CPS Translations and Applications: the Cube and Beyond (preliminary report)

Gilles Barthe[*] John Hatcliff[†] Morten Heine Sørensen[‡]

December 6, 1996

Abstract

Continuation passing style (CPS) translations of typed λ-calculi have numerous applications. However, the range of these applications have been confined thus far by the fact that CPS translations are known only for non-dependent type systems, thus excluding a number of well-known systems, e.g., the calculus of constructions and LF. The need for more general CPS translations has appeared in several lines of recent work in which the authors are involved.

We discuss the difficulties involved with CPS translating more general systems. We then introduce CPS translations of Barendregt's cube of typed λ-calculi, including the calculus of constructions and the LF calculus. This is generalized further to what we call *strict logical* pure type systems. This class includes all the systems of the cube and logical non-dependent systems of Coquand and Herbelin. Almost all known (call-by-name, Plotkin-style) CPS translations appear as special cases.

Direct style (DS) (i.e., inverse CPS) translations have been used in both technical and implementation-oriented applications. Existing DS translations treat only untyped or simply-typed call-by-value languages. We introduce call-by-name DS translations for all the systems of the cube.

Several applications are sketched.

1 Introduction

1.1 Motivation

Continuation passing style (CPS) translations of typed λ-calculi have appeared throughout the literature since the work of Meyer and Wand [58]. Applications, too numerous to be listed exhaustively here, include compilation, transformation, and analysis of typed languages [1, 13, 36, 64, 76, 77, 82], embedding of classical logics in intuitionistic logics [44, 59], techniques to infer strong normalization from weak normalization in typed λ-calculi [85, 91], and the construction of looping combinators in inconsistent logical pure type systems [16].

The range of these applications have been confined thus far by the fact that CPS translations are known only for non-dependent type systems. Indeed, the most general class of systems with known CPS translation seems to be the logical non-dependent systems, studied by Coquand and Herbelin [16] and later by Werner [90]. While this class contains a number of well-known systems, like simply typed λ-calculus and Girard's second and higher-order typed λ-calculus, it also excludes some other well-known dependent systems, e.g., the calculus of constructions, and the LF calculus [45].

More specifically, the need for more general CPS translations has appeared in several lines of recent work in which the authors are involved [10, 85]. Moreover, further applications of such general translations are emerging and may include conservativity results for classical logics and strong normalization of pure type systems with definitions [81].

[*]Address: CWI, PO Box 94079, 1090 GB Amsterdam, The Netherlands, gilles@cwi.nl

[†]Address: Oklahoma State University, Department of Computer Science, 219 Math Sciences, Stillwater, OK, USA, 74078, hatcliff@a.cs.okstate.edu

[‡]Address: DIKU, Universitetsparken 1, DK-2100 Copenhagen, Denmark, rambo@diku.dk

1.2 This paper

In this paper we introduce CPS translations of Barendregt's cube of typed λ-calculi [8], including the calculus of constructions and the LF calculus. This is then generalized further to what we call *strict logical* pure type systems. This class includes all the systems of the cube and logical non-dependent systems of Coquand and Herbelin. Almost all known (call-by-name, Plotkin-style) CPS translations appear as special cases.

Direct style (DS) (i.e., inverse CPS) translations [19, 21, 76] have been used in both technical and implementation-oriented applications. Existing DS translations treat only untyped or simply-typed call-by-value languages. In this work, we introduce call-by-name DS translations for all the systems of the cube.

In this paper we do not develop applications in detail but merely sketch three areas for applications of generalized CPS translations: infering strong normalization from weak normalization in classes of PTSs, construction of looping combinators in some inconsistent PTSs, and embedding classical PTSs in PTSs.

The techniques we introduce can be applied to other CPS-related translations: call-by-value CPS and DS translations, "optimizing" CPS translations (which yield terms with no administrative redexes), and Fischer-style CPS and DS translations (where continuations appear as the first argument to functions). We plan to present the results for these in the final version of this paper.

1.3 Outline

The rest of the paper is organized as follows. Section 2 reviews Barendregt's cube and pure type systems. Section 3 presents schemes for separating terms in the cube and pure type systems into disjoint categories. This classification of terms is crucial to our definition of CPS transformations.

Section 4 gives several different techniques for CPS translation of the cube. The section begins by outlining technical issues involved in translating pure type systems, and summarizes how these issues have been addressed (or have *not* been addressed) in previous work.

Section 5 defines a DS translation for the cube, and establishes its interaction with the corresponding CPS translation. Section 6 scales up the previously defined CPS translations to pure type systems. Section 7 summarizes various applications of the given CPS translations. Section 8 concludes.

2 Pure Type Systems

Pure type systems were introduced by Barendregt, Berardi and Terlouw as a general framework to define and study typed λ-calculi (see [8, 40]). This section reviews pure type systems and recalls how these can be used to describe Barendregt's cube of eight well-known typed λ-calculi.

Pure type systems are "Church-style" type systems — they feature *domain-specified* abstractions $\lambda x{:}\, A.M$ where a tag A indicates the domain of the argument x. We see in Section 4 that these domain tags are a significant obstacle to a simple formulation of CPS translations for pure type systems. Motivated by this, the authors introduced a variant of pure type systems, called *domain-free* pure type systems, in which abstractions do not carry domain tags, i.e., they have the form $\lambda x.M$ [9]. We present the important properties of domain-free pure type systems, and summarize their connection to domain-specified systems.

2.1 Specifications

Specifications are the abstract structures from which pure type systems and domain-free pure type systems are generated.

Definition 1 A *specification* is a triple $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ where

1. $\mathcal{S}$ is a set of *sorts*;
2. $\mathcal{A} \subseteq \mathcal{S} \times \mathcal{S}$ is a set of *axioms*;
3. $\mathcal{R} \subseteq \mathcal{S} \times \mathcal{S} \times \mathcal{S}$ is a set of *rules*.

As usual, we use (s_1, s_2) to denote rules of the form (s_1, s_2, s_2).

In the process of defining CPS translations, we will consider some standard classes of specifications.

Definition 2 A specification $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ is *functional* if $\mathcal{A}$ and $\mathcal{R}$ are partial functions, i.e. for all $s_1, s_2, s_2', s_3, s_3' \in \mathcal{S}$

- $(s_1, s_2) \in \mathcal{A} \quad \wedge \quad (s_1, s_2') \in \mathcal{A} \quad \Rightarrow \quad s_2 = s_2'$;
- $(s_1, s_2, s_3) \in \mathcal{R} \quad \wedge \quad (s_1, s_2, s_3') \in \mathcal{R} \quad \Rightarrow \quad s_3 = s_3'$.

It is *injective* if it is functional and $\mathcal{A}$ and $\mathcal{R}$ are injective functions, i.e. for all $s_1, s_1' s_2, s_2', s_3, \in S$

- $(s_1, s_2) \in \mathcal{A} \quad \wedge \quad (s_1', s_2) \in \mathcal{A} \quad \Rightarrow \quad s_1 = s_1'$;
- $(s_1, s_2, s_3) \in \mathcal{R} \quad \wedge \quad (s_1', s_2', s_3) \in \mathcal{R} \quad \Rightarrow \quad s_1 = s_1' \quad \wedge \quad s_2 = s_2'$.

In [8], the names of singly-sorted and singly-occupied are used instead of functional and injective.

Throughout the rest of this section, $\mathbf{S} = (\mathcal{S}, \mathcal{A}, \mathcal{R})$ is a fixed specification.

2.2 Pure type systems

Pure type systems PTS are defined as follows.

Definition 3 [pure type systems]

1. The set *Pseudo-Terms*[PTS] is given by the abstract syntax:

$$\begin{array}{rcl} A, B & \in & \textit{Pseudo-Terms}[\mathsf{PTS}] \\ A, B & ::= & x \mid s \mid A\,B \mid \lambda x{:}A.\,B \mid \Pi x{:}A.\,B \end{array}$$

   where $x \in V$ and $s \in \mathcal{S}$.

2. The set *Pseudo-Contexts*[PTS] is given by the abstract syntax:

$$\begin{array}{rcl} \Gamma & \in & \textit{Pseudo-Contexts}[\mathsf{PTS}] \\ \Gamma & ::= & \cdot \mid \Gamma, x : A \end{array}$$

3. *β-reduction* $\rightarrow_\beta$ is defined as the compatible closure of the contraction

$$(\lambda x{:}A.\,M)\;N \rightarrow_\beta M\{x := N\}$$

4. The *pure type system derivability* relation $\vdash$ is given by the rules of Figure 1.

In the definition above, V denotes a fixed but arbitrary set of variables. For convenience, we assume that V may be partitioned as $V = \bigcup_{s \in \mathcal{S}} V^s$ with every set V^s being countably infinite. We identify terms up to renaming of bound variables (i.e., up to α-equivalence), and write $A \equiv B$ when A and B are α-equivalent. Capture-free substitution of B for x in A is denoted $A\{x := B\}$. We use use standard notation and conventions for free variables, contexts, etc. [7]. We write $A \rightarrow B$ for $\Pi x{:}A.\,B$ when $x \notin \mathrm{FV}(B)$. We write $d :: \Gamma \vdash A : B$ when d is a derivation of $\Gamma \vdash A : B$.

Eight of the most important type systems which occur in the literature have been neatly classified in Barendregt's cube of pure type systems.

Definition 4 [Barendregt's cube] Let $\mathcal{S} = \{*, \Box\}$ and $\mathcal{A} = \{(* : \Box)\}$. The eight systems of the cube are obtained by taking the following sets of rules:

$\lambda\!\rightarrow$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
λP	$(*,*)$		$(*,\Box)$	
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda P\omega = \lambda C$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

(axiom)	$<>\vdash s_1 : s_2$	if $(s_1, s_2) \in \mathcal{A}$
(start)	$\dfrac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}$	if $x \notin \Gamma$ and $x \in V^s$
(weakening)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}$	if $x \notin \Gamma$ and $x \in V^s$
(product)	$\dfrac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (\Pi x{:}A.\, B) : s_3}$	if $(s_1, s_2, s_3) \in \mathcal{R}$
(application)	$\dfrac{\Gamma \vdash F : (\Pi x{:}A.\, B) \quad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B\{x := a\}}$	
(abstraction)	$\dfrac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash (\Pi x{:}A.\, B) : s}{\Gamma \vdash \lambda x{:}A.\, b : \Pi x{:}A.\, B}$	
(conversion)	$\dfrac{\Gamma \vdash A : B \quad \Gamma \vdash B' : s}{\Gamma \vdash A : B'}$	if $B =_\beta B'$

Figure 1: PURE TYPE SYSTEMS

$\lambda\!\!\rightarrow$ corresponds to the simply typed λ-calculus whereas $\lambda 2$ and $\lambda\omega$ correspond respectively to second-order and higher-order λ-calculus. λP corresponds to the Edinburgh Logical Framework whereas λC corresponds to the Calculus of Constructions. See [8, 40] for references.

It is sometimes necessary to restrict ourselves to specifications with normalizing types.

Definition 5 **S** has *normalizing types* if every $M \in$ *Pseudo-Terms*[PTS] s.t. $\Gamma \vdash M : s$ for some $\Gamma \in$ *Pseudo-Contexts*[PTS] and $s \in \mathcal{S}$ is normalizing.

2.3 Domain-free pure type systems

We now define domain-free pure type systems PTS^- where domain tags are omitted from abstractions.

Definition 6 [domain-free pure type systems]

1. The set *Pseudo-Terms*[PTS^-] of *domain-free pseudo-terms* is given by the abstract syntax:

$$\begin{array}{rcl} A, B & \in & \textit{Pseudo-Terms}[\mathsf{PTS}^-] \\ A, B & ::= & x \mid s \mid A\, B \mid \lambda x.A \mid \Pi x{:}A.\, B \end{array}$$

   where $x \in V$ and $s \in \mathcal{S}$.

2. The set *Pseudo-Contexts*[PTS^-] of *pseudo-contexts* is given by the abstract syntax:

$$\begin{array}{rcl} \Gamma & \in & \textit{Pseudo-Contexts}[\mathsf{PTS}^-] \\ \Gamma & ::= & \cdot \mid \Gamma, x{:}A \end{array}$$

3. *$\underline{\beta}$-reduction* $\rightarrow_{\underline{\beta}}$ is defined as the compatible closure of the contraction

$$(\lambda x.M)\, N \rightarrow_{\underline{\beta}} M\{x := N\}$$

   .

4. The *domain-free derivability* relation $\Vdash$ is given by the rules of Figure 2.

(axiom)	$<> \Vdash s_1 : s_2$	if $(s_1, s_2) \in \mathcal{A}$
(start)	$\dfrac{\Gamma \Vdash A : s}{\Gamma, x : A \Vdash x : A}$	if $x \notin \Gamma$ and $x \in V^s$
(weakening)	$\dfrac{\Gamma \Vdash A : B \quad \Gamma \Vdash C : s}{\Gamma, x : C \Vdash A : B}$	if $x \notin \Gamma$ and $x \in V^s$
(product)	$\dfrac{\Gamma \Vdash A : s_1 \quad \Gamma, x : A \Vdash B : s_2}{\Gamma \Vdash (\Pi x{:}A.\, B) : s_3}$	if $(s_1, s_2, s_3) \in \mathcal{R}$
(application)	$\dfrac{\Gamma \Vdash F : (\Pi x{:}A.\, B) \quad \Gamma \Vdash a : A}{\Gamma \Vdash F\, a : B\{x := a\}}$	
(abstraction)	$\dfrac{\Gamma, x : A \Vdash b : B \quad \Gamma \Vdash (\Pi x{:}A.\, B) : s}{\Gamma \Vdash \lambda x.b : \Pi x{:}A.\, B}$	
(conversion)	$\dfrac{\Gamma \Vdash A : B \quad \Gamma \Vdash B' : s}{\Gamma \Vdash A : B'}$	if $B =_{\underline{\beta}} B'$

Figure 2: Domain-free pure type systems

2.4 Comparing pure type systems and domain-free pure type systems

The relationship between pure type systems and domain-free pure type systems was closely studied in [9]. We review the main results here.

Definition 7 [erasure] The *erasure* map $|.| : \mathit{Pseudo\text{-}Terms}[\mathsf{PTS}] \to \mathit{Pseudo\text{-}Terms}[\mathsf{PTS}^-]$ is defined as follows:

$$\begin{array}{rcl} |x| & = & x \\ |s| & = & s \\ |t\ u| & = & |t|\ |u| \\ |\lambda x : A.t| & = & \lambda x.|t| \\ |\Pi x : A.B| & = & \Pi x : |A|.|B| \end{array}$$

Erasure preserves and, under certain conditions, reflects typing.

Proposition 1

1. *If* $\Gamma \vdash M : A$ *then* $|\Gamma| \Vdash |M| : |A|$.
2. *Let* $\mathbf{S}$ *be a functional specification with normalizing types. If* $\Delta \Vdash E : F$ *then there exists* $\Gamma \in$ *Pseudo-Contexts*[PTS] *and* $M, A \in$ *Pseudo-Terms*[PTS] *s.t.* $\Gamma \vdash M : A$, $|\Gamma| \equiv \Delta$, $|M| \equiv E$ *and* $|A| \equiv F$.

In fact we can define $\Gamma \vdash M : A$ uniquely by requiring that Γ, M and A are in *domain normal form*, where domain-reduction $\to_{\beta_d}$ is defined as the largest subrelation of $\to_\beta$ s.t. $|M| \equiv |N|$ for every $M \to_{\beta_d} N$. In the sequel, we let $\mathsf{lift}(\Delta \Vdash E : F)$ denote the unique judgment in domain normal form whose erasure is $\Delta \Vdash E : F$. Moreover we let $\mathsf{lift}_c(\Delta \Vdash E : F)$, $\mathsf{lift}_s(\Delta \Vdash E : F)$ and $\mathsf{lift}_p(\Delta \Vdash E : F)$ denote respectively the context, subject and predicate of $\mathsf{lift}(\Delta \Vdash E : F)$ so that the latter may be written as

$$\mathsf{lift}_c(\Delta \Vdash E : F) \vdash \mathsf{lift}_s(\Delta \Vdash E : F) : \mathsf{lift}_p(\Delta \Vdash E : F)$$

2.5 Terms in context

In the course of the paper, two types of CPS translations will be considered: domain-free translations which map pseudo-terms to domain-free pseudo-terms, and domain-specified translations which map pseudo-terms to pseudo-terms. Unlike the domain-free translations, the domain-specified translation is not defined by induction on the structure of the terms. Instead, a more elaborate induction principle must be used. The principle, which we define below, already appears in a slightly different form in [22, 23, 24] to prove the decidability of second-order matching, the completeness of the resolution system and the existence of η-long normal form in the systems of the cube. Throughout this section, we assume that **S** is specification with normalizing types.

Definition 8

1. A *term in context* is a pair $\Gamma \vdash M$ where Γ is a pseudo-context and M is a pseudo-term. The set of terms in context is denoted by $\mathcal{TC}$.
2. The set $\mathsf{Sub}(\Gamma \vdash M)$ of *subterms in context* of $(\Gamma \vdash M)$ is defined inductively as follows:
$$\begin{array}{rcl}
\mathsf{Sub}(\Gamma \vdash x) & = & \emptyset \\
\mathsf{Sub}(\Gamma \vdash s) & = & \emptyset \\
\mathsf{Sub}(\Gamma \vdash M\ N) & = & \{\Gamma \vdash M;\ \Gamma \vdash N\} \cup \mathsf{Sub}(\Gamma \vdash M) \cup \mathsf{Sub}(\Gamma \vdash N) \\
\mathsf{Sub}(\Gamma \vdash \lambda x{:}A.\, M) & = & \{\Gamma \vdash A;\ \Gamma, x : A \vdash M\} \cup \mathsf{Sub}(\Gamma \vdash A) \cup \mathsf{Sub}(\Gamma, x : A \vdash M) \\
\mathsf{Sub}(\Gamma \vdash \Pi x{:}A.\, B) & = & \{\Gamma \vdash A;\ \Gamma, x : A \vdash B\} \cup \mathsf{Sub}(\Gamma \vdash A) \cup \mathsf{Sub}(\Gamma, x : A \vdash B)
\end{array}$$
3. The *subterm relation* $\lhd$ is defined by
$$(\Gamma \vdash M) \lhd (\Delta \vdash N) \quad \Leftrightarrow \quad (\Gamma \vdash M) \in \mathsf{Sub}(\Delta \vdash N)$$
4. The relation $\prec$ is defined as the smallest relation on terms in context s.t.
   (a) $\lhd \subseteq \prec$;
   (b) for every term in context (Γ, M) and $A \in$ *Pseudo-Terms*[PTS],
$$\Gamma \vdash M : A \quad \Rightarrow \quad (\Gamma, \mathsf{nf}\ A) \prec (\Gamma, M)$$
5. The relation $\prec_{\mathsf{nf}}$ is defined as the restriction of $\prec$ to normal forms.

In [24], Dowek, Huet and Werner prove:

Theorem 1 $\prec_{\mathsf{nf}}$ *is well-founded for systems of the λ-cube.*

It follows:

Corollary 1 $\prec$ *is well-founded for systems of the λ-cube.*

3 Classification of Terms

Our methods for defining CPS translations require that we classify terms according to the rôle they play in the formation of legal terms. Intuitively, the translations behave differently when translating terms from different classes. This section presents classification schemes for the cube, and then generalizes this to pure type systems generated by what we call *strict logical* specifications.

3.1 Classification of terms in the cube

Below we show how terms belonging to systems in the cube can be classified according to the familiar notions of *object*, *constructors*, and *kinds* [45].

Definition 9 [classification of pseudo-terms of the cube]
Define the sets *Pseudo-Obj*[CUBE], *Pseudo-Constr*[CUBE], *Pseudo-Kind*[CUBE] as follows:

$$\begin{array}{lcl} O & \in & \textit{Pseudo-Obj}[\mathsf{CUBE}] \\ O & ::= & V^{*} \mid \lambda V^{*}{:}C.\,O \mid O\,O' \mid \lambda V^{\square}{:}K.\,O \mid O\,C \\ \\ C & \in & \textit{Pseudo-Constr}[\mathsf{CUBE}] \\ C & ::= & V^{\square} \mid \lambda V^{*}{:}C.\,C \mid C\,O \mid \lambda V^{\square}{:}K.\,C \mid C\,C \mid \Pi V^{*}{:}C.\,C \mid \Pi V^{\square}{:}K.\,C \\ \\ K & \in & \textit{Pseudo-Kind}[\mathsf{CUBE}] \\ K & ::= & \Pi V^{*}{:}C.\,K \mid \Pi V^{\square}{:}K.\,K \mid * \end{array}$$

Definition 10 [Classification of legal terms in the cube]
Define the sets *Legal-Obj*[CUBE], *Legal-Constr*[CUBE], *Legal-Kind*[CUBE] as follows:

$$\begin{array}{rcl} \textit{Legal-Obj}[\mathsf{CUBE}] & = & \{O \in \textit{Pseudo-Obj}[\mathsf{CUBE}] \mid \exists \Gamma, A\,.\,\Gamma \vdash O : A\} \\ \textit{Legal-Constr}[\mathsf{CUBE}] & = & \{C \in \textit{Pseudo-Constr}[\mathsf{CUBE}] \mid \exists \Gamma, A\,.\,\Gamma \vdash C : A\} \\ \textit{Legal-Kind}[\mathsf{CUBE}] & = & \{K \in \textit{Pseudo-Kind}[\mathsf{CUBE}] \mid \exists \Gamma\,.\,\Gamma \vdash K : \square\} \end{array}$$

Property 1 [Classification of the cube]

1. The sets *Legal-Obj*[CUBE],*Legal-Constr*[CUBE], *Legal-Kind*[CUBE], and $\{\square\}$ are pairwise disjoint and closed under reduction.
2. If $\Gamma \vdash A : B$ then exactly one of the following holds:
   - $(A, B) \in$ *Legal-Obj*[CUBE] $\times$ *Legal-Constr*[CUBE], or
   - $(A, B) \in$ *Legal-Constr*[CUBE] $\times$ *Legal-Kind*[CUBE], or
   - $(A, B) \in$ *Legal-Kind*[CUBE] $\times \{\square\}$

One obtains similar classifications for particular systems within the cube.

1. For $\lambda{\rightarrow}$,$\lambda 2$,$\lambda\omega$, and $\lambda\underline{\omega}$ omit the clauses $\lambda V^{*}{:}C.\,C$, $C\,O$, and $\Pi V^{*}{:}C.\,K$;
2. For $\lambda{\rightarrow}$,$\lambda 2$,$\lambda P2$, and λP omit the clauses $\lambda V^{\square}{:}K.\,C$, $C\,C$, and $\Pi V^{\square}{:}K.\,K$;
3. For $\lambda{\rightarrow}$,$\lambda\underline{\omega}$,$\lambda P\underline{\omega}$, and λP omit the clauses $\lambda V^{\square}{:}K.\,O$, $O\,C$, and $\Pi V^{\square}{:}K.\,C$.

3.2 Classification of terms in pure type systems

Not all pure type systems admit a classification scheme analogous to the one defined for the cube. For example, one might imagine a non-functional specification where a term plays "multiple rôles" — acting e.g., like a constructor as well as a kind. However, we can identify a large class of pure type systems that admit a classification property sufficient to support the definition of our CPS translations. We must consider pure type systems with a specific sort of propositions. It is convenient to follow [16] and use the notion of logical specification.

Definition 11

1. A *logical specification* is a quadruple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathsf{Prop})$ where $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ is a functional specification and $\mathsf{Prop} \in \mathcal{S}$ is a sort s.t.
   (a) there exists $s \in \mathcal{S}$ s.t. $(\mathsf{Prop}, s) \in \mathcal{A}$;

(b) there is no $s \in \mathcal{S}$ s.t. $(s, \mathsf{Prop}) \in \mathcal{A}$;

(c) $(\mathsf{Prop}, \mathsf{Prop}, \mathsf{Prop}) \in \mathcal{R}$.

2. A logical specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ is *non-dependent* if the only rules in which Prop occurs are of the form (s, Prop).

3. A logical specification $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ is *strict* if every rule in which Prop has one of the following forms:

   (a) (s, Prop), or

   (b) (Prop, s, s') with $s, s' \neq \mathsf{Prop}$.

The notion of non-dependent logical specification was introduced by Coquand and Herbelin in their study of looping combinators in pure type systems [16]. Examples of non-dependent logical specifications include $\lambda{\rightarrow}$, $\lambda 2$, $\lambda\omega$ but also λHOL, λU^- and λU. In contrast, λP, $\lambda P2$ and λC are examples of dependent but strict logical specifications. More generally, every logical specification with rules of the form (s_1, s_2) only is strict. Thus, all systems of the cube are strict logical specifications (the cube sort $*$ corresponds to Prop). Finally, $\lambda *$ is not a logical specification because $\mathsf{Prop} : \mathsf{Prop}$ is an axiom and hence the second requirement is violated.

We now present a scheme for classifying terms into four categories: *proofs*, *elements*, *propositions*, and *sets*.

Definition 12

$$\begin{array}{rcl} \Gamma{-}\mathsf{Proof} & = & \{M \in \mathcal{T} \mid \Gamma \vdash M : A : \mathsf{Prop} \quad \text{for some } A \in \mathcal{T}\} \\ \Gamma{-}\mathsf{El} & = & \{M \in \mathcal{T} \mid \Gamma \vdash M : A : s \quad \text{for some } A \in \mathcal{T} \text{ and } s \in \mathcal{S} \setminus \{\mathsf{Prop}\}\} \\ \Gamma{-}\mathsf{Prop} & = & \{M \in \mathcal{T} \mid \Gamma \vdash M : \mathsf{Prop}\} \\ \Gamma{-}\mathsf{Set} & = & \{M \in \mathcal{T} \mid \Gamma \vdash M : s \quad \text{for some } s \in \mathcal{S} \setminus \{\mathsf{Prop}\}\} \end{array}$$

Moreover we define

$$\mathsf{Proof} = \bigcup_{\Gamma \in \mathcal{C}} \Gamma{-}\mathsf{Proof} \quad \mathsf{El} = \bigcup_{\Gamma \in \mathcal{C}} \Gamma{-}\mathsf{El} \quad \mathsf{Prop} = \bigcup_{\Gamma \in \mathcal{C}} \Gamma{-}\mathsf{Prop} \quad \mathsf{Set} = \bigcup_{\Gamma \in \mathcal{C}} \Gamma{-}\mathsf{Set}$$

A strict logical specification (S, A, R, Prop) has the *classification property* if $\mathsf{Prop} \cap \mathsf{Set} = \emptyset$.

Propositions are defined as those types which inhabit Prop whereas sets are defined as those types which inhabit another universe. Here the word 'set' is used very loosely for any type not being a proposition. Quite naturally, we further distinguish between proofs, which inhabit propositions, and elements, which inhabit sets. The classification property states that terms cannot be simultaneously a proof and an element.

It is possible to give a precise analysis of the structure of proofs for strict logical specifications.

Lemma 1 *Let* **S** *be a strict logical specification.*

1. *If x is legal in Γ, then $x \in \Gamma{-}\mathsf{Proof} \quad \Leftrightarrow \quad x \in V^{\mathsf{Prop}}$.*

2. *For every $s \in \mathcal{S}$, $s \notin \Gamma{-}\mathsf{Proof}$.*

3. *If $M\ M'$ is legal in Γ, then $M\ M' \in \Gamma{-}\mathsf{Proof} \quad \Leftrightarrow \quad M \in \Gamma{-}\mathsf{Proof}$.*

4. *If $\lambda x{:}A.\, M$ is legal in Γ, then $\lambda x{:}A.\, M \in \Gamma{-}\mathsf{Proof} \quad \Leftrightarrow \quad M \in (\Gamma, x : A){-}\mathsf{Proof}$.*

5. *For every $\Pi x{:}A.\, B \in \mathcal{T}$, $\Pi x{:}A.\, B \notin \Gamma{-}\mathsf{Proof}$.*

Proof: By a direct analysis of the possible derivations. ∎

As a result, we obtain:

Proposition 2 *Let* **S** *be a strict logical specification. Then* $\mathsf{Proof} \cap \mathsf{El} = \emptyset$.

Proof: We show by induction on the structure of the terms that

$$M \in \mathsf{El} \quad \Rightarrow \quad M \notin \mathsf{Proof}$$

We treat two cases:

- *application:* assume $M \equiv M_1\, M_2$ and $\Gamma \vdash M_1\, M_2 : C : s$ for some Γ, C and $s \neq \mathsf{Prop}$. By generation, $\Gamma \vdash M_1 : \Pi x{:}\, A.\, B$ and $\Gamma \vdash M_2 : A$ for some $\Pi x{:}\, A.\, B$ s.t. $B\{x := M_2\} =_\beta C$. By Correctness of Types, $\Gamma \vdash \Pi x{:}\, A.\, B : s'$ for some $s' \in \mathcal{S}$. By generation, $\Gamma \vdash A : s_1$ and $\Gamma, x : A \vdash B : s_2$ for some $(s_1, s_2, s') \in \mathcal{R}$. By substitution, $\Gamma \vdash B\{x := M_2\} : s_2$. By Uniqueness of Types, $s \equiv s_2$ and hence $s_2 \neq \mathsf{Prop}$. By strictness, $s' \neq \mathsf{Prop}$. Hence $M_1 \in \mathsf{El}$. By induction hypothesis $M_1 \notin \mathsf{Proof}$ and by Lemma 1.3 $M \notin \mathsf{Proof}$.
- *abstraction:* assume $M \equiv \lambda x{:}\, A.\, M_1$ and $\Gamma \vdash \lambda x{:}\, A.\, M_1 : C : s$ for some Γ, C and $s \neq \mathsf{Prop}$. By generation, $\Gamma, x : A \vdash M_1 : B$ and $\Gamma \vdash \Pi x{:}A.\, B : s'$ for some $s' \in \mathcal{S}$ and $\Pi x{:}A.\, B =_\beta C$. By Uniqueness of Types, $s \equiv s'$ and hence $s' \neq \mathsf{Prop}$. By generation, $\Gamma \vdash A : s_1$ and $\Gamma, x : A \vdash B : s_2$ for some $(s_1, s_2, s') \in \mathcal{R}$. By strictness, $s_2 \neq \mathsf{Prop}$. Hence $M_1 \in \mathsf{El}$. By induction hypothesis $M_1 \notin \mathsf{Proof}$ and by Lemma 1.4 $M \notin \mathsf{Proof}$.

∎

The following observation is sometimes useful to prove that a given strict logical specification has the classification property.

Lemma 2 *Let* **S** *be a strict logical specification and let $X \subseteq \mathcal{S}$. The closure $\mathsf{cl}(X) \subseteq \mathcal{S}$ is defined as follows:*

1. $X \subseteq \mathsf{cl}(X)$
2. *if $(s_1, s_2, s_3) \in \mathcal{R}$ and $s_2 \in \mathsf{cl}(X)$ then $s_3 \in \mathsf{cl}(X)$.*

Let

$$\begin{array}{rcl} \mathcal{S}_{\mathsf{Prop}} & = & \mathsf{cl}(\{s \in \mathcal{S} \mid (\mathsf{Prop}, s) \in \mathcal{A}\}) \\ \mathcal{S}_{\mathsf{Set}} & = & \mathsf{cl}(\{s \in \mathcal{S} \mid (s', s) \in \mathcal{A} \textit{ for some } s' \neq \mathsf{Prop}\}) \end{array}$$

If $\mathcal{S}_{\mathsf{Prop}} \cap \mathcal{S}_{\mathsf{Set}} = \emptyset$, then **S** *has the classification property.*

Note that, in case all rules of **S** are of the form (s_1, s_2) then $\mathsf{cl}(X) = X$. Hence **S** has the classification property if for every $s, s' \in \mathcal{S}$, we have

$$(\mathsf{Prop}, s) \in \mathcal{A} \quad \wedge \quad (s', s) \in \mathcal{A} \quad \Rightarrow \quad s' = \mathsf{Prop}$$

As an application of Lemma 2, we obtain:

Corollary 2 *Systems of Barendregt's cube, λHOL, λU^- and λU have the Classification Property.*

4 CPS Translations of the Cube

4.1 Issues

4.1.1 Which method of definition should be used?

Our CPS translations for pure type systems follow the structure of Plotkin's call-by-name CPS translation [70] for untyped λ-terms as presented in Figure 3.[1]

The translation is defined by *induction over the structure (or alternatively, the size) of terms.* This method of definition scales up to richer languages that are either untyped [20, 76] or typed using a "Curry-style" type system [47, 48].

[1] We follow Hatcliff and Danvy [49] on the translation of variables.

$$\begin{aligned}
\mathcal{C}\langle x\rangle &= \lambda k.x\,k\\
\mathcal{C}\langle \lambda x.M\rangle &= \lambda k.k\,(\lambda x.\mathcal{C}\langle M\rangle)\\
\mathcal{C}\langle M_1\,M_2\rangle &= \lambda k.\mathcal{C}\langle M_1\rangle\,(\lambda y.y\,\mathcal{C}\langle M_2\rangle\,k)
\end{aligned}$$

$$M \;::=\; x \;\mid\; \lambda x.M \;\mid\; M_1\,M_2$$

Figure 3: Plotkin's call-by-name CPS translation for untyped λ-terms

However, in "Church-style" type systems (including the original formulation of PTS as in Section 2.2), this method is not sufficient. These systems feature *domain-specified* abstractions $\lambda x{:}\,A.\,M$ where a tag A is attached to indicate the domain of the argument x. In these systems, one needs additional information information about the type of a term to generate tags for the abstractions of the form $\lambda k.\ldots$ and $\lambda y.\ldots$ in the translation above.

The conventional solution is to define the CPS translation by induction over the structure (or over the height) of derivations $\Gamma \vdash M : A$. For example, this is the approach taken by Harper and Lillibridge when translating $\lambda\omega$ into CPS [46]. With such definitions, one generally desires a *coherence property*: given two different derivations of the same judgment $d_1, d_2 :: \Gamma \vdash M : A$, the results of the translation $\mathcal{C}\langle d_1\rangle$ and $\mathcal{C}\langle d_2\rangle$ are *equivalent*. Reasonable notions of equivalence include syntactic identity as well as weaker notions such as β-convertibility. Coherence is a crucial property upon which proofs of other properties are built (e.g., properties showing how the CPS translation interacts with substitution, reduction, and conversion, as well properties showing that the translation preserves derivable judgments).

When naively defining the CPS translation of the cube by induction over derivations, one runs into difficulty with the *conversion* rule (which involves β-conversion) and the *application* rule (which involves substitution). The definition of the translation itself involves convertability and substitution. Thus, one cannot proceed by first addressing coherence and *then* proceeding to substitution, and conversion, etc.; one is forced to tackle the associated properties simultaneously.

As a point of comparison, Harper and Lillibridge [46] avoid both the problems associated with the conversion and application rules. When translating two different derivations of the same judgment $d_1, d_2 :: \Gamma \vdash M : A$, use of the conversion rule in either d_1 or d_2 may cause the terms $\mathcal{C}\langle d_1\rangle$ and $\mathcal{C}\langle d_2\rangle$ to have different domain tags on abstractions. Instead of considering a general theory of equality based on β-convertability (which is what we desire), Harper and Lillibridge consider a notion of object equivalence based on an operationally-flavored form of reduction (e.g., call-by-name standard reduction) that is insensitive to domain tags. This insensitivity implies that differences in domain tags do not affect the equivalence of terms [46, p. 214]. The substitution in the application rule is neither an issue since $\lambda\omega$ (the system considered by Harper and Lillibridge) is a non-dependent system, and thus the substitution $B\{x := a\}$ degenerates into B.

The substitution in the application rule is one instance of several technical hurdles one meets when treating dependent systems (e.g., the systems on the right-hand side of the cube – λP, $\lambda P2$, $\lambda P\underline{\omega}$, λC). In the non-dependent systems of the cube, one does not have the rule $(*, \Box)$. This leads to a clean stratification of levels: kinds, constructors (whose legal formation depends only on kinds), and objects (whose legal formation depends only on kinds, and constructors). When addressing the correctness of translations of these systems, one may consider the translation of kinds in isolation, and then proceed to constructors, and end up with objects. This is the route taken by Harper and Lillibridge, and also by Coquand and Herbelin [16] in their CPS translation of non-dependent pure type systems. In dependent systems, the categories of terms (kinds, constructors, and objects) are mutually-dependent. Thus, properties associated with the translation of objects, constructors, and kinds must typically be proved simultaneously.

In conclusion, the issues related to coherence and dependent systems discussed above are not insignificant technical hurdles. As evidence, consider that Coquand and Herbelin explicitly limit themselves to non-dependent systems when using a CPS translation to show the existence of looping combinators in certain pure type systems — even though including dependent systems is more natural goal [16].

In the present work we take an application-directed approach to these challenges. For example, in some applications of CPS translations for pure type systems, it is necessary to consider translations to domain-free systems (as introduced in Section 2.3). The absence of domain tags in the target systems allows one simply to define the CPS translation by induction on the structure of terms — thus, avoiding many of the problems discussed above. This was the motivation for introducing domain-free systems in Section 2. Exploration of translations into domain-free systems is our primary focus (beginning in Section 4.2). For the sake of completeness, we also present a translation which maps domain-specified terms to domain-specified terms (see Section 4.5).

4.1.2 To which abstractions should one pass continuations?

When one moves from $\lambda\!\!\rightarrow$ to systems with richer type structure, it is not immediately clear which syntactic categories should be converted into CPS. For example, in the cube, one might imagine a *non-pervasive* CPS translation where only objects are CPSed (i.e., only abstractions at the object level are passed continuations), or a *pervasive* translation where both objects and constructors are CPSed (i.e., all abstractions are passed continuations).

Again, our presentation is motivated by particular applications.

Classical pure type systems: In classical pure type systems [10], the control operator corresponding to the *reductio ad absurdum* rule only appears at the object level. Thus, one only needs to CPS objects when embedding classical pure type systems into traditional (constructive) pure type systems.

Existence of looping combinators: In Coquand and Herbelin's [16] method for showing the existence of looping combinators, one only needs to CPS objects.

The classification properties of Section 3 allow us to define translations where only certain terms are converted to CPS.

4.2 Domain-free CPS translation of the cube

Figure 4 presents the CPS translation from the domain-specified cube CUBE into the domain-free cube CUBE^-. As noted earlier, the translation of objects is based on Plotkin's original call-by-name translation of the untyped λ-calculus (see Figure 3). Continuation-passing only occurs at the object level. Thus, the translation of constructors and kinds is straightforward. $\neg\mathcal{C}$ abbreviates $\mathcal{C} \rightarrow \bot$ where $\bot \in V^{\square}$ is a distinguished (but arbitrary) variable of *answers*.

The top level translation $\mathcal{C}\{\!|\cdot|\!\}$ for objects and kinds simply calls $\mathcal{C}\langle\cdot\rangle$. For constructors, a double negation is added at the outer level. For assumptions, the answer variable $\bot$ is added.

The following theorem establishes the correctness of the translation.

Theorem 2 [Correctness of CPS translation]

$$\Gamma \vdash A : B \quad\Rightarrow\quad \mathcal{C}\{\!|\Gamma|\!\} \Vdash \mathcal{C}\langle A\rangle : \mathcal{C}\{\!|B|\!\}$$

Proof sketch: Since the rules of the cube involve β-convertability, one must first show that the CPS translation preserves β-convertability. This requires showing how the translation interacts with substitution. The proof then follows by induction over the height of derivations. ∎

CPS translations for specific systems within the cube are obtained by specializing the translated language following the presentation in Section 3.1.

4.3 Language of CPS terms

In Section 5, we present a DS translation for the cube that maps continuation-passing terms back to direct-style terms. Since the rules of the cube involve β-conversion, the DS translation must be able to handle not only terms in the image of the CPS translation, but terms β-convertible with terms in the image of the CPS translation. Since systems of the cube are Church-Rosser, it is sufficient to reason about the language of

Objects

$$\begin{array}{rcl}
\mathcal{C}\langle V^* \rangle & = & \lambda k. V^*\, k \\
\mathcal{C}\langle \lambda V^* {:} C.\, O \rangle & = & \lambda k. k\, (\lambda V^*. \mathcal{C}\langle O \rangle) \\
\mathcal{C}\langle O\, O' \rangle & = & \lambda k. \mathcal{C}\langle O \rangle\, (\lambda y. y\, \mathcal{C}\langle O' \rangle\, k) \\
\mathcal{C}\langle \lambda V^\square {:} K.\, O \rangle & = & \lambda k. k\, (\lambda V^\square. \mathcal{C}\langle O \rangle) \\
\mathcal{C}\langle O\, C \rangle & = & \lambda k. \mathcal{C}\langle O \rangle\, (\lambda y. y\, \mathcal{C}\langle C \rangle\, k)
\end{array}$$

Constructors

$$\begin{array}{rcl}
\mathcal{C}\langle V^\square \rangle & = & V^\square \\
\mathcal{C}\langle \lambda V^* {:} C.\, C' \rangle & = & \lambda V^*. \mathcal{C}\langle C' \rangle \\
\mathcal{C}\langle C\, O \rangle & = & \mathcal{C}\langle C \rangle\, \mathcal{C}\langle O \rangle \\
\mathcal{C}\langle \lambda V^\square {:} K.\, C \rangle & = & \lambda V^\square. \mathcal{C}\langle C \rangle \\
\mathcal{C}\langle C\, C' \rangle & = & \mathcal{C}\langle C \rangle\, \mathcal{C}\langle C' \rangle \\
\mathcal{C}\langle \Pi V^* {:} C.\, C' \rangle & = & \Pi V^* {:} \neg\neg\mathcal{C}\langle C \rangle.\, \neg\neg\mathcal{C}\langle C' \rangle \\
\mathcal{C}\langle \Pi V^\square {:} K.\, C \rangle & = & \Pi V^\square {:} \mathcal{C}\langle K \rangle.\, \neg\neg\mathcal{C}\langle C \rangle
\end{array}$$

Kinds

$$\begin{array}{rcl}
\mathcal{C}\langle * \rangle & = & * \\
\mathcal{C}\langle \Pi V^* {:} C.\, K \rangle & = & \Pi V^* {:} \neg\neg\mathcal{C}\langle C \rangle.\, \mathcal{C}\langle K \rangle \\
\mathcal{C}\langle \Pi V^\square {:} K.\, K' \rangle & = & \Pi V^\square {:} \mathcal{C}\langle K \rangle.\, \mathcal{C}\langle K' \rangle
\end{array}$$

Contexts

$$\begin{array}{rcl}
\mathcal{C}\langle \cdot \rangle & = & \cdot \\
\mathcal{C}\langle \Gamma, V^* {:} C \rangle & = & \mathcal{C}\langle \Gamma \rangle, V^* {:} \neg\neg\mathcal{C}\langle C \rangle \\
\mathcal{C}\langle \Gamma, V^\square {:} K \rangle & = & \mathcal{C}\langle \Gamma \rangle, V^\square {:} \mathcal{C}\langle K \rangle
\end{array}$$

Top-level translation

$$\begin{array}{rcl}
\mathcal{C}\langle\!\!\langle O \rangle\!\!\rangle & = & \mathcal{C}\langle O \rangle \\
\mathcal{C}\langle\!\!\langle C \rangle\!\!\rangle & = & \neg\neg\mathcal{C}\langle C \rangle \\
\mathcal{C}\langle\!\!\langle K \rangle\!\!\rangle & = & \mathcal{C}\langle K \rangle \\
\mathcal{C}\langle\!\!\langle \square \rangle\!\!\rangle & = & \square \\
\mathcal{C}\langle\!\!\langle \Gamma \rangle\!\!\rangle & = & \bot {:} *, \mathcal{C}\langle \Gamma \rangle
\end{array}$$

Figure 4: Call-by-name CPS translation into the domain-free cube

legal terms in the image of CPS translation closed under β-reduction. This section presents CPS pseudo-terms, and then gives rules for deriving legal CPS terms. Although the CPS translation of Figure 4 yields domain-free terms, we present a domain-specified CPS language. The corresponding domain free language is obtained by an obvious erasure mapping and modified rules analogous to those introduced in Section 2.3.

When discussing terms in the image of the CPS translation, it is convenient to divide abstractions into two classes.

source abstractions: These are CPSed versions of abstractions appearing in the argument of the CPS translation. In Figure 4, abstractions of the form $(\lambda V^{*}. \ldots)$ and $(\lambda V^{\square}. \ldots)$ in the image of the translation are source abstractions.

administrative abstractions: These are abstractions introduced by the translation to manipulate continuations — there are no corresponding abstractions in the argument of the translation. In Figure 4, abstractions of the form $(\lambda k. \ldots)$ and $(\lambda y. \ldots)$ are administrative abstractions.

Figure 5 presents the pseudo-terms for the CPS language. Objects are divided into the following five categories.

Computations: These are pseudo-objects to which continuations will be passed. The types of these objects will be double-negated at the top level.

Values: These are the source abstractions defined above. The types of these objects will not be double-negated at the top-level.

The terms *computation* and *value* are inspired by presentations of CPS based on monads [48]. In a monadic framework, our computations have "computational types", and our values have "value types".

Arguments: These will be the arguments to source abstractions.

Answers: These are pseudo-objects that will have answer type $\perp$. An answer results from passing a value to continuation, or passing a continuation to a computation.

Continuations: These are either continuation identifiers, or abstractions that will be passed values.

It is convenient to divide identifiers of the CPS language into four disjoint sets:

Constructor identifiers: bound by source abstractions that take constructors as arguments;

Computation identifiers: bound by source abstractions that take computations as arguments;

Value identifiers: bound by administrative abstractions that take values as arguments — they are the formal parameters of continuations;

Continuation identifiers: bound by administrative abstractions that take continuations as arguments.

The first three sets are countably infinite. Only one continuation identifier k is needed — a familiar property of CPS terms [19, 21, 76].

CPS contexts always include the answer variable $\perp$. CPS contexts only contain constructor and computation identifiers. It is unnecessary for contexts to contain value and continuation identifiers (which are the identifiers bound by administrative abstractions). These are handled as special cases when defining legal terms.

Figures 6 and 7 present rules for deriving legal terms in each syntactic category of the CPS language. In the judgments for answers and continuations in Figure 6, continuation identifiers k are given special treatment in the context.

In addition to the rules for specific categories, Figure 8 gives generic weakening and conversion rules which apply to each syntactic category. Since all of the conversion rules have a similar form, we only show cases.

A derivation of a domain-free version CPS^{-} of the CPS language is similar the derivation of the domain-free cube in Section 2.3. We use $\Vdash$ in the judgments for domain-free CPS language.

The following property states that legal terms in CPS are also legal terms in CUBE.

Objects

$$\mathcal{M} ::= \mathit{Computations}[\mathsf{CPS}] \qquad \mathcal{A} \in \mathit{Answers}[\mathsf{CPS}]$$
$$\mathcal{M} ::= x \mid \lambda k{:}\neg C.\, \mathcal{A} \mid \mathcal{V}\,\mathcal{N} \qquad \mathcal{A} ::= \mathcal{K}\,\mathcal{V} \mid \mathcal{M}\,\mathcal{K}$$

$$\mathcal{V} \in \mathit{Values}[\mathsf{CPS}] \qquad \mathcal{K} \in \mathit{Continuations}[\mathsf{CPS}]$$
$$\mathcal{V} ::= \lambda x{:}\neg\neg C_1.\, \lambda k{:}\neg C_2.\, \mathcal{A} \mid \lambda z{:}K.\, \lambda k{:}\neg C_2.\, \mathcal{A} \qquad \mathcal{K} ::= k \mid \lambda y{:}C.\, y\,\mathcal{N}\,\mathcal{K}$$

$$\mathcal{N} \in \mathit{Arguments}[\mathsf{CPS}]$$
$$\mathcal{N} ::= \lambda k{:}\neg C.\, \mathcal{A} \mid C$$

Constructors and Kinds

$$\begin{array}{lcl} C & \in & \mathit{Constr}[\mathsf{CPS}] \\ C & ::= & z \mid \lambda x{:}\neg\neg C_1.\, C_2 \mid \lambda z{:}K_1.\, C_2 \mid C\,\mathcal{N} \mid \\ & & \Pi x{:}\neg\neg C_1.\, \neg\neg C_2 \mid \Pi z{:}K.\, \neg\neg C \end{array}$$

$$\begin{array}{lcl} K & \in & \mathit{Kind}[\mathsf{CPS}] \\ K & ::= & * \mid \Pi x{:}\neg\neg C.\, K \mid \Pi z{:}K_1.\, K_2 \end{array}$$

Identifiers

$$\begin{array}{lcll} z & \in & \mathit{Constructor\text{-}ident}[\mathsf{CPS}] & \subseteq V^{\square} \\ x & \in & \mathit{Computation\text{-}ident}[\mathsf{CPS}] & \subseteq V^{*} \\ y & \in & \mathit{Value\text{-}ident}[\mathsf{CPS}] & \subseteq V^{*} \\ k & \in & \mathit{Continuation\text{-}ident}[\mathsf{CPS}] = \{k\} & \subseteq V^{*} \end{array}$$

Contexts

$$\begin{array}{lcl} \Gamma & \in & \mathit{Pseudo\text{-}Contexts}[\mathsf{CPS}] \\ \Gamma & ::= & \bot : * \mid \Gamma, x : \neg\neg C \mid \Gamma, z : K \end{array}$$

Figure 5: CPS PSEUDO-TERMS

Computations

$$\frac{\Gamma \vdash_{\mathsf{con}} C : *}{\Gamma, x : \neg\neg C \vdash_{\mathsf{com}} x : \neg\neg C} \quad \text{if } x \notin \Gamma \text{ and } x \in \mathit{Computation\text{-}ident}[\mathsf{CPS}]$$

$$\frac{\langle\Gamma\rangle\,\langle k : \neg C\rangle \vdash_{\mathsf{ans}} \mathcal{A} : \bot}{\Gamma \vdash_{\mathsf{com}} \lambda k{:}\neg C.\,\mathcal{A} : \neg\neg C}$$

$$\frac{\Gamma \vdash_{\mathsf{val}} \mathcal{V} : \Pi z{:}K_1.\,\neg\neg C_2 \qquad \Gamma \vdash_{\mathsf{arg}} C_1 : K_1}{\Gamma \vdash_{\mathsf{com}} \mathcal{V}\,C_1 : \neg\neg C_2\{z := C_1\}}$$

$$\frac{\Gamma \vdash_{\mathsf{val}} \mathcal{V} : \Pi x{:}\neg\neg C_1.\,\neg\neg C_2 \qquad \Gamma \vdash_{\mathsf{arg}} (\lambda k{:}\neg C_1.\,\mathcal{A}) : \neg\neg C_1}{\Gamma \vdash_{\mathsf{com}} \mathcal{V}\,(\lambda k{:}\neg C_1.\,\mathcal{A}) : \neg\neg C_2\{x := \lambda k{:}\neg C_1.\,\mathcal{A}\}}$$

Values

$$\frac{\Gamma, x : \neg\neg C_1 \vdash_{\mathsf{com}} \lambda k{:}\neg C_2.\,\perp : \neg\neg C_2 \qquad \Gamma \vdash_{\mathsf{con}} \Pi x{:}\neg\neg C_1.\,\neg\neg C_2 : *}{\Gamma \vdash_{\mathsf{val}} \lambda x{:}\neg\neg C_1.\,\lambda k{:}\neg C_2.\,\mathcal{A} : \Pi x{:}\neg\neg C_1.\,\neg\neg C_2}$$

$$\frac{\Gamma, z : K_1 \vdash_{\mathsf{com}} \lambda k{:}\neg C_2.\,\perp : \neg\neg C_2 \qquad \Gamma \vdash_{\mathsf{con}} \Pi z{:}K_1.\,\neg\neg C_2 : *}{\Gamma \vdash_{\mathsf{val}} \lambda z{:}K_1.\,\lambda k{:}\neg C_2.\,\mathcal{A} : \Pi z{:}K_1.\,\neg\neg C_2}$$

Arguments

$$\frac{\langle\Gamma\rangle\,\langle k : \neg C\rangle \vdash_{\mathsf{ans}} \mathcal{A} : \bot}{\Gamma \vdash_{\mathsf{arg}} \lambda k{:}\neg C.\,\mathcal{A} : \neg\neg C} \qquad \frac{\Gamma \vdash_{\mathsf{con}} C : K}{\Gamma \vdash_{\mathsf{arg}} C : K}$$

Answers

$$\frac{\langle\Gamma\rangle\,\langle k : \neg C_1\rangle \vdash_{\mathsf{cnt}} \mathcal{K} : \neg C_2 \qquad \Gamma \vdash_{\mathsf{val}} \mathcal{V} : C_2}{\langle\Gamma\rangle\,\langle k : \neg C_1\rangle \vdash_{\mathsf{ans}} \mathcal{K}\,\mathcal{V} : \bot}$$

$$\frac{\Gamma \vdash_{\mathsf{com}} \mathcal{M} : \neg\neg C_2 \qquad \langle\Gamma\rangle\,\langle k : \neg C_1\rangle \vdash_{\mathsf{cnt}} \mathcal{K} : \neg C_2}{\langle\Gamma\rangle\,\langle k : \neg C_1\rangle \vdash_{\mathsf{ans}} \mathcal{K}\,\mathcal{V} : \bot}$$

Continuations

$$\frac{\Gamma \vdash_{\mathsf{con}} C : *}{\langle\Gamma\rangle\,\langle k : \neg C\rangle \vdash_{\mathsf{cnt}} k : \neg C}$$

$$\frac{\Gamma \vdash_{\mathsf{con}} \Pi x{:}\neg\neg C_1.\,\neg\neg C_2 : * \quad \Gamma \vdash_{\mathsf{arg}} \lambda k{:}\neg C_1.\,\mathcal{A} : \neg\neg C_1 \quad \langle\Gamma\rangle\,\langle k : \neg C_0\rangle \vdash_{\mathsf{cnt}} \mathcal{K} : \neg C_2\{x := (\lambda k{:}\neg C_1.\,\mathcal{A})\}}{\langle\Gamma\rangle\,\langle k : \neg C_0\rangle \vdash_{\mathsf{cnt}} (\lambda y{:}(\Pi x{:}\neg\neg C_1.\,\neg\neg C_2).\,y\,(\lambda k{:}\neg C_1.\,\mathcal{A})\,\mathcal{K}) : \neg\Pi x{:}\neg\neg C_1.\,\neg\neg C_2}$$

if $y \notin \Gamma$ and $y \in \mathit{Computation\text{-}ident}[\mathsf{CPS}]$

$$\frac{\Gamma \vdash_{\mathsf{con}} \Pi z{:}K_1.\,\neg\neg C_2 : * \qquad \Gamma \vdash_{\mathsf{arg}} C_1 : K_1 \qquad \langle\Gamma\rangle\,\langle k : \neg C_0\rangle \vdash_{\mathsf{cnt}} \mathcal{K} : \neg C_2\{z := C_1\}}{\langle\Gamma\rangle\,\langle k : \neg C_0\rangle \vdash_{\mathsf{cnt}} (\lambda y{:}(\Pi z{:}K_1.\,\neg\neg C_2).\,y\,C_1\,\mathcal{K}) : \neg\Pi z{:}K_1.\,\neg\neg C_2}$$

if $y \notin \Gamma$ and $y \in \mathit{Computation\text{-}ident}[\mathsf{CPS}]$

Figure 6: CPS LEGAL OBJECTS

Constructors

$$\frac{\Gamma \vdash_{\mathsf{knd}} K : \Box}{\Gamma, z{:}K \vdash_{\mathsf{con}} z : K} \quad \text{if } z \notin \Gamma \text{ and } z \in \mathit{Constructor\text{-}ident}[\mathsf{CPS}]$$

$$\frac{\Gamma, x{:}\neg\neg C_1 \vdash_{\mathsf{con}} C_2 : K_2 \qquad \Gamma \vdash_{\mathsf{knd}} \Pi x{:}\neg\neg C_1.\, K_2 : \Box}{\Gamma \vdash_{\mathsf{con}} \lambda x{:}\neg\neg C_1.\, C_2 : \Pi x{:}\neg\neg C_1.\, K_2}$$

$$\frac{\Gamma, z{:}K_1 \vdash_{\mathsf{con}} C_2 : K_2 \qquad \Gamma \vdash_{\mathsf{knd}} \Pi z{:}K_1.\, K_2 : \Box}{\Gamma \vdash_{\mathsf{con}} \lambda z{:}K_1.\, C_2 : \Pi z{:}K_2.\, K_2}$$

$$\frac{\Gamma \vdash_{\mathsf{con}} C_0 : \Pi x{:}\neg\neg C_1.\, K_2 \qquad \Gamma \vdash_{\mathsf{com}} \lambda k{:}\neg C_1.\, \mathcal{A} : \neg\neg C_1}{\Gamma \vdash_{\mathsf{con}} C_0\ (\lambda k{:}\neg C_1.\, \mathcal{A}) : K_2\{x := \lambda k{:}\neg C_1.\, \mathcal{A}\}}$$

$$\frac{\Gamma \vdash_{\mathsf{con}} C_0 : \Pi z{:}K_1.\, K_2 \qquad \Gamma \vdash_{\mathsf{com}} C_1 : K_1}{\Gamma \vdash_{\mathsf{con}} C_0\, C_1 : K_2\{z := C_1\}}$$

$$\frac{\Gamma \vdash_{\mathsf{con}} C_1 : * \qquad \Gamma, x{:}\neg\neg C_1 \vdash_{\mathsf{con}} C_2 : *}{\Gamma \vdash_{\mathsf{con}} \Pi x{:}\neg\neg C_1.\, \neg\neg C_2 : *}$$

$$\frac{\Gamma \vdash_{\mathsf{con}} K_1 : \Box \qquad \Gamma, z{:}K_1 \vdash_{\mathsf{con}} C_2 : *}{\Gamma \vdash_{\mathsf{con}} \Pi z{:}K_1.\, \neg\neg C_2 : *}$$

Kinds

$$\bot : * \vdash_{\mathsf{knd}} * : \Box$$

$$\frac{\Gamma \vdash_{\mathsf{con}} C : * \qquad \Gamma, x{:}\neg\neg C \vdash_{\mathsf{knd}} K : \Box}{\Gamma \vdash_{\mathsf{knd}} \Pi x{:}\neg\neg C.\, K : \Box}$$

$$\frac{\Gamma \vdash_{\mathsf{knd}} K_1 : \Box \qquad \Gamma, z{:}K_1 \vdash_{\mathsf{knd}} K_2 : \Box}{\Gamma \vdash_{\mathsf{knd}} \Pi z{:}K_1.\, K_2 : \Box}$$

Figure 7: CPS legal constructors and kinds

Weakening

$$\frac{\Gamma \vdash_\theta A : B \qquad \Gamma \vdash_{\mathsf{knd}} K : \Box}{\Gamma, z{:}K \vdash_\theta A : B} \quad \text{if } z \notin \Gamma \text{ and } z \in \mathit{Constructor\text{-}ident}[\mathsf{CPS}]$$

$$\frac{\langle\Gamma\rangle\,\langle k{:}\neg C_0\rangle \vdash_\phi A : B \qquad \Gamma \vdash_{\mathsf{knd}} K : \Box}{\langle\Gamma, z{:}K\rangle\,\langle k{:}\neg C_0\rangle \vdash_\phi A : B} \quad \text{if } z \notin \Gamma \text{ and } z \in \mathit{Constructor\text{-}ident}[\mathsf{CPS}]$$

$$\frac{\Gamma \vdash_\theta A : B \qquad \Gamma \vdash_{\mathsf{con}} C : *}{\Gamma, x{:}\neg\neg C \vdash_\theta A : B} \quad \text{if } x \notin \Gamma \text{ and } x \in \mathit{Computation\text{-}ident}[\mathsf{CPS}]$$

$$\frac{\langle\Gamma\rangle\,\langle k{:}\neg C_0\rangle \vdash_\phi A : B \qquad \Gamma \vdash_{\mathsf{knd}} K : \Box}{\langle\Gamma, x{:}\neg\neg C\rangle\,\langle k{:}\neg C_0\rangle \vdash_\phi A : B} \quad \text{if } x \notin \Gamma \text{ and } x \in \mathit{Computation\text{-}ident}[\mathsf{CPS}]$$

Conversion: (excerpts)

$$\frac{\Gamma \vdash_{\mathsf{con}} C : K \qquad \Gamma \vdash_{\mathsf{knd}} K' : \Box}{\Gamma \vdash_{\mathsf{con}} C : K'} \quad \text{if } K =_\beta K'$$

$$\frac{\Gamma \vdash_{\mathsf{com}} \mathcal{M} : \neg\neg C \qquad \Gamma \vdash_{\mathsf{con}} C' : *}{\Gamma \vdash_{\mathsf{com}} \mathcal{M} : \neg\neg C'} \quad \text{if } C =_\beta C'$$

$$\begin{array}{lcl} \theta & \in & \{com, val, arg, con, knd\} \\ \phi & \in & \{ans, cnt\} \end{array}$$

Figure 8: CPS weakening and conversion rules

Property 2

$$\begin{array}{rcllcl} \Gamma \vdash_\theta A : B & \Rightarrow & \Gamma \vdash A : B & \theta & \in & \{com, val, arg, con, knd\} \\ \langle\Gamma\rangle\,\langle k{:}\neg C\rangle \vdash_\phi A : B & \Rightarrow & \Gamma, k{:}\neg C \vdash A : B & \phi & \in & \{ans, cnt\} \end{array}$$

The following property states that the domain-free CPS language does indeed contain the image of the CPS translation.

Property 3

$$\begin{array}{rcl} \Gamma \vdash O : C & \Rightarrow & \mathcal{C}\{\!\!\{\Gamma\}\!\!\} \Vdash_{\mathsf{com}} \mathcal{C}\langle O\rangle : \mathcal{C}\{\!\!\{C\}\!\!\} \\ \Gamma \vdash C : K & \Rightarrow & \mathcal{C}\{\!\!\{\Gamma\}\!\!\} \Vdash_{\mathsf{con}} \mathcal{C}\langle C\rangle : \mathcal{C}\{\!\!\{K\}\!\!\} \\ \Gamma \vdash K : \Box & \Rightarrow & \mathcal{C}\{\!\!\{\Gamma\}\!\!\} \Vdash_{\mathsf{knd}} \mathcal{C}\langle K\rangle : \mathcal{C}\{\!\!\{\Box\}\!\!\} \end{array}$$

The following property states that legal terms in the CPS language are closed under reduction.

Property 4

$$\begin{array}{rcllcl} \Gamma \vdash_\theta A : B \;\; and \;\; A \rightarrow_\beta A' & \Rightarrow & \Gamma \vdash_\theta A' : B & \theta & \in & \{com, val, arg, con, knd\} \\ \langle\Gamma\rangle\,\langle k{:}\neg C\rangle \vdash_\phi A : B \;\; and \;\; A \rightarrow_\beta A' & \Rightarrow & \langle\Gamma\rangle\,\langle k{:}\neg C\rangle \vdash_\phi A' : B & \phi & \in & \{ans, cnt\} \end{array}$$

4.4 An Optimizing CPS Translation

Reducing a CPS term involves contracting many *administrative redexes* — redexes involving administrative abstractions [70, p. 149]. We write $A \rightarrow_{adm} A'$ when A β-reduces to A' by contracting administrative reductions only.

It is often useful to define an optimizing version of a particular CPS translation that produces terms with fewer administrative redexes [20, 37, 70, 76, 83]. In this section, we present an optimizing version of the translation in Figure 4 that yields terms in *administrative normal-form* (i.e., the terms contain no administrative redexes). We need to consider the CPS translation of objects only, since continuations are introduced only in this category.

The optimizing translation manipulates object call-by-name *evaluation contexts* [34] defined by the following grammar.

$$\begin{array}{rcl} E & \in & \textit{Evaluation-Contexts}[\mathsf{CUBE}] \\ E & ::= & [\cdot] \;\mid\; E\,O \;\mid\; E\,C \end{array}$$

Each object can be uniquely factored into an evaluation context and an abstraction or variable.

Property 5 (Unique decomposition) *For all objects $O \in$ Pseudo-Obj[CUBE], there exists a unique evaluation context $E \in$ Evaluation-Contexts[CUBE] and unique term $A \in$ Pseudo-Terms[CUBE] such that $O \equiv E[A]$ where A is one of the following: V^*, $\lambda V^*{:}C.\,O$, $\lambda V^\Box{:}K.\,C$.*

Proof: by a simple induction over the structure of O. ∎

We write $O \equiv! E[A]$ to denote the unique factoring of object O into evaluation context E and term A.

Figure 9 presents the optimizing translation. $E : A : \mathcal{K}$ represents the translation of an object uniquely factored into E and A where $\mathcal{K}$ is a continuation corresponding to the evaluation context in which $E[A]$ appeared. The translation proceeds recursively over E — transforming it into a continuation with no administrative redexes. Depending on the form of A, the resulting continuation is either passed as an argument to the translation of A (if A is a variable), or applied to the translation of A (if A is an abstraction). We only present the translation on objects; translation of the remaining syntactic categories is the same as in Figure 4.

Lemma 3 [optimizing CPS translation for the cube] For all $A \in$ *Pseudo-Terms*[CUBE],

- $\mathcal{C}\langle A\rangle \rightarrow_{adm} \mathcal{C}^+\langle A\rangle$
- $\mathcal{C}^+\langle A\rangle$ is in administrative normal-form.

$$\begin{array}{rcl}
\mathcal{C}^+\langle O\rangle & = & \lambda k.E : A : k \quad \text{where } O \equiv! E[A] \\
\\
[\cdot] : V^* : \mathcal{K} & = & V^*\,\mathcal{K} \\
[\cdot] : (\lambda V^*{:}C.O) : k & = & k\,(\lambda V^*.\mathcal{C}^+\langle O\rangle) \\
[\cdot] : (\lambda V^{\square}{:}K.O) : k & = & k\,(\lambda V^*.\mathcal{C}^+\langle O\rangle) \\
[\cdot] : (\lambda V^*{:}C.O) : (\lambda y.y\,\mathcal{N}\,\mathcal{K}) & = & (\lambda V^*.\mathcal{C}^+\langle O\rangle)\,\mathcal{N}\,\mathcal{K} \\
[\cdot] : (\lambda V^{\square}{:}K.O) : (\lambda y.y\,\mathcal{N}\,\mathcal{K}) & = & (\lambda V^{\square}.\mathcal{C}^+\langle O\rangle)\,\mathcal{N}\,\mathcal{K} \\
E\,O : A : \mathcal{K} & = & E : A : (\lambda y.y\,\mathcal{C}^+\langle O\rangle\,\mathcal{K}) \\
E\,C : A : \mathcal{K} & = & E : A : (\lambda y.y\,\mathcal{C}^+\langle C\rangle\,\mathcal{K})
\end{array}$$

Figure 9: OPTIMIZING CPS TRANSLATION FOR THE CUBE

4.5 Domain-specified CPS translation for the cube

The translation of the previous section maps domain-specified pseudo-terms to domain-free pseudo-terms. In this section, we consider a translation which maps domain-specified pseudo-terms to domain-specified pseudo-terms. The domain-specified CPS translation is defined by $\prec$-induction (as introduced in Section 2.5.

Definition 13 $\underline{\mathcal{C}}_\Gamma\langle . \rangle$: *Legal-Terms*[CUBE] $\to$ *Legal-Terms*[CUBE] is defined in Figure 10.

We prove:

Theorem 3 $\underline{\mathcal{C}}_\Gamma\langle\cdot\rangle$ *is a well-defined function. Moreover*

1. *(Thinning) If* $\Gamma \vdash M : A$ *and* Δ *legal with* $\Gamma \subseteq \Delta$ *then* $\underline{\mathcal{C}}_\Gamma\langle M\rangle =_\beta \underline{\mathcal{C}}_\Delta\langle M\rangle$.
2. *(Substitution) If* $\Gamma, x : C, \Delta \vdash M : E$ *and* $\Gamma \vdash N : C$ *then*
$$\underline{\mathcal{C}}_{\Gamma,x:C,\Delta}\langle M\rangle\ \{x := \underline{\mathcal{C}}_\Gamma\langle N\rangle\ \} \twoheadrightarrow_\beta \underline{\mathcal{C}}_{\Gamma,\Delta^\bullet}\langle M\{x := N\}\rangle$$
where $\Delta^\bullet \equiv \Delta\{x := N\}$.
3. *(Preservation of reduction) If* $\Gamma \vdash M : A$ *and* $M \to_\beta N$ *then* $\underline{\mathcal{C}}_\Gamma\langle M\rangle \twoheadrightarrow_\beta \underline{\mathcal{C}}_\Gamma\langle N\rangle$.
4. *(Preservation of derivations) If* $\Gamma \vdash M : A$ *then* $\underline{\mathcal{C}}⟪\Gamma⟫ \vdash \underline{\mathcal{C}}_\Gamma\langle M\rangle : \underline{\mathcal{C}}_\Gamma⟪A⟫$.

The domain-free and domain-specified translations are related by the following result.

Proposition 3 *If* $\Gamma \vdash M : A$, *then*

1. $|\underline{\mathcal{C}}_\Gamma\langle M\rangle\ | \equiv \mathcal{C}\langle M\rangle$,
2. $\underline{\mathcal{C}}_\Gamma\langle M\rangle\ \equiv \mathsf{lift}_s(\mathcal{C}⟪\Gamma⟫ \Vdash \mathcal{C}\langle M\rangle : \mathcal{C}⟪A⟫)$.

Proof: By induction on the structure of derivations. ∎

5 DS Translation

5.1 Domain-free DS translation for the cube

Figure 11 presents a DS translation from the domain-specified CPS language to the domain specified cube. For objects, the most interesting aspect is that continuations are translated to call-by-name evaluation contexts. The holes in the evaluation contexts are filled during the translation of CPS ans syntactic category. The translation of constructors and kinds is straightforward — double negations are removed and translation continues on substructures.

$$\begin{array}{lcll}
\underline{\mathcal{C}}_\Gamma\langle x\rangle & = & \begin{cases} \lambda k{:}\neg\underline{\mathcal{C}}^{\mathsf{nf}}_\Gamma\langle D\rangle.\ x\ k & \text{if } \Gamma\ \vdash\ x : D : \mathsf{Prop} \\ x & \text{otherwise} \end{cases} \\[2ex]
\underline{\mathcal{C}}_\Gamma\langle s\rangle & = & s \\[2ex]
\underline{\mathcal{C}}_\Gamma\langle \lambda x{:}A.\,M\rangle & = & \begin{cases} \lambda k{:}\neg\underline{\mathcal{C}}^{\mathsf{nf}}_\Gamma\langle D\rangle.\ k\ (\lambda x{:}\underline{\mathcal{C}}_\Gamma\langle A\rangle\ .\,\underline{\mathcal{C}}_{(\Gamma,x:A)}\langle M\rangle\ ) & \text{if } \Gamma\ \vdash\ \lambda x{:}A.\,M\ : D : \mathsf{Prop} \\ \lambda x{:}\underline{\mathcal{C}}_\Gamma\langle A\rangle\ .\,\underline{\mathcal{C}}_{(\Gamma,x:A)}\langle M\rangle & \text{otherwise} \end{cases} \\[2ex]
\underline{\mathcal{C}}_\Gamma\langle M\ M'\rangle & = & \begin{cases} \lambda k{:}\neg\underline{\mathcal{C}}^{\mathsf{nf}}_\Gamma\langle D\rangle.\,\underline{\mathcal{C}}_\Gamma\langle M\rangle\ \ (\lambda j{:}\neg\underline{\mathcal{C}}^{\mathsf{nf}}_\Gamma\langle D'\rangle.\ j\ \underline{\mathcal{C}}_\Gamma\langle M'\rangle\ \ k) & \text{if } \Gamma\ \vdash\ M\ M' : D : \mathsf{Prop} \\ & \text{and } \Gamma\ \vdash\ M : D' \\ \underline{\mathcal{C}}_\Gamma\langle M\rangle\ \ \underline{\mathcal{C}}_\Gamma\langle M'\rangle & \text{otherwise} \end{cases} \\[2ex]
\underline{\mathcal{C}}_\Gamma\langle \Pi x{:}A.\,B\rangle & = & \Pi x{:}\underline{\mathcal{C}}_\Gamma\{\!\!|A|\!\!\}.\,\underline{\mathcal{C}}_{(\Gamma,x:A)}\{\!\!|B|\!\!\} \\[3ex]
\underline{\mathcal{C}}_\Gamma\{\!\!|M|\!\!\} & = & \begin{cases} \neg\neg\underline{\mathcal{C}}_\Gamma\langle M\rangle & \text{if } \Gamma\ \vdash\ M : \mathsf{Prop} \\ \underline{\mathcal{C}}_\Gamma\langle M\rangle & \text{otherwise} \end{cases} \\[3ex]
\underline{\mathcal{C}}^{\mathsf{nf}}_\Gamma\langle M\rangle & = & \underline{\mathcal{C}}_\Gamma\langle \mathsf{nf}\ M\rangle \\[3ex]
\underline{\mathcal{C}}\langle [\,]\rangle & = & [\,] \\
\underline{\mathcal{C}}\langle \Gamma, x : A\rangle & = & \underline{\mathcal{C}}\langle\Gamma\rangle\ , x : \underline{\mathcal{C}}_\Gamma\langle A\rangle \\[3ex]
\underline{\mathcal{C}}\{\!\!|\Gamma|\!\!\} & = & \bot : \mathsf{Prop}, \underline{\mathcal{C}}\langle\Gamma\rangle
\end{array}$$

Figure 10: Domain-specified CPS translation

In contrast to the CPS translation, the DS translation can be defined by induction over the structure of pseudo-terms even when mapping to domain-specified systems. Since no abstractions (e.g., abstractions analogous to the administrative abstractions of the CPS translations) are introduced during the DS translation, all required domain tags can be constructed by translating tags appearing on abstractions in the argument of the translation.

Theorem 4 [Correctness of DS translation]

$$\Gamma \vdash_\theta A : B \quad \Rightarrow \quad \mathcal{D}\{\!\![\Gamma]\!\!\} \vdash \mathcal{D}\langle A\rangle : \mathcal{D}\{\!\![B]\!\!\}$$

for $\theta \in \{com, con, knd\}$.

5.2 Interaction between CPS and DS translations

Since β-conversion is the principle notion of equality in the cube, we now consider the interaction of the translations up to this notion of equality. The following theorem states an equational correspondence (as presented by Sabry and Felleisen [76]) between direct style terms and the CPS language. Here, we consider only translations between domain-free languages. We have not checked all the proofs for the domain-specified case, but we expect no difficulties.

Theorem 5 *Let* $\Gamma \Vdash A, A_1, A_2 : B$ *and* $\Sigma \Vdash_\theta P, P_1, P_2 : Q$ *for* $\theta \in \{com, con, knd\}$.

1. $A =_{\underline{\beta}} \mathcal{D}\langle\mathcal{C}\langle A\rangle\rangle$
2. $P =_{\underline{\beta}} \mathcal{C}\langle\mathcal{D}\langle P\rangle\rangle$
3. $A_1 =_{\underline{\beta}} A_2$ *iff* $\mathcal{C}\langle A_1\rangle =_{\underline{\beta}} \mathcal{C}\langle A_2\rangle$
4. $P_1 =_{\underline{\beta}} P_2$ *iff* $\mathcal{D}\langle P_1\rangle =_{\underline{\beta}} \mathcal{D}\langle P_2\rangle$

Components 3 and 4 of this theorem can be strengthened to a correspondence of *reductions*. This is noteworthy since the applications we have in mind often require relating DS and CPS reductions.

6 Beyond the Cube

In this section, we extend the CPS translations for the cube to larger classes of specifications. In a first instance, we consider a class of pure type systems for which the context-independent translation of the Section 4.2 may be applied directly. Later, we consider a larger class of specifications for which the translation is defined relative to a context. Although we only consider pure CPS translations, it is also possible to extend the optimized CPS translations and direct-style translations to the class of pure type systems considered.

6.1 Context-independent translation

In this subsection, we present the CPS translation for those strict logical specifications with a Classification Property. Throughout this subsection, we assume that $\mathbf{S} = (S, A, R, \mathsf{Prop})$ is a fixed strict logical specification with the Classification Property.

Definition 14 The context-independent domain-free CPS translation is defined in Figure 12.

Following the method of Theorem 2, one proves:

Theorem 6

$$\Gamma \vdash A : B \quad \Rightarrow \quad \mathcal{C}\{\!\![\Gamma]\!\!\} \Vdash \mathcal{C}\langle A\rangle : \mathcal{C}\{\!\![B]\!\!\}$$

Objects

$$
\begin{array}{rcl}
\mathcal{D}\langle x\rangle_{\mathrm{com}} & = & x \\
\mathcal{D}\langle \lambda k\colon \neg C.\, \mathcal{A}\rangle_{\mathrm{com}} & = & \mathcal{D}\langle \mathcal{A}\rangle_{\mathrm{ans}} \\
\mathcal{D}\langle \mathcal{V}\ \mathcal{N}\rangle_{\mathrm{com}} & = & \mathcal{D}\langle \mathcal{V}\rangle_{\mathrm{val}}\ \mathcal{D}\langle \mathcal{N}\rangle_{\mathrm{arg}} \\
\\
\mathcal{D}\langle \lambda x\colon \neg\neg C_1.\, \lambda k\colon \neg C_2.\, \mathcal{A}\rangle_{\mathrm{val}} & = & \lambda x\colon \mathcal{D}\langle C_1\rangle_{\mathrm{con}}.\, \mathcal{D}\langle \lambda k\colon \neg C_2.\, \mathcal{A}\rangle_{\mathrm{com}} \\
\mathcal{D}\langle \lambda z\colon K.\, \lambda k\colon \neg C_2.\, \mathcal{A}\rangle_{\mathrm{val}} & = & \lambda z\colon \mathcal{D}\langle K\rangle_{\mathrm{knd}}.\, \mathcal{D}\langle \lambda k\colon \neg C_2.\, \mathcal{A}\rangle_{\mathrm{com}} \\
\\
\mathcal{D}\langle \lambda k\colon \neg C.\, \mathcal{A}\rangle_{\mathrm{arg}} & = & \mathcal{D}\langle \lambda k\colon \neg C.\, \mathcal{A}\rangle_{\mathrm{com}} \\
\mathcal{D}\langle C\rangle_{\mathrm{arg}} & = & \mathcal{D}\langle C\rangle_{\mathrm{con}} \\
\\
\mathcal{D}\langle \mathcal{K}\ \mathcal{V}\rangle_{\mathrm{ans}} & = & \mathcal{D}\langle \mathcal{K}\rangle_{\mathrm{cnt}}[\mathcal{D}\langle \mathcal{V}\rangle_{\mathrm{val}}] \\
\mathcal{D}\langle \mathcal{M}\ \mathcal{K}\rangle_{\mathrm{ans}} & = & \mathcal{D}\langle \mathcal{K}\rangle_{\mathrm{cnt}}[\mathcal{D}\langle \mathcal{M}\rangle_{\mathrm{com}}] \\
\\
\mathcal{D}\langle k\rangle_{\mathrm{cnt}} & = & [\cdot] \\
\mathcal{D}\langle \lambda y\colon C.\, y\ \mathcal{N}\ \mathcal{K}\rangle_{\mathrm{cnt}} & = & \mathcal{D}\langle \mathcal{K}\rangle_{\mathrm{cnt}}[[\cdot]\ \mathcal{D}\langle \mathcal{N}\rangle_{\mathrm{arg}}]
\end{array}
$$

Constructors

$$
\begin{array}{rcl}
\mathcal{D}\langle z\rangle_{\mathrm{con}} & = & z \\
\mathcal{D}\langle \lambda x\colon \neg\neg C_1.\, C_2\rangle_{\mathrm{con}} & = & \lambda x\colon \mathcal{D}\langle C_1\rangle_{\mathrm{con}}.\, \mathcal{D}\langle C_2\rangle_{\mathrm{con}} \\
\mathcal{D}\langle C_0\ (\lambda k\colon \neg C_1.\, \mathcal{A})\rangle_{\mathrm{con}} & = & \mathcal{D}\langle C_0\rangle_{\mathrm{con}}\ \mathcal{D}\langle \lambda k\colon \neg C_1.\, \mathcal{A}\rangle_{\mathrm{com}} \\
\mathcal{D}\langle \lambda z\colon K.\, C\rangle_{\mathrm{con}} & = & \lambda z\colon \mathcal{D}\langle K\rangle_{\mathrm{knd}}.\, \mathcal{D}\langle C\rangle_{\mathrm{con}} \\
\mathcal{D}\langle C_1\ C_2\rangle_{\mathrm{con}} & = & \mathcal{D}\langle C_1\rangle_{\mathrm{con}}\ \mathcal{D}\langle C_2\rangle_{\mathrm{con}} \\
\mathcal{D}\langle \Pi x\colon \neg\neg C_1.\, \neg\neg C_2\rangle_{\mathrm{con}} & = & \Pi x\colon \mathcal{D}\langle C_1\rangle_{\mathrm{con}}.\, \mathcal{D}\langle C_2\rangle_{\mathrm{con}} \\
\mathcal{D}\langle \Pi z\colon K.\, \neg\neg C\rangle_{\mathrm{con}} & = & \Pi z\colon \mathcal{D}\langle K\rangle_{\mathrm{knd}}.\, \mathcal{D}\langle C\rangle_{\mathrm{con}}
\end{array}
$$

Kinds

$$
\begin{array}{rcl}
\mathcal{D}\langle \Pi x\colon \neg\neg C.\, K\rangle_{\mathrm{knd}} & = & \Pi x\colon \mathcal{D}\langle C\rangle_{\mathrm{con}}.\, \mathcal{D}\langle K\rangle_{\mathrm{knd}} \\
\mathcal{D}\langle \Pi z\colon K_1.\, K_2\rangle_{\mathrm{knd}} & = & \Pi z\colon \mathcal{D}\langle K_1\rangle_{\mathrm{knd}}.\, \mathcal{D}\langle K_2\rangle_{\mathrm{knd}} \\
\mathcal{D}\langle *\rangle_{\mathrm{knd}} & = & *
\end{array}
$$

Contexts

$$
\begin{array}{rcl}
\mathcal{D}\langle \bot : *\rangle & = & \cdot \\
\mathcal{D}\langle \Gamma, x : \neg\neg C\rangle & = & \mathcal{D}\langle \Gamma\rangle, x : \mathcal{D}\langle C\rangle_{\mathrm{con}} \\
\mathcal{D}\langle \Gamma, z : K\rangle & = & \mathcal{D}\langle \Gamma\rangle, z : \mathcal{D}\langle K\rangle_{\mathrm{knd}}
\end{array}
$$

Top-level translation

$$
\begin{array}{rcl}
\mathcal{D}\llparenthesis O\rrparenthesis & = & \mathcal{D}\langle O\rangle \\
\mathcal{D}\llparenthesis \neg\neg C\rrparenthesis & = & \mathcal{D}\langle C\rangle \\
\mathcal{D}\llparenthesis K\rrparenthesis & = & \mathcal{D}\langle K\rangle \\
\mathcal{D}\llparenthesis \Box\rrparenthesis & = & \Box \\
\mathcal{D}\llparenthesis \Gamma\rrparenthesis & = & \mathcal{D}\langle \Gamma\rangle
\end{array}
$$

Figure 11: DS translation for the cube

$$\mathcal{C}\langle x\rangle = \begin{cases} \lambda k.x\,k & \text{if } x \in V^{\mathsf{Prop}} \\ x & \text{otherwise} \end{cases}$$

$$\mathcal{C}\langle s\rangle = s$$

$$\mathcal{C}\langle \lambda x{:}A.\,M\rangle = \begin{cases} \lambda k.k\,(\lambda x.\mathcal{C}\langle M\rangle) & \text{if } \lambda x{:}A.\,M \in \mathsf{Proof} \\ \lambda x.\mathcal{C}\langle M\rangle & \text{otherwise} \end{cases}$$

$$\mathcal{C}\langle M\,M'\rangle = \begin{cases} \lambda k.\mathcal{C}\langle M\rangle\,(\lambda j.j\;\mathcal{C}\langle M'\rangle\;k) & \text{if } M\,M' \in \mathsf{Proof} \\ \mathcal{C}\langle M\rangle\,\mathcal{C}\langle M'\rangle & \text{otherwise} \end{cases}$$

$$\mathcal{C}\langle \Pi x{:}A.\,B\rangle = \Pi x{:}\mathcal{C}\langle\!\langle A\rangle\!\rangle.\,\mathcal{C}\langle\!\langle B\rangle\!\rangle$$

$$\mathcal{C}\langle\!\langle M\rangle\!\rangle = \begin{cases} \neg\neg\mathcal{C}\langle M\rangle & \text{if } M \in \mathsf{Prop} \\ \mathcal{C}\langle M\rangle & \text{otherwise} \end{cases}$$

$$\mathcal{C}\langle [\,]\rangle = [\,]$$
$$\mathcal{C}\langle \Gamma, x : A\rangle = \mathcal{C}\langle\Gamma\rangle, x : \mathcal{C}\langle A\rangle$$

$$\mathcal{C}\langle\!\langle\Gamma\rangle\!\rangle = \bot : \mathsf{Prop}, \mathcal{C}\langle\Gamma\rangle$$

Figure 12: Context-independent domain-free CPS translation

6.2 Context-dependent translation

Although the classification property is useful for defining the domain-free CPS translation, it is not absolutely necessary. One may extend the CPS translation to the whole class of strict logical specifications provided we take care to define the translation relative to a pseudo-context. Throughout this section, we assume that $\mathbf{S} = (S, A, R, \mathsf{Prop})$ is a fixed strict logical specification.

Definition 15 The domain-free CPS translation is defined in Figure 13.

Following the method of Theorem 2, one proves:

Theorem 7

$$\Gamma \vdash A : B \quad\Rightarrow\quad \mathcal{C}\langle\!\langle\Gamma\rangle\!\rangle \Vdash \mathcal{C}_\Gamma\langle A\rangle : \mathcal{C}_\Gamma\langle\!\langle B\rangle\!\rangle$$

For those specifications with the classification property, the context-independent and context-dependent translations coincide on legal terms.

Lemma 4 *If* $\mathbf{S}$ *has the classification property, then* $\mathcal{C}_\Gamma\langle M\rangle \equiv \mathcal{C}\langle M\rangle$ *whenever* $\Gamma \vdash M : A$.

Proof: By induction on the structure of derivations. ∎

6.3 Domain-specified translation

In Section 4.5, we presented a domain-specified translation for systems of the cube. The question naturally arises whether it is possible to generalize this translation for an arbitrary strict logical specification. It is easy to see that the domain-specified translation may be extended to such specifications provided $\prec$ is well-defined and well-founded.

$$\mathcal{C}_\Gamma\langle x\rangle = \begin{cases} \lambda k.x\,k & \text{if } \Gamma \vdash x : D : \mathsf{Prop} \\ x & \text{otherwise} \end{cases}$$

$$\mathcal{C}_\Gamma\langle s\rangle = s$$

$$\mathcal{C}_\Gamma\langle \lambda x{:}A.\,M\rangle = \begin{cases} \lambda k.k\;(\lambda x.\mathcal{C}_{(\Gamma,x:A)}\langle M\rangle) & \text{if } \Gamma \vdash \lambda x{:}A.\,M : D : \mathsf{Prop} \\ \lambda x{:}\mathcal{C}_\Gamma\langle A\rangle.\,\mathcal{C}_{(\Gamma,x:A)}\langle M\rangle & \text{otherwise} \end{cases}$$

$$\mathcal{C}_\Gamma\langle M\,M'\rangle = \begin{cases} \lambda k.\mathcal{C}_\Gamma\langle M\rangle\;(\lambda j.j\;\mathcal{C}_\Gamma\langle M'\rangle\;k) & \text{if } \Gamma \vdash M\,M' : D : \mathsf{Prop} \text{ and } \Gamma \vdash M : D' \\ \mathcal{C}_\Gamma\langle M\rangle\,\mathcal{C}_\Gamma\langle M'\rangle & \text{otherwise} \end{cases}$$

$$\mathcal{C}_\Gamma\langle \Pi x{:}A.\,B\rangle = \Pi x{:}\mathcal{C}_\Gamma\langle\!\langle A\rangle\!\rangle.\,\mathcal{C}_{\Gamma,x:A}\langle\!\langle B\rangle\!\rangle$$

$$\mathcal{C}_\Gamma\langle\!\langle M\rangle\!\rangle = \begin{cases} \neg\neg\mathcal{C}_\Gamma\langle M\rangle & \text{if } \Gamma \vdash M : \mathsf{Prop} \\ \mathcal{C}_\Gamma\langle M\rangle & \text{otherwise} \end{cases}$$

$$\begin{aligned} \mathcal{C}\langle[]\rangle &= [] \\ \mathcal{C}\langle\Gamma, x : A\rangle &= \mathcal{C}\langle\Gamma\rangle, x : \mathcal{C}_\Gamma\langle A\rangle \end{aligned}$$

$$\mathcal{C}\langle\!\langle\Gamma\rangle\!\rangle = \bot : \mathsf{Prop}, \mathcal{C}\langle\Gamma\rangle$$

Figure 13: CONTEXT-DEPENDENT DOMAIN-FREE CPS TRANSLATION

7 Applications

In this section we sketch three areas for applications of generalized CPS translations: infering strong normalization from weak normalization in classes of PTSs, construction of looping combinators in some inconsistent PTSs, and embedding classical PTSs in PTSs. We do not develop the applications as such, merely suggest how they may be undertaken.

7.1 Strong normalization from weak normalization

A PTS is *weakly normalizing* if, from every legal term, there is at least one finite reduction sequence ending in a normal form. A PTS is *strongly normalizing* if there is no legal term with an infinite reduction sequence. The latter property trivially implies the former, but the converse is not obvious when it holds.

The classical proof of strong normalization for β-reduction in simply typed λ-calculus is due to Tait [86]. It was generalized to second-order typed λ-calculus by Girard [41], and subsequently simplified by Tait [87]. It has since been generalized to a variety of λ-calculi, see [8, 38, 43, 50, 56, 88].

For notions of reduction in some typed λ-calculi there is a technique to prove weak normalization that is simpler than the Tait-Girard technique to prove strong normalization. For instance, Turing [39] proves weak normalization for β-reduction in simply typed λ-calculus by giving an explicit measure which decreases in every step of a certain β-reduction sequence. Prawitz [71] independently uses the same technique to prove weak normalization for reduction of natural deduction derivations in predicate logic.

Nederpelt [63], Klop [55], Khasidashvili [54], Karr [52], de Groote [26], and Kfoury and Wells [53] have invented techniques to infer strong normalization from weak normalization. However, these techniques all infer strong normalization of one notion of reduction from weak normalization of a *more complicated* notion of reduction. Sørensen [85] and Xi [91] recently developed techniques which infer strong normalization of some notion of reduction in a typed λ-calculus from weak normalization of the *same* notion of reduction.

These techniques provide some hope for a positive answer to a conjecture, presented by Barendregt at *Typed Lambda-Calculus and Applications, Edinburgh 1995*, stating that for every pure type system [8] weak normalization of β-reduction implies strong normalization of β-reduction. The conjecture has also been mentioned by Geuvers [40], and, in a less concrete form, by Klop.

To explain these recent techniques we consider simply typed λ-calculus á la Curry. The main idea is to modify Plotkin's CPS-translation (see Fig. 3) by changing the clause for abstractions into:

$$\mathcal{C}\langle\lambda x.M\rangle \quad = \quad \lambda k.k\,\lambda x.\lambda h.y\,(\mathcal{C}\langle P\rangle\;h)\;x$$

For every abstraction $\lambda x.Q$ in $\mathcal{C}\langle M\rangle$, x occurs free in Q. By the Conservation Theorem for λI (see [7]) weak normalization of $\mathcal{C}\langle P\rangle$ implies strong normalization of $\mathcal{C}\langle P\rangle$. Moreover, one can show that strong normalization of $\mathcal{C}\langle P\rangle$ implies strong normalization of P. Now, one can also show that the translation maps simply typable terms to simply typable terms. From this it follows that weak normalization of all typable terms implies strong normalization of all typable terms.

Thus far the technique has not been applied to systems more powerful than $\lambda\omega$, partly because CPS translations for more powerful systems has not been known.

It would be interesting to use our generalized CPS-translations to extend the above technique to more powerful systems.

7.2 Classical pure type systems

The *Curry-Howard isomorphism* [17, 51] states a correspondence between constructive logics and typed functional calculi, and reflects an old idea that proofs in formal logics are certain functions and objects. The isomorphism has evolved with the invention of numerous typed λ-calculi and corresponding natural deduction logics—see [8, 40]. Until the the late 1980's, the Curry-Howard isomorphism was concerned exclusively with constructive logics. At that time Griffin [44] discovered that Felleisen's [33, 35] control operator $\mathcal{C}$ could be meaningfully added to the simply typed λ-calculus by typing $\mathcal{C}$ with the double negation rule. The reduction rules for $\mathcal{C}$ are related to well-known reductions on classical proofs [72, 79, 84]. Moreover, Griffin discovered that well-known embeddings of classical logic in intuitionistic logics corresponds to CPS-translations.

Griffin's discovery was followed by several lines of work on classical logic, control operators, and the Curry-Howard isomorphism—some initiated independently of his work. It is not possible here to explain the aims and achievements of the individual lines of work; it must suffice simply to mention the work of Murthy [14, 59, 60, 61, 62], Barbanera and Berardi [2, 3, 4, 5, 6], Rezus [74, 75], Parigot [66, 67, 68, 69], de Groote [25, 27, 28, 29, 30], Krivine [57], Girard [42], Danos, Joinet, and Schellinx [18], Rehof and Sørensen [73], Duba, Harper, and MacQueen[32], Harper and Lillibridge [46, 47], Coquand [15], Berardi, Bezem, and Coquand [11], Ong [65], Underwood [89], and Sato [78]. Most of these lines of work study one specific classical natural deduction system or one specific typed λ-calculus enriched with control operators.

The authors [10] study a notion of classical pure type system (CPTS), systematizing such calculi similarly to how PTSs systematize typed λ-calculi. In order to study embeddings of CPTSs into PTSs, in general, the need arises for more general CPS-translations. Applications of CPS-translations in this connection would include proof of consistency of a classical calculus from that of the corresponding constructive calculus, as well as results concerning normalization (see [90]) and conservativity.

It would be interesting to develop these applications using our generalized CPS-translations.

7.3 Looping combinators

A looping combinator of sort s in a PTS is a term of type $\Pi\alpha : s.(\alpha \to \alpha) \to \alpha$ with the same Böhm tree as the usual fixed point combinator Y.

Definition 16 *A* looping combinator *of sort s in a PTS is a term of type $\Pi\alpha : s.(\alpha \to \alpha) \to \alpha$ s.t. there exists a sequence $(Y_n)_{n\in\mathbf{N}}$ of type $\Pi\alpha : s.(\alpha \to \alpha) \to \alpha$ satisfying*

$$(Y_n\;A\;f) =_\beta f\;(Y_{n+1}\;A\;f)$$

for every $A : s, f : A \to A$.

Coquand and Herbelin [16] show how one can extract looping combinators from logical, nondependent, inconsistent PTSs, using a CPS-translation which works on such systems. This result can be used to show that type checking is undecidable in some PTSs.

It would be interesting to extend their results to the class of inconsistent, strict, logical PTSs.

8 Conclusion

We have presented initial results of an investigation of CPS translations for pure type systems.

The status of the domain-specified translation needs to be investigated further. More precisely, we need to determine exactly the strength of the assumption that $\prec$ is well-founded. Indeed, we are currently unaware of a proof of the well-foundedness of $\prec$ which does not rely on the pure type system to be strongly normalizing. As a result, the domain-specified translation seems to have, in the current state of affairs, little interest. It would be interesting to determine whether one could prove $\prec$ to be well-founded for an arbitrary pure type system with normalizing types. If so, the domain-specified translation would become useful in several applications, including "weak normalization implies strong normalization".

The techniques introduced here can be used to define other CPS translations. For example, it is easy to envision a call-by-value CPS translation analogous to the call-by-name translation in Figure 4, or even a generic staging through a monadic metalanguage as presented by Hatcliff and Danvy [48]. However, the call-by-value translation seems to only make sense if one considers pure type systems where the notion of equality is based on β_v-equality instead of β-equality. It is unclear what the applications would be for type systems with this stronger notion of equality.

The usefulness of call-by-value translations may depend on whether one considers dependent systems or simply non-dependent systems. For example, Harper and Lillibridge consider call-by-value reduction in their treatment of $\lambda\omega$ with control operators [46]. Since $\lambda\omega$ is non-dependent (and thus e.g., terms do not occur in types), their retention of the usual β-conversion as the notion of equality on constructors and kinds, while using another notion of equality on terms (e.g., equality based on β_v-conversion) makes sense. In dependent systems (where terms can occur in types), the implications are less clear. Given the classification scheme of Section 3, one might imagine a hybrid notion of equality where object redexes are contracted following the β_v rule, while all others are contracted according to the β rule.

Fischer-style CPS translations (where continuations are the first arguments to functions) may also be worth exploring in the pure type system context. As Sabry and Felleisen [76] illustrate, the CPS terms produced by these translations have slightly different reduction properties than those produced by Plotkin-style translations. This may be of use in applications such as "weak normalization implies strong normalization" where a tight correspondence of reductions is crucial to the result.

The applications sketched above give obvious directions for future work which may clarify these issues further.

References

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

[2] F. Barbanera and S. Berardi. A constructive valuation interpretation for classical logic and its use in witness extraction. In *Colloquium on Trees in Algebra and Programming*, volume 581 of *Lecture Notes in Computer Science.* Springer-Verlag, 1992.

[3] F. Barbanera and S. Berardi. Continuations and simple types: A strong normalization result. In *ACM SIGPLAN Workshop on Continuations*, 1992.

[4] F. Barbanera and S. Berardi. Extracting constructive content from classical proofs via control-like reductions. In Bezem and Groote [12].

[5] F. Barbanera and S. Berardi. Witness extraction in classical logic through normalization. In G. Huet and G. Plotkin, editors, *Logical Environments.* Cambridge University Press, 1993.

[6] F. Barbanera and S. Berardi. A symmetric lambda calculus for "classical" program extraction. In M. Hagiya and J.C. Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 495–515. Springer-Verlag, 1994.

[7] H.P. Barendregt. *The Lambda Calculus — Its Syntax and Semantics.* North-Holland, 1984.

[8] H.P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, pages 117–309. Oxford Science Publications, 1992. Volume 2.

[9] G. Barthe and M.H. Sørensen. Domain-free pure type systems. Manuscript, 1996.

[10] G. Barthe, J. Hatcliff, and M.H. Sørensen. Classical pure type systems. Manuscript, 1996.

[11] S. Berardi, M. Bezem, and T. Coquand. A realization of the negative interpretation of the axiom of choice. In Dezani-Ciancaglini and Plotkin [31], pages 32–46.

[12] M. Bezem and J.F. Groote, editors. *Typed Lambda Calculus and Applications*, volume 664 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[13] C. Consel and O. Danvy. For a better support of static data flow. In J. Hughes, editor, *Conference on Functional Programming and Computer Architecture*, volume 523 of *Lecture Notes in Computer Science*, pages 495–519. Springer-Verlag, 1991.

[14] R. Constable and C.R. Murthy. Finding computational contents in classical proofs. In G. Huet and G. Plotkin, editors, *Proceedings of the First Workshop on Logical Frameworks*, pages 341–362. Cambridge University Press, 1990.

[15] T Coquand. A semantics of evidence of classical arithmetic. *Journal of Symbolic Logic*, 60:230–260, 1995.

[16] T. Coquand and H. Herbelin. A-translation and looping combinators in pure type systems. *Journal of Functional Programming*, 4(1):77–88, 1994.

[17] H.B. Curry and R Feys. *Combinatory Logic*. North-Holland, 1958.

[18] V. Danos, J.-B. Joinet, and H. Schellinx. A new deconstructive logic: Linear logic. In J.-Y. et al. Girard, editor, *Advances in Linear Logic*. Cambridge University Press, 1994.

[19] O. Danvy. Back to direct style. *Science of Computer Programming*, 1993. Special issue on ESOP'92, the Fourth European Symposium on Programming, Rennes, February 26-28, 1992. To appear.

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

[21] O. Danvy and J. Lawall. Back to direct style II: First-class continuations. In William Clinger, editor, *Proceedings of the 1992 ACM Conference on Lisp and Functional Programming*, LISP Pointers, Vol. V, No. 1, pages 299–310, San Francisco, California, June 1992. ACM Press.

[22] G. Dowek. A second-order pattern matching algorithm for the cube of typed lambda-calculi. In A. Tarlecki, editor, *Proceedings of MFCS'91*, volume 520 of *Lecture Notes in Computer Science*, pages 151–160. Springer-Verlag, 1991.

[23] G. Dowek. A complete proof synthesis method for the cube of type systems. *Journal of Logic and Computation*, 3(3):287–315, 1993.

[24] G. Dowek, G. Huet, and B. Werner. On the existence of long $\beta\eta$-normal forms in the cube. In H. Geuvers, editor, *Informal Proceedings of TYPES'93*, pages 115–130, 1993. Available from http://www.dcs.ed.ac.uk/lfcsinfo/research/types-bra/proc/index.html.

[25] P. de Groote. Denotations for classical proofs: Preliminary results. In A. Nerode and M. Taitslin, editors, *Symposium on Logical Foundations of Computer Science*, volume 620 of *Lecture Notes in Computer Science*, pages 105–116. Springer-Verlag, 1992.

[26] P. de Groote. The conservation theorem revisited. In Bezem and Groote [12], pages 163–178.

[27] P. de Groote. A CPS-translation of the $\lambda\mu$-calculus. In S. Tison, editor, *Colloquium on Trees in Algebra and Programming*, volume 787 of *Lecture Notes in Computer Science*, pages 85–99. Springer-Verlag, 1994.

[28] P. de Groote. On the relation between the $\lambda\mu$-calculus and the syntactic theory of sequential control. In *International Conference on Logic and Automated Reasoning*, volume 822 of *Lecture Notes in Computer Science*, pages 31–43. Springer-Verlag, 1994.

[29] P. de Groote. Strong normalization in a non-deterministically typed lambda-calculus. In A. Nerode and Y.V. Matiyasevich, editors, *Symposium on Logical Foundations of Computer Science*, volume 813 of *Lecture Notes in Computer Science*, pages 142–152. Springer-Verlag, 1994.

[30] P. de Groote. A simple calculus of exception handling. In Dezani-Ciancaglini and Plotkin [31], pages 201–215.

[31] M. Dezani-Ciancaglini and G. Plotkin, editors. *Typed Lambda Calculus and Applications*, volume 902 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995.

[32] B.F. Duba, R. Harper, and D. MacQueen. Typing first-class continuations in ML. In *Conference Record of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1991.

[33] M. Felleisen. *The Calculi of λ_v-CS Conversion: A Syntactic theory of Control and State in Imperative Higher Order programming Languages.* PhD thesis, Indiana University, 1987.

[34] M. Felleisen and D. Friedman. Control operators, the SECD machine, and the l-calculus. In M. Wirsing, editor, *Formal Description of Programming Concepts III*, pages 193–217. North-Holland, 1986.

[35] M. Felleisen, D. Friedman, E. Kohlbecker, and B. Duba. A syntactic theory of sequential control. *Theoretical Computer Science*, 52(3):205–237, 1987.

[36] C. Flanagan, A. Sabry, B. Duba, and M. Felleisen. The essence of compiling with continuations. In *Programming Language Design and Implementation*, 1993.

[37] D. Friedman, M. Wand, and C. Haynes. *Essentials of Programming Languages.* MIT Press and McGraw-Hill, 1991.

[38] J.H. Gallier. On Girard's "candidats de reductibilité". In P. Oddifreddi, editor, *Logic and Computer Science*, pages 123–203. Academic Press Limited, 1990.

[39] R.O. Gandy. An early proof of normalization by A.M. Turing. In Seldin and Hindley [80], pages 453–455.

[40] J.H. Geuvers. *Logics and Type Systems.* PhD thesis, University of Nijmegen, 1993.

[41] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupres dans l'arithmétique d'ordre supérieur.* PhD thesis, Université Paris VII, 1972.

[42] J.-Y. Girard. A new constructive logic: Classical logic. *Mathematical Structures in Computer Science*, 1(3):255–296, 1991.

[43] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1989.

[44] T.G. Griffin. A formulae-as-types notion of control. In *Conference Record of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 47–58. ACM Press, 1990.

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

[46] R. Harper and M. Lillibridge. Explicit polymorphism and CPS conversion. In *Conference Record of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 206–219. ACM Press, 1993.

[47] R. Harper and M. Lillibridge. Polymormphic type assignment and CPS conversion. *LISP and Symbolic Computation*, 6:361–380, 1993.

[48] J. Hatcliff and O. Danvy. A generic account of continuation-passing styles. In Hans Boehm, editor, *Proceedings of the Twenty-first Annual ACM Symposium on Principles of Programming Languages*, Portland, Oregon, January 1994. ACM Press.

[49] J. Hatcliff and O. Danvy. Thunks and the λ-calculus. *Journal of Functional Programming*, 1995. (in press). The extended version of this paper appears as DIKU-Report 95/3.

[50] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-calculus.* Cambridge University Press, 1986.

[51] W. Howard. The formulae-as-types notion of construction. In Seldin and Hindley [80], pages 479–490.

[52] M. Karr. "Delayability" in proofs of strong normalizability in the typed lambda calculus. In H. Ehrig, C. Floyd, M. Nivat, and J. Thatcher, editors, *Mathematical Foundations of Computer Software*, volume 185 of *Lecture Notes in Computer Science*, pages 208–222. Springer-Verlag, 1985.

[53] A.J. Kfoury and J. Wells. New notions of reduction and non-semantic proofs of β-strong normalization in typed λ-calculi. Technical Report 94-014, Boston University Computer Science Department, 1994.

[54] Z. Khasidashvili. *Form Reduction Systems and Reductions of Contracted Forms and Lambda-Terms.* PhD thesis, Tbilisi State University, 1988. In Russian.

[55] J.W. Klop. *Combinatory Reduction Systems.* PhD thesis, Utrecht University, 1980. CWI Tract, Amsterdam.

[56] J.-L. Krivine. *Lambda-Calculus, Types and Models.* Ellies Horwood Series in Computers and their Applications. Masson and Ellis Horwood, English Edition, 1993.

[57] J.-L. Krivine. Classical logic, storage operators, and second-order λ-calculus. *Annals of Pure and Applied Logic*, 68:53–78, 1994.

[58] A.R. Meyer and M. Wand. Continuation semantics in typed lambda-calculi (summary). In R. Parikh, editor, *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 219–224. Springer-Verlag, 1985.

[59] C. Murthy. *Extracting Constructive Contents from Classical Proofs.* PhD thesis, Cornell University, 1990.

[60] C. Murthy. An evaluation semantics for classical proofs. In *Logic in Computer Science*, 1991.

[61] C.R. Murthy. A computational analysis of Girard's translation and LC. In *Logic in Computer Science*, 1992.

[62] C.R. Murthy. Control operators, hierachies, and pseudo-classical type systems: A-translation at work. In *ACM SIGPLAN Workshop on Continuations*, 1992.

[63] R. Nederpelt. *Strong normalization for a typed lambda calculus with lambda structured types.* PhD thesis, Eindhoven, 1973.

[64] K. Nielsen and M.H. Sørensen. Call-by-name CPS-translation as a binding-time improvement. In A. Mycroft, editor, *Static Analysis Symposium*, volume 983 of *Lecture Notes in Computer Science*, pages 296–313. Springer-Verlag, 1995.

[65] C.-H. L. Ong. A semantic view of classical proofs: Type-theoretic, categorical, and denotational characterizations. In *Logic in Computer Science*, 1996.

[66] M. Parigot. Free deduction: An analysis of "computations" in classical logic. In *Second Russian Conference on Logic programming*, volume 592 of *Lecture Notes in Artificial Intelligence*, pages 361–380. Springer-Verlag, 1991.

[67] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *International Conference on Logic Programming and Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*, pages 190–201. Springer-Verlag, 1992.

[68] M. Parigot. Classical proofs as programs. In *Kurt Gödel Colloquium*, volume 713 of *Lecture Notes in Computer Science*, pages 263–276. Springer-Verlag, 1993.

[69] M. Parigot. Strong normalization for second order classical natural deduction. In *Logic in Computer Science*, 1993.

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

[71] D. Prawitz. *Natural Deduction: A proof theoretical study.* Almquist & Wiksell, 1965.

[72] D. Prawitz. Ideas and results of proof theory. In J.E. Fenstad, editor, *The 2nd Scandinavian Logical Symposium*, pages 235–307. North-Holland, 1970.

[73] N.J. Rehof and M.H. Sørensen. The λ_Δ calculus. In M. Hagiya and J. Mitchell, editors, *Theoretical Aspects of Computer Software*, volume 789 of *Lecture Notes in Computer Science*, pages 516–542. Springer-Verlag, 1994.

[74] A. Rezus. Classical proofs: Lambda calculus methods in elementary proof theory, 1991. Manuscript.

[75] A. Rezus. Beyond BHK, 1993. In Barendregt, Bezem, and Klop, editors, *Dirk van Dalen Festschrift*, pages 114–120. University of Utrecht, 1993.

[76] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp and Symbolic Computation*, 6:289–360, 1993.

[77] A. Sabry and M. Felleisen. Is continuation passing useful for data-flow analysis? In *Programming Language Design and Implementation*, 1994.

[78] M. Sato. Intuitionistic and classical natural deduction systems with the catch and throw rules. Manuscript, 1995.

[79] J.P Seldin. Normalization and excluded middle. *Studia Logica*, XLVIII(2):193–217, 1989.

[80] J.P. Seldin and J.R. Hindley, editors. *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism.* Academic Press Limited, 1980.

[81] P. Severi and E. Poll. PTS with definitions. In A. Nerode and Y.N. Matiyasevich, editors, *Proceedings of LFCS'94*, volume 813 of *Lecture Notes in Computer Science*, pages 316–328. Springer-Verlag, 1994.

[82] O. Shivers. *Control-Flow Analysis of Higher-Order Languages.* PhD thesis, Carnegie Mellon University, 1991.

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

[84] G. Stålmarck. Normalization theorems for full first order classical natural deduction. *Journal of Symbolic Logic*, 56(1):129–149, 1991.

[85] M.H. Sørensen. Strong normalization from weak normalization in typed λ-calculi. Submitted, 1996.

[86] W.W. Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):190–212, 1967.

[87] W.W. Tait. A realizability interpretation of the theory of species. In R. Parikh, editor, *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer-Verlag, 1975.

[88] A.S. Troelstra. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lecture Notes in Mathematics*. Springer-Verlag, 1973.

[89] J.L. Underwood. *Aspects of the Computational Content of Proofs.* PhD thesis, Cornell University, 1994.

[90] B. Werner. Continuations, evaluation styles and types systems. Manuscript, 1992.

[91] H. Xi. On weak and strong normalisations. Manuscript announced on the types mailing list, February, 1996.