma \rhd e : \sigma$, *then* $\mathsf{DM}^+ \vdash ||\Gamma||_{cbv} \rhd |e| : |\sigma|_{cbv}$.

Proof: Given such a transform we could form $|e_0|$ (where e_0 is given above) by regarding callcc and throw as variables of polytype σ_{callcc} and σ_{throw}, respectively. By the typing property this term has type $|\mathsf{bool}|_{cbv}$, under the assumption that callcc and throw have types $||\sigma_{\mathsf{callcc}}||_{cbv}$ and $||\sigma_{\mathsf{throw}}||_{cbv}$, respectively. Consequently the substitution instance $e_1 = [||\mathsf{callcc}||_{cbv}, ||\mathsf{throw}||_{cbv}/\mathsf{callcc}, \mathsf{throw}]\,|e_0|$ has type $|\mathsf{bool}|_{cbv} = (bool \to \alpha) \to \alpha$. But the corresponding substitution instance of $|e_0|_{cbv}$ is precisely the call-by-value CPS transform of e_0, taking account of callcc and throw directly. Since $\beta\eta$-conversion is preserved under substitution, we have by the equivalence property that e_1 is $\beta\eta$-convertible to $|e_0|_{cbv}$. Now, we know that $|e_0|_{cbv}\,\lambda x.x$ evaluates under call-by-value to 0. Consequently, this expression's $\beta\eta$ (and hence β) normal

[3] This argument can be made without constants but at the cost of increased complexity. Constants of base type can easily be added to any of the transforms presented in this paper by defining $||c|| = c$, c a constant. Constants of non-base type must be handled on a case-by-case basis.

form is 0. Therefore, we have that $e_1\,\lambda x.x$ is β-reducible to 0. But this is a violation of the subject reduction property of DM^+ [11] since $e_1\,\lambda x.x$ has type bool!

The conditions of Theorem 4.9 leave open the possibility of either finding a variant call-by-value transform that is not convertible to the standard one, or else varying the type transform in such a way that a Meyer-Wand-like typing property can be proved, or both. Any variant type transform must be such that either $||\mathsf{callcc}||_{cbv}$ or $||\mathsf{throw}||_{cbv}$ fail to have the required types under this transform so as to preclude extension to $\mathsf{DM} + \mathsf{cont}$. We know of no such variants, but have no evidence that none exist.

4.4 Related Transforms

It seems worthwhile, however, to point out that there is a variant type transform that "almost" works. This transform is defined by taking $||\forall t.\sigma|| = \forall t.|\sigma|$, and $|\sigma| = (||\sigma|| \to \alpha) \to \alpha$. The intuition behind this choice is to regard polymorphic instantiation as a "serious" computation (in roughly the sense of Reynolds [13]). This interpretation is arguably at variance with the usual semantics of ML polymorphism since it admits primitives that have non-trivial computational effects when polymorphically instantiated. Nevertheless, we can use this type transform to extend the Meyer-Wand theorem to a variant call-by-value CPS transform for DM^- and to a variant call-by-name CPS transform for DM, provided that we restrict attention to programs of monomorphic type. It does not provide a variant call-by-value CPS transform for full DM because of the way in which polymorphic generalization is handled.

To make these observations precise, we sketch the definitions of variant CPS transforms based on this type interpretation. The main idea is to define the CPS transform by induction on typing derivations so that the effect of polymorphic generalization and instantiation can be properly handled. We give here only the two most important clauses, those governing the rules GEN and INST:

$$\begin{aligned} |\Gamma \rhd e : \forall t.\sigma| &= \lambda k.k\,|e|, \text{ where} \\ |\Gamma \rhd e : \sigma| &= |e| \\[1em] |\Gamma \rhd e : [\tau/t]\sigma| &= \lambda k.|e|\,(\lambda x.x\,k), \text{ where} \\ |\Gamma \rhd e : \forall t.\sigma| &= |e| \end{aligned}$$

This definition may be extended to the other inference rules in such a way as to implement either a call-by-name or call-by-value interpretation of application. However, the transform fails (in general) to agree with the usual (call-by-value or call-by-name) ML semantics on terms of polymorphic type. Specifically, the transformation of a GEN rule applies the current continuation to the suspended computation of e. If this continuation is not strict, then an expression that would abort in ML terminates normally after transformation into CPS. For example, consider the principal typing derivation of the term $\mathsf{hd\,nil}$ in a context assigning the obvious types to hd and nil. The resulting transform, when applied to $\lambda x.0$, will yield answer 0, despite the fact that the usual ML semantics leads to aborting in this case.

By restricting attention to programs of monomorphic type, we may obtain a correct CPS transform for DM^- (under call-by-value) and DM (under call-by-name). This is essentially because in DM^- under call-by-value there are no non-trivial polymorphic computations, and because in DM under call-by-name the semantics is defined by substitution. But the above argument shows that this transform is incorrect for DM under call-by-value. Specifically, it fails to correctly implement the usual ML semantics for expressions such as $\mathsf{let}\,x\,\mathsf{be\,hd\,nil\,in}\,0$ (which, under the above transformation yields result 0 rather than aborting).

5 Conclusion

The Meyer-Wand typing theorem for the call-by-value CPS transform for the simply-typed λ-calculus establishes a simple and natural relationship between the type of a term and the type of its call-by-value CPS transform. Meyer and Wand exploited this relationship in their proof of the equivalence of the direct and continuation semantics of $\lambda^{\to}$ [8]. A minor extension of this result may be used to establish the soundness of typing for $\lambda^{\to} + \mathsf{cont}$, the extension of $\lambda^{\to}$ with continuation-passing primitives [3], under call-by-value.

In this paper we have presented a systematic study of the extension of the Meyer-Wand theorem to the Damas-Milner system of polymorphic type assignment. Our main positive results are the extension of the

Meyer-Wand theorem to the call-by-value interpretation of a restricted form of polymorphism, and to the call-by-name interpretation of the unrestricted language. These results have as a consequence the soundness (in the sense of Damas and Milner [2]) of these programming languages. We have also argued that there is no "natural" call-by-value CPS transform for the unrestricted language, but this leaves open the possibility of finding a transformation that is radically different in character from the usual one.

Our investigation makes clear that there is a fundamental tension between implicit polymorphism and the by-value interpretation of let. In particular, we are able to provide a CPS transform for the full Damas-Milner language that extends to continuation-passing primitives, but which is "not quite" equivalent to the usual call-by-value semantics. This suggests that a language in which polymorphic generalization and instantiation are semantically significant would be well-behaved, and might be a suitable alternative to ML-style implicit polymorphism. We plan to report on this subject in a future paper.

6 Acknowledgments

We are grateful to Olivier Danvy, Tim Griffin, Mark Leone, and the referees for their helpful comments on earlier drafts of this paper.

References

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

[2] Luis Damas and Robin Milner. Principal type schemes for functional programs. In *Ninth ACM Symposium on Principles of Programming Languages*, pages 207–212, 1982.

[3] Bruce Duba, Robert Harper, and David MacQueen. Typing first-class continuations in ML. In *Eighteenth ACM Symposium on Principles of Programming Languages*, January 1991.

[4] Matthias Felleisen, Daniel Friedman, Eugene Kohlbecker, and Bruce Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52(3):205–237, 1987.

[5] Timothy Griffin. Private communication., January 1992.

[6] Robert Harper, Bruce Duba, and David MacQueen. Typing first-class continuations in ML. Revised and expanded version of [3], in preparation.

[7] Robert Harper and Mark Lillibridge. Announcement on the `types` electronic forum., July 1991.

[8] Albert R. Meyer and Mitchell Wand. Continuation semantics in typed lambda calculi (summary). In Rohit Parikh, editor, *Logics of Programs*, volume 224 of *Lecture Notes in Computer Science*, pages 219–224. Springer-Verlag, 1985.

[9] Robin Milner. A theory of type polymorphism in programming languages. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[10] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. MIT Press, 1990.

[11] John C. Mitchell. A type-inference approach to reduction properties and semantics of polymorphic expressions. In *1986 Symposium on LISP and Functional Programming*, pages 308–319, August 1986.

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

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

[14] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. Technical Report TR91–160, Department of Computer Science, Rice University, July 1991.