CPS Translating Inductive and Coinductive Types

[Extended Abstract]

Gilles Barthe
INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93
F-06902 Sophia-Antipolis Cedex, France
Gilles.Barthe@inria.fr

Tarmo Uustalu*
Dep. de Informática, Universidade do Minho
Campus de Gualtar
P-4710-057 Braga, Portugal
tarmo@di.uminho.pt

ABSTRACT

We investigate CPS translatability of typed λ-calculi with inductive and coinductive types. We show that tenable Plotkin-style call-by-name CPS translations exist for simply typed λ-calculi with a natural number type and stream types and, more generally, with arbitrary positive inductive and coinductive types. These translations also work in the presence of control operators and generalize for dependently typed calculi where case-like eliminations are only allowed in non-dependent forms. No translation is possible along the same lines for small Σ-types and sum types with dependent case.

Categories and Subject Descriptors

F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*Functional constructs, Control primitives, Type structure*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Lambda calculus and related systems, Proof theory*

General Terms

theory

Keywords

inductive and coinductive types, CPS translations, typed λ-calculi, classical logic and control, dependent types

1. INTRODUCTION

Background

Continuation-passing style (CPS) is a style of programming well suited for program analyses and optimizations, so CPS languages are commonly used as intermediate languages in compilers, see, e.g., [3, 18]. *CPS translations* are transformations converting programs into CPS terms. In a seminal paper [41], Plotkin defined call-by-value and call-by-name CPS translations for the untyped λ-calculus and established some of their important properties. Felleisen et al. [17] extended the call-by value translation to also cover their control operator $\mathcal{C}$. Meyer and Wand [32] showed that Plotkin's call-by-value CPS translation is type-correct in a simply typed setting. Griffin [23] noted that Felleisen's $\mathcal{C}$ types with the double negation rule of classical logic and showed that both Felleisen's CPS translation and its call-by-name version correspond to well-known embeddings of classical logic into intuitionistic logic. Subsequently, typed CPS translations and correctness results have been given for more powerful typed λ-calculi, see, e.g., [27, 28, 8], and applied to the compilation and optimization of typed languages, see, e.g., [19, 44]. Griffin's discovery initiated a series of studies on the computational content of classical proofs where CPS translations are a frequently employed tool, see, e.g., [13, 33, 38, 39, 34, 12, 4, 42, 24, 25, 35, 36, 6].

Inductive and coinductive types, see, e.g., [31, 29, 20, 15, 40], are syntactic representations for initial algebras (such as natural numbers and lists) resp. final coalgebras (such as conatural numbers and streams) in typed λ-calculi. Despite being pervasive in the type-theoretical literature on functional languages and proof assistants, we are not aware of any study of CPS translations for (co)inductive types.

Contribution

The purpose of this paper is to present CPS translations for (co)inductive types. Our contribution is three-fold.

First, we extend the typed version of Plotkin's call-by-name CPS translation to (co)inductive types. More precisely, we define a type-preserving and reduction-preserving CPS translation for $\lambda_{\mu,\nu}$, a simply-typed λ-calculus with sum and product types and positive inductive and coinductive types. A salient feature of our translation is that it also applies under μ and ν. This is required for the translation to enjoy a substitution property for types and hence for scaling it up for more powerful type disciplines such as polymorphism or higher-order polymorphism. One of the effects is that strictly positive (co)inductive types such as $\mathsf{Nat} = \mu Z.\ 1 + Z$ get transformed into (double negations of) non-strictly positive ones such as $\mathsf{Nat}^{\mathsf{c}} = \mu Z.\ 1 + \neg\neg Z$, cf. [11].

Second, we consider the CPS translation of $\lambda\Delta_{\mu,\nu}$, which

*On leave from Inst. of Cybernetics, Tallinn Technical University, Akadeemia tee 21, EE-12618 Tallinn, Estonia, tarmo@cs.ioc.ee.

PEPM '02 Jan. 14–15, 2002 Portland, OR, USA

extends $\lambda_{\mu,\nu}$ with a control operator Δ described in [42]. Since the CPS translation provides an embedding from languages with control to languages without control, the target language remains $\lambda_{\mu,\nu}$. This allows us to use the translation to prove that every number-theoretic function representable in $\lambda\Delta_{\mu,\nu}$ is also representable in $\lambda_{\mu,\nu}$, indicating one possible line of application for CPS translations for (co)inductive types.

Third, we study extensions of the suggested translation for more powerful type disciplines. Building on [8], one can show that the translation scales up to systems with dependent types. However, we prove that the translation is extendable neither to small Σ-types nor to sum types with dependent case. For the latter, we analyse Geuvers' [21] proof of inconsistency of classical logic in the Calculus of Inductive Constructions to conclude that the translation developed in this paper, which is the natural generalization of Plotkin's original CPS translation, cannot be extended to deal with dependent case analysis.

Besides, our results shed some light on the simulability of inductive types by coinductive types and vice versa, cf. [2], and the computational content of commuting reductions, cf. [26].

Organization of the paper

The paper is organized as follows. In Section 2, we review CPS translation of the standard extension of simply typed λ-calculus with sums and products, empty and unit types. In Section 3, we generalize this for a source language with natural numbers and streams. In Section 4, we extend these translations further to a system with general positive (co)inductive types. In all cases, we consider a standard CPS translation and an alternative "de Morgan" version in which the CPS image of a sum type (resp. an inductive type) is defined via a product type (resp. a coinductive type). Further we show the translations to be type-preserving and reduction-preserving.

In Section 5, we briefly discuss extensions of the systems introduced with commuting conversions. In Section 6, we extend the source language with a control operator, and prove that the extended CPS is correct wrt. typing and convertibility, and conclude that every number-theoretic function representable in the source language is also representable in the target language (without control).

In Section 7, we discuss generalizations of the translation to the Calculus of Inductive Constructions (CIC). We first present the translation of the Calculus of Constructions (CC) and then discuss specific type constructions of CIC.

In Section 8, we list some conclusions and directions for future research.

2. SIMPLY TYPED LAMBDA-CALCULUS WITH SUM AND PRODUCT TYPES

We start by introducing $\lambda_{+,\times}$, a simply typed λ-calculus with sum and product types, an empty type and a unit type (in other words, a term calculus for full intuitionistic propositional logic) and looking at its CPS translation. We use it as a base for extensions we are interested in and as an instructive example, since sum and product types are prototypical for inductive and coinductive types.

The language of the system $\lambda_{+,\times}$ (types, objects) is given by the grammar

$$\begin{array}{rcl} A,B,C & ::= & X \mid A \to B \mid A_1 + A_2 \mid 0 \mid A_1 \times A_2 \mid 1 \\ M,N,P & ::= & x \mid \lambda x.\ M \mid N\ P \mid \mathsf{inl}(M) \mid \mathsf{inr}(M) \\ & & \mid \mathsf{case}(N, u.\ P_1, u.\ P_2) \mid \nabla(N) \\ & & \mid \langle M_1, M_2 \rangle \mid \langle\rangle \mid \mathsf{fst}(N) \mid \mathsf{snd}(N) \end{array}$$

The typing and reduction rules of $\lambda_{+,\times}$ appear in Fig. 1.

The most straightforward generalization of Plotkin's [41] call-by-name CPS translation of λ gives a translation of $\lambda_{+,\times}$ to itself. This is presented in Fig. 2 in a semi-optimized, "near-colon" version. There and elsewhere below, $\perp$ is a distinguished type variable (for the answer type), $\neg A$ is used as shorthand for $A \to \perp$. Numerous other generalizations are possible. To demonstrate this, we look also at a variant where $+$, 0 are treated differently, derived from the intuitionistic de Morgan laws $\neg(A_1 + A_2) \cong \neg A_1 \times \neg A_2$, $\neg 0 \cong 1$. This different translation, with $\lambda_\times$ as the target calculus, is given in Fig. 3.

Both translations are correct in the sense of preserving typing and reductions. The proof hinges on a substitution lemma.

Lemma 1 (Substitution lemma).

1. (a) $\textsc{cont}[\![A]\!][\textsc{cont}[\![C]\!]/\neg Z] = \textsc{cont}[\![A[C/Z]]\!]$,
   (b) $\textsc{cpst}[\![A]\!][\textsc{cont}[\![C]\!]/\neg Z] = \textsc{cpst}[\![A[C/Z]]\!]$.
2. (a) $([\![M]\!]{:}K)[\textsc{cps}[\![P]\!]/u] \twoheadrightarrow_\beta [\![M[P/u]]\!]{:}(K[\textsc{cps}[\![P]\!]/u])$,
   (b) $\textsc{cps}[\![M]\!][\textsc{cps}[\![P]\!]/u] \twoheadrightarrow_\beta \textsc{cps}[\![M[P/u]]\!]$.

Proposition 1 (Correctness).

1. *If* $\Gamma \vdash M : A$, *then*
   (a) $\textsc{cpst}[\![\Gamma]\!] \vdash K : \textsc{cont}[\![A]\!]$ *implies* $\Gamma \vdash [\![M]\!]{:}K : \perp$,
   (b) $\textsc{cpst}[\![\Gamma]\!] \vdash \textsc{cps}[\![M]\!] : \textsc{cpst}[\![A]\!]$.
2. *If* $M \twoheadrightarrow_\beta M'$, *then*
   (a) $[\![M]\!]{:}K \twoheadrightarrow_\beta [\![M']\!]{:}K$,
   (b) $\textsc{cps}[\![M]\!] \twoheadrightarrow_\beta \textsc{cps}[\![M']\!]$.

3. NATURAL NUMBERS AND STREAMS

We continue by CPS translating the simplest examples of inductive and coinductive types: the natural number type $\mathsf{Nat} = \mu Z.\ 1 + Z$ and stream types $\mathsf{Str}(A) = \nu Z.\ A \times Z$. To keep the presentation simple, we equip the natural number types with iteration and case and stream types with coiteration and projections (as opposed to, say, primitive recursion resp. primitive corecursion). The language of the system $\lambda_{\mathsf{Nat},\mathsf{Str}}$ is given by the grammar

$$\begin{array}{rcl} A,B,C & ::= & \ldots \mid \mathsf{Nat} \mid \mathsf{Str}(A) \\ M,N,P & ::= & \ldots \mid \mathsf{o} \mid \mathsf{s}(M) \mid \mathsf{niter}(N, P_o, u.\ P_s) \\ & & \mid \mathsf{ncase}(N, P_o, u.\ P_s) \mid \mathsf{scoit}(M, y.\ Q_h, y, Q_t) \\ & & \mid \mathsf{cons}(M_h, M_t) \mid \mathsf{hd}(N) \mid \mathsf{tl}(N) \end{array}$$

The typing and reduction rules are given in Fig. 4.

Translating Nat and $\mathsf{Str}(A)$, we keep in mind the isomorphisms $\mathsf{Nat} \cong 1 + \mathsf{Nat}$, $\mathsf{Str}(A) \cong A \times \mathsf{Str}(A)$, proceed from the translations of $+$, $\times$ from Sec. 2, and strive for preserving the isomorphisms. Such translations are possible in terms of the types $\mathsf{Nat}^{\mathsf{c}} = \mu Z.\ 1 + \neg\neg Z$ (modulo rejecting $\neg\neg 1$ in favor of 1) and $\mathsf{Str}^{\mathsf{c}}(A) = \nu Z.\ A \times \neg\neg Z$, doubly negated modifications of Nat and $\mathsf{Str}(A)$. Note that while Nat, $\mathsf{Str}(A)$, just as

Typing rules:

$$\frac{}{\Gamma \vdash x : C} \text{ if } (x : C) \text{ in } \Gamma \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\ M : A \to B} \qquad \frac{\Gamma \vdash N : A \to B \quad \Gamma \vdash P : A}{\Gamma \vdash N\ P : B}$$

$$\frac{\Gamma \vdash M : A_1}{\Gamma \vdash \mathsf{inl}(M) : A_1 + A_2} \qquad \frac{\Gamma \vdash M : A_2}{\Gamma \vdash \mathsf{inr}(M) : A_1 + A_2} \qquad \frac{\Gamma \vdash N : A_1 + A_2 \quad \Gamma, u : A_1 \vdash P_1 : C \quad \Gamma, u : A_2 \vdash P_2 : C}{\Gamma \vdash \mathsf{case}(N, u.\ P_1, u.\ P_2) : C} \qquad \frac{\Gamma \vdash N : 0}{\Gamma \vdash \nabla(N) : C}$$

$$\frac{\Gamma \vdash M_1 : A_1 \quad \Gamma \vdash M_2 : A_2}{\Gamma \vdash \langle M_1, M_2 \rangle : A_1 \times A_2} \qquad \frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{\Gamma \vdash N : A_1 \times A_2}{\Gamma \vdash \mathsf{fst}(N) : A_1} \qquad \frac{\Gamma \vdash N : A_1 \times A_2}{\Gamma \vdash \mathsf{snd}(N) : A_2}$$

β-reduction rules:

$$\begin{array}{rcl}
(\lambda x.\ M)\ P & \to & M[P/x] \\
\mathsf{case}(\mathsf{inl}(M), u.\ P_1, u.\ P_2) & \to & P_1[M/u] \\
\mathsf{case}(\mathsf{inr}(M), u.\ P_1, u.\ P_2) & \to & P_2[M/u] \\
\mathsf{fst}(\langle M_1, M_2 \rangle) & \to & M_1 \\
\mathsf{snd}(\langle M_1, M_2 \rangle) & \to & M_2
\end{array}$$

Figure 1: Typing and reduction rules of $\lambda_{+,\times}$

Translation of types:

$$\begin{array}{rcl}
\textsc{cpst}[\![A]\!] & = & \neg\textsc{cont}[\![A]\!] \\
\textsc{cont}[\![X]\!] & = & \neg X \\
\textsc{cont}[\![A \to B]\!] & = & \neg(\textsc{cpst}[\![A]\!] \to \textsc{cpst}[\![B]\!]) \\
\textsc{cont}[\![A_1 + A_2]\!] & = & \neg(\textsc{cpst}[\![A_1]\!] + \textsc{cpst}[\![A_2]\!]) \\
\textsc{cont}[\![0]\!] & = & \neg 0 \\
\textsc{cont}[\![A_1 \times A_2]\!] & = & \neg(\textsc{cpst}[\![A_1]\!] \times \textsc{cpst}[\![A_2]\!]) \\
\textsc{cont}[\![1]\!] & = & \neg 1
\end{array}$$

Translation of objects:

$$\begin{array}{rcl}
\textsc{cps}[\![M]\!] & = & \lambda k.\ [\![M]\!]{:}k \\
[\![x]\!]{:}K & = & x\ K \\
[\![\lambda x.\ M]\!]{:}K & = & K\ (\lambda x.\ \textsc{cps}[\![M]\!]) \\
[\![N\ P]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ n\ \textsc{cps}[\![P]\!]\ K) \\
[\![\mathsf{inl}(M)]\!]{:}K & = & K\ \mathsf{inl}(\textsc{cps}[\![M]\!]) \\
[\![\mathsf{inr}(M)]\!]{:}K & = & K\ \mathsf{inr}(\textsc{cps}[\![M]\!]) \\
[\![\mathsf{case}(N, u.\ P_1, u.\ P_2)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{case}(n, u.\ [\![P_1]\!]{:}K, u.\ [\![P_2]\!]{:}K)) \\
[\![\nabla(N)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \nabla(n)) \\
[\![\langle M_1, M_2 \rangle]\!]{:}K & = & K\ \langle \textsc{cps}[\![M_1]\!], \textsc{cps}[\![M_2]\!] \rangle \\
[\![\mathsf{fst}(N)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{fst}(n)\ K) \\
[\![\mathsf{snd}(N)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{snd}(n)\ K) \\
[\![\langle\rangle]\!]{:}K & = & K\ \langle\rangle
\end{array}$$

Translation of contexts:

$$\begin{array}{rcl}
\textsc{cpst}[\![\diamond]\!] & = & \diamond \\
\textsc{cpst}[\![\Gamma, x : C]\!] & = & \textsc{cpst}[\![\Gamma]\!], x : \textsc{cpst}[\![C]\!]
\end{array}$$

Figure 2: CPS translation of $\lambda_{+,\times}$

Translation of types:

$$\begin{array}{rcl} \textsc{cont}[\![A_1 + A_2]\!] & = & \neg\textsc{cpst}[\![A_1]\!] \times \neg\textsc{cpst}[\![A_2]\!] \\ \textsc{cont}[\![0]\!] & = & 1 \end{array}$$

Translation of objects:

$$\begin{array}{rcl} [\![\mathsf{inl}(M)]\!]{:}K & = & \mathsf{fst}(K)\ \textsc{cps}[\![M]\!] \\ [\![\mathsf{inr}(M)]\!]{:}K & = & \mathsf{snd}(K)\ \textsc{cps}[\![M]\!] \\ [\![\mathsf{case}(N, u.\ P_1, u.\ P_2)]\!]{:}K & = & [\![N]\!]{:}\langle \lambda u.\ [\![P_1]\!]{:}K, \lambda u.\ [\![P_2]\!]{:}K\rangle \\ [\![\nabla(N)]\!]{:}K & = & [\![N]\!]{:}\langle\rangle \end{array}$$

Figure 3: Alternative CPS translation of $\lambda_{+,\times}$

Typing rules:

$$\frac{}{\Gamma \vdash \mathsf{o} : \mathsf{Nat}} \qquad \frac{\Gamma \vdash M : \mathsf{Nat}}{\Gamma \vdash \mathsf{s}(M) : \mathsf{Nat}} \qquad \frac{\Gamma \vdash N : \mathsf{Nat} \quad \Gamma \vdash P_o : C \quad \Gamma, u : C \vdash P_s : C}{\Gamma \vdash \mathsf{niter}(N, P_o, u.\ P_s) : C}$$

$$\frac{\Gamma \vdash N : \mathsf{Nat} \quad \Gamma \vdash P_o : C \quad \Gamma, u : \mathsf{Nat} \vdash P_s : C}{\Gamma \vdash \mathsf{ncase}(N, P_o, u.\ P_s) : C} \qquad \frac{\Gamma \vdash M : D \quad \Gamma, y : D \vdash Q_h : A \quad \Gamma, y : D \vdash Q_t : D}{\Gamma \vdash \mathsf{scoit}(M, y.\ Q_h, y.\ Q_t) : \mathsf{Str}(A)}$$

$$\frac{\Gamma \vdash M_h : A \quad \Gamma \vdash M_t : \mathsf{Str}(A)}{\Gamma \vdash \mathsf{cons}(M_h, M_t) : \mathsf{Str}(A)} \qquad \frac{\Gamma \vdash N : \mathsf{Str}(A)}{\Gamma \vdash \mathsf{hd}(N) : A} \qquad \frac{\Gamma \vdash N : \mathsf{Str}(A)}{\Gamma \vdash \mathsf{tl}(N) : \mathsf{Str}(A)}$$

β−reduction rules:

$$\begin{array}{rcl} \mathsf{niter}(\mathsf{o}, P_o, u.\ P_s) & \to & P_o \\ \mathsf{niter}(\mathsf{s}(M), P_o, u.\ P_s) & \to & P_s[\mathsf{niter}(M, P_o, u.\ P_s)/u] \\ \mathsf{ncase}(\mathsf{o}, P_o, u.\ P_s) & \to & P_o \\ \mathsf{ncase}(\mathsf{s}(M), P_o, u.\ P_s) & \to & P_s[M/u] \\ \mathsf{hd}(\mathsf{scoit}(M, y.\ Q_h, y.\ Q_t)) & \to & Q_h[M/y] \\ \mathsf{tl}(\mathsf{scoit}(M, y.\ Q_h, y.\ Q_t)) & \to & \mathsf{scoit}(Q_t[M/y], y.\ Q_h, y.\ Q_t) \\ \mathsf{hd}(\mathsf{cons}(M_h, M_t)) & \to & M_h \\ \mathsf{tl}(\mathsf{cons}(M_h, M_t)) & \to & M_t \end{array}$$

Figure 4: Typing and reduction rules of $\lambda_{\mathsf{Nat},\mathsf{Str}}$

any "normal" (co)inductive types are strictly positive, $\mathsf{Nat}^{\mathsf{c}}$ and $\mathsf{Str}^{\mathsf{c}}(A)$ are non-strictly positive. The language of the system $\lambda_{\mathsf{Nat}^{\mathsf{c}},\mathsf{Str}^{\mathsf{c}}}$ is given by the grammar

$$\begin{array}{rcl} A,B,C & = & \ldots \mid \mathsf{Nat}^{\mathsf{c}} \mid \mathsf{Str}^{\mathsf{c}}(A) \\ M,N,P & = & \ldots \mid \mathsf{o}^{\mathsf{c}} \mid \mathsf{s}^{\mathsf{c}}(M) \mid \mathsf{niter}^{\mathsf{c}}(N, P_o, u.\ P_s) \\ & & \mid \mathsf{ncase}^{\mathsf{c}}(N, P_o, u.\ P_s) \mid \mathsf{scoit}^{\mathsf{c}}(M, y.\ Q_h, y, Q_t) \\ & & \mid \mathsf{cons}^{\mathsf{c}}(M_h, M_t) \mid \mathsf{hd}^{\mathsf{c}}(N) \mid \mathsf{tl}^{\mathsf{c}}(N) \end{array}$$

The typing and reduction rules for this system appear in Fig. 5.

The two translations of $\lambda_{+,\times}$ from Sec. 2 lead to two translations for $\lambda_{\mathsf{Nat},\mathsf{Str}}$. The more straightforward translation is given in Fig. 6. The other is a derivate from the intuitionistic isomorphism $\neg\mathsf{Nat} \cong \mathsf{Str}(\bot)$, an instance of a generalization of the intuitionistic de Morgan law for inductive types. This translation, presented in Fig. 7, has $\lambda_{\mathsf{Str}^{\mathsf{c}}}$ as the target system. Observe that the second translation of Nat is smoother. (In the first one, the clumsiness in the rendering of niter is due to a misbalance in the number of negations; the bureaucracy marked by an underline replaces three negations by one. In the second translation, no such misbalance arises.)

Again, both translations are correct.

PROPOSITION 2.

1. *If* $\Gamma \vdash M : A$, *then* $\textsc{cpst}[\![\Gamma]\!] \vdash \textsc{cps}[\![M]\!] : \textsc{cpst}[\![A]\!]$.
2. *If* $M \twoheadrightarrow_\beta M'$, *then* $\textsc{cps}[\![M]\!] \twoheadrightarrow_\beta \textsc{cps}[\![M']\!]$.

Note that our second translation provides a simulation of an inductive type by a coinductive type, which, however, only works, if the whole language is translated. This can be contrasted with the recent work [2] by Altenkirch on representing function types with inductive domains as coinductive types.

4. GENERAL POSITIVE INDUCTIVE AND COINDUCTIVE TYPES

The underlying idea behind the translation of $\mathsf{Nat}(A)$ and $\mathsf{Str}(A)$ extends to arbitrary positive inductive and coinductive types (with iterators and destructors resp. coiterators and constructors as the elimination resp. introduction operators). We now sketch this extension. Our source system of interest is $\lambda_{\mu,\nu}$, an extension of $\lambda_{+,\times}$. The language is given by the grammar

$$\begin{array}{rcl} A,B,C & ::= & \ldots \mid \mu Z.\ A \mid \nu Z.\ A \\ M,N,P & ::= & \ldots \mid \mathsf{in}_{Z.\,A}(M) \mid \mathsf{iter}(N, u.\ P) \mid \mathsf{in}^{-1}(N) \\ & & \mid \mathsf{coit}(M, y.\ Q) \mid \mathsf{out}^{-1}(M) \mid \mathsf{out}_{Z.\,A}(N) \end{array}$$

The raw types $\mu Z.\ A$, $\nu Z.\ A$ are legal if A is positive wrt. Z, in which case an operator $Map_{Z.\,A}$ is defined such that

$$\frac{\Gamma \vdash M : A[D/Z] \quad \Gamma, x : D \vdash P : C}{\Gamma \vdash Map_{Z.\,A}(M, x.\ P) : A[C/Z]}$$

The typing and reduction rules of $\lambda_{\mu,\nu}$ appear in Fig. 8.

Generalizing the translations of Sec. 3, we arrive at two translations of $\lambda_{\mu,\nu}$, the first into $\lambda_{\mu,\nu}$, the second into λ_ν; the second translation draws on the isomorphism $\neg\mu Z.\ A[\neg Z/Z] \cong \nu Z.\ \neg A$. The translations are presented in Figs. 9, 10. Note that here the second translation of μ is even more smooth than the first than in the previous section, since the first involves a map operator in the translation of iter while the second does not.

Both translations are type-preserving; the preservation of convertibility depends on the exact definitions of the map operators.

Inconveniences with map operators are an inherent trait of calculi with positive (co)inductive types. These are overcome in calculi with (co)recursors à la Mendler [31, 30, 45] where the positivity condition can be dropped and the map operators are not needed in order to formulate the reduction rules. CPS translations of calculi with Mendler-style (co)inductive types will be discussed in the full version of the paper.

5. COMMUTING REDUCTIONS

Optionally, the reduction calculus of $\lambda_{+,\times}$ may be completed with the so-called commuting reductions for $+$, 0. These are

$$\begin{array}{rcl} \mathsf{elim}(\mathsf{case}(N, u.\ P_1, u.\ P_2)) & \to & \mathsf{case}(N, u.\ \mathsf{elim}(P_1), u.\ \mathsf{elim}(P_2)) \\ \mathsf{elim}(\nabla(N)) & \to & \nabla(N) \end{array}$$

Here, $\mathsf{elim}(N)$ is a general notation for any elimination object with N as the main argument. In the case of $\lambda_{+,\times}$, this covers object forms $N\ P$, $\mathsf{case}(N, u.\ P_1, u.\ P_2)$, $\nabla(N)$, $\mathsf{fst}(N)$, $\mathsf{snd}(N)$. The two CPS translations are correct also wrt. βc-reductions; actually, they collaps c-reductions into identities. This fact may be interpreted as evidence in support of the statement that commuting reductions carry no computational content. In [26], the same effect was achieved with a more complex CPS translation of $\lambda_{+,\times}$ to λ.

To $\lambda_{\mathsf{Nat},\mathsf{Str}}$, one can add a commuting reduction for ncase analogous that for case –

$$\begin{array}{l} \mathsf{elim}(\mathsf{ncase}(N, P_o, u.\ P_s)) \\ \quad \to \mathsf{ncase}(N, \mathsf{elim}(P_o), u.\ \mathsf{elim}(P_s)) \end{array}$$

– and the correctness result extends. The controversial issue of commuting reductions for niter and scoit will be analysed in the full version of the paper.

6. CLASSICAL LOGIC

We now consider a combination of inductive types with control. For the sake of simplicity, we only consider natural numbers. We extend λ_{Nat} with a local control operator Δ introduced in [42].

The language $\lambda\Delta_{\mathsf{Nat}}$ is obtained from λ_{Nat} by adding an object form $\Delta x.\ N$ with the typing rule

$$\frac{\Gamma, x : A \to 0 \vdash N : 0}{\Gamma \vdash \Delta x.\ N : A}$$

a commuting reduction rule

$$\mathsf{elim}(\Delta x.\ N) \quad \to \quad \Delta y.\ N[\lambda z.\ y\ \mathsf{elim}(z)/x]$$

and two further reduction rules of a specific nature

$$\begin{array}{rcll} \Delta x.\ x\ N & \to & N & \text{if } x \notin \mathrm{FV}(N) \\ \Delta x.\ x\ \Delta y.\ N & \to & \Delta x.\ N[x/y] & \end{array}$$

We speak of both types of reductions as Δ-reductions. The rules presented strengthen those for ∇ from Sec. 2, so ∇ is definable through Δ: $\nabla(N) = \Delta x.\ N$. From a logical viewpoint, the typing rule of Δ is the double negation rule, which is the classical elimination rule for falsum, whereas that of

Typing rules:

$$\frac{}{\Gamma \vdash \mathsf{o}^{\mathsf{c}} : \mathsf{Nat}^{\mathsf{c}}} \quad \frac{\Gamma \vdash M : \neg\neg\mathsf{Nat}^{\mathsf{c}}}{\Gamma \vdash \mathsf{s}^{\mathsf{c}}(M) : \mathsf{Nat}^{\mathsf{c}}} \quad \frac{\Gamma \vdash N : \mathsf{Nat}^{\mathsf{c}} \quad \Gamma \vdash P_o : C \quad \Gamma, u : \neg\neg C \vdash P_s : C}{\Gamma \vdash \mathsf{niter}^{\mathsf{c}}(N, P_o, u.\ P_s) : C}$$

$$\frac{\Gamma \vdash N : \mathsf{Nat}^{\mathsf{c}} \quad \Gamma \vdash P_o : C \quad \Gamma, u : \neg\neg\mathsf{Nat}^{\mathsf{c}} \vdash P_s : C}{\Gamma \vdash \mathsf{ncase}^{\mathsf{c}}(N, P_o, u.\ P_s) : C} \quad \frac{\Gamma \vdash M : D \quad \Gamma, y : D \vdash Q_h : A \quad \Gamma, y : D \vdash Q_t : \neg\neg D}{\Gamma \vdash \mathsf{scoit}^{\mathsf{c}}(M, y.\ Q_h, y.\ Q_t) : \mathsf{Str}^{\mathsf{c}}(A)}$$

$$\frac{\Gamma \vdash M_h : A \quad \Gamma \vdash M_t : \neg\neg\mathsf{Str}^{\mathsf{c}}(A)}{\Gamma \vdash \mathsf{cons}^{\mathsf{c}}(M_h, M_t) : \mathsf{Str}^{\mathsf{c}}(A)} \quad \frac{\Gamma \vdash N : \mathsf{Str}^{\mathsf{c}}(A)}{\Gamma \vdash \mathsf{hd}^{\mathsf{c}}(N) : A} \quad \frac{\Gamma \vdash N : \mathsf{Str}^{\mathsf{c}}(A)}{\Gamma \vdash \mathsf{tl}^{\mathsf{c}}(N) : \neg\neg\mathsf{Str}^{\mathsf{c}}(A)}$$

β−reduction rules:

$$\begin{array}{rcl}
\mathsf{niter}^{\mathsf{c}}(\mathsf{o}^{\mathsf{c}}, P_o, u.\ P_s) & \to & P_o \\
\mathsf{niter}^{\mathsf{c}}(\mathsf{s}^{\mathsf{c}}(M), P_o, u.\ P_s) & \to & P_s[\lambda k.\ M\ (\lambda n.\ k\ \mathsf{niter}^{\mathsf{c}}(n, P_o, u.\ P_s))/u] \\
\mathsf{ncase}^{\mathsf{c}}(\mathsf{o}^{\mathsf{c}}, P_o, u.\ P_s) & \to & P_o \\
\mathsf{ncase}^{\mathsf{c}}(\mathsf{s}^{\mathsf{c}}(M), P_o, u.\ P_s) & \to & P_s[M/u] \\
\mathsf{hd}^{\mathsf{c}}(\mathsf{scoit}^{\mathsf{c}}(M, y.\ Q_h, y.\ Q_t)) & \to & Q_h[M/y] \\
\mathsf{tl}^{\mathsf{c}}(\mathsf{scoit}^{\mathsf{c}}(M, y.\ Q_h, y.\ Q_t)) & \to & \lambda k.\ Q_t[M/y]\ (\lambda m.\ k\ \mathsf{scoit}^{\mathsf{c}}(m, y.\ Q_h, y.\ Q_t)) \\
\mathsf{hd}^{\mathsf{c}}(\mathsf{cons}^{\mathsf{c}}(M_h, M_t)) & \to & M_h \\
\mathsf{tl}^{\mathsf{c}}(\mathsf{cons}^{\mathsf{c}}(M_h, M_t)) & \to & M_t
\end{array}$$

Figure 5: Typing and reduction rules of $\lambda_{\mathsf{Nat}^{\mathsf{c}},\mathsf{Str}^{\mathsf{c}}}$

Translation of types:

$$\begin{array}{rcl}
\textsc{cont}[\![\mathsf{Nat}]\!] & = & \neg\mathsf{Nat}^{\mathsf{c}} \\
\textsc{cont}[\![\mathsf{Str}(A)]\!] & = & \neg\mathsf{Str}^{\mathsf{c}}(\textsc{cpst}[\![A]\!])
\end{array}$$

Translation of objects:

$$\begin{array}{rcl}
[\![\mathsf{o}]\!]{:}K & = & K\ \mathsf{o}^{\mathsf{c}} \\
[\![\mathsf{s}(M)]\!]{:}K & = & K\ \mathsf{s}^{\mathsf{c}}(\textsc{cps}[\![M]\!]) \\
[\![\mathsf{niter}(N, P_o, u.\ P_s)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{niter}^{\mathsf{c}}(n, \textsc{cps}[\![P_o]\!], u.\ \textsc{cps}[\![P_s]\!][\underline{\lambda k'.}\ u\ \underline{\lambda u'.\ u'\ k'}/u])\ K) \\
[\![\mathsf{ncase}(N, P_o, u.\ P_s)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{ncase}^{\mathsf{c}}(n, [\![P_o]\!]{:}K, u.\ [\![P_s]\!]{:}K)) \\
[\![\mathsf{scoit}(M, y.\ Q_h, y.\ Q_t)]\!]{:}K & = & K\ \mathsf{scoit}^{\mathsf{c}}(\textsc{cps}[\![M]\!], y.\ \textsc{cps}[\![Q_h]\!], y.\ \lambda u'.\ u'\ \textsc{cps}[\![Q_t]\!]) \\
[\![\mathsf{cons}(M_h, M_t)]\!]{:}K & = & K\ \mathsf{cons}^{\mathsf{c}}(\textsc{cps}[\![M_h]\!], \textsc{cps}[\![M_t]\!]) \\
[\![\mathsf{hd}(N)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{hd}^{\mathsf{c}}(n)\ K) \\
[\![\mathsf{tl}(N)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{tl}^{\mathsf{c}}(n)\ K)
\end{array}$$

Figure 6: CPS translation of $\lambda_{\mathsf{Nat},\mathsf{Str}}$

Translation of types:

$$\textsc{cont}[\![\mathsf{Nat}]\!] \quad = \quad \mathsf{Str}^{\mathsf{c}}(\bot)$$

Translation of objects:

$$\begin{array}{rcl}
[\![\mathsf{o}]\!]{:}K & = & \mathsf{hd}^{\mathsf{c}}(K) \\
[\![\mathsf{s}(M)]\!]{:}K & = & \mathsf{tl}^{\mathsf{c}}(K)\ \textsc{cps}[\![M]\!] \\
[\![\mathsf{niter}(N, P_o, u.\ P_s)]\!]{:}K & = & [\![N]\!]{:}\mathsf{scoit}^{\mathsf{c}}(K, k'.\ [\![P_o]\!]{:}k', k'.\ \lambda u.\ [\![P_s]\!]{:}k') \\
[\![\mathsf{ncase}(N, P_o, u.\ P_s)]\!]{:}K & = & [\![N]\!]{:}\mathsf{cons}^{\mathsf{c}}([\![P_o]\!]{:}K, \lambda u.\ [\![P_s]\!]{:}K)
\end{array}$$

Figure 7: Alternative CPS translation of $\lambda_{\mathsf{Nat},\mathsf{Str}}$

Typing rules:

$$\frac{\Gamma \vdash M : A[\mu Z.\ A/Z]}{\Gamma \vdash \mathsf{in}_{Z.\ A}(M) : \mu Z.\ A} \qquad \frac{\Gamma \vdash N : \mu Z.\ A \quad \Gamma, u : A[C/Z] \vdash P : C}{\Gamma \vdash \mathsf{iter}(N, u.\ P) : C} \qquad \frac{\Gamma \vdash M : \mu Z.\ A}{\Gamma \vdash \mathsf{in}^{-1}(M) : A[\mu Z.\ A/Z]}$$

$$\frac{\Gamma \vdash M : D \quad \Gamma, y : D \vdash Q : A[D/Z]}{\Gamma \vdash \mathsf{coit}(M, y.\ Q) : \nu Z.\ A} \qquad \frac{\Gamma \vdash N : A[\nu Z.\ A/Z]}{\Gamma \vdash \mathsf{out}^{-1}(N) : \nu Z.\ A} \qquad \frac{\Gamma \vdash N : \nu Z.\ A}{\Gamma \vdash \mathsf{out}_{Z.\ A}(N) : A[\nu Z.\ A/Z]}$$

β-reduction rules:

$$\begin{array}{rcl}
\mathsf{iter}(\mathsf{in}_{Z.\ A}(M), u.\ P) & \to & P[Map_{Z.\ A}(M, n.\ \mathsf{iter}(n, u.\ P))/u] \\
\mathsf{in}^{-1}(\mathsf{in}_{Z.\ A}(M)) & \to & M \\
\mathsf{out}_{Z.\ A}(\mathsf{coit}(M, y.\ Q)) & \to & Map_{Z.\ A}(Q[M/y], m.\ \mathsf{coit}(m, y.\ Q)) \\
\mathsf{out}_{Z.\ A}(\mathsf{out}^{-1}(M)) & \to & M
\end{array}$$

Figure 8: Typing and reduction rules of $\lambda_{\mu,\nu}$

Translation of types:

$$\begin{array}{rcl}
\textsc{cont}[\![\mu Z.\ A]\!] & = & \neg\mu Z.\ \textsc{cpst}[\![A]\!] \\
\textsc{cont}[\![\nu Z.\ A]\!] & = & \neg\nu Z.\ \textsc{cpst}[\![A]\!]
\end{array}$$

Translation of objects:

$$\begin{array}{rcl}
[\![\mathsf{in}_{Z.\ A}(M)]\!]{:}K & = & K\ \mathsf{in}_{Z.\textsc{cpst}[\![A]\!]}(\textsc{cps}[\![M]\!]) \\
[\![\mathsf{iter}(N, u.\ P)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{iter}(n, u.\ \textsc{cps}[\![P]\!][Map_{Z.\textsc{cpst}[\![A]\!][Z/\neg\neg Z]}(u, x.\ \lambda k'.\ x\ (\lambda x'.\ x'\ k'))/u])\ K) \\
[\![\mathsf{in}^{-1}(N)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{in}^{-1}(n)\ K) \\
[\![\mathsf{coit}(M, y.\ Q)]\!]{:}K & = & K\ \mathsf{coit}(\textsc{cps}[\![M]\!], y.Map_{Z.\textsc{cpst}[\![A]\!][Z/\neg\neg Z]}(\textsc{cps}[\![Q]\!], x.\ \lambda u'.\ u'\ x)) \\
[\![\mathsf{out}^{-1}(M)]\!]{:}K & = & K\ \mathsf{out}^{-1}(\textsc{cps}[\![M]\!]) \\
[\![\mathsf{out}_{Z.\ A}(N)]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ \mathsf{out}_{Z.\textsc{cpst}[\![A]\!]}(n)\ K)
\end{array}$$

Figure 9: CPS translation of $\lambda_{\mu,\nu}$

Translation of types:

$$\textsc{cont}[\![\mu Z.\ A]\!] \quad = \quad \nu Z.\ \neg\textsc{cpst}[\![A]\!][Z/\neg Z]$$

Translation of objects:

$$\begin{array}{rcl}
[\![\mathsf{in}_{Z.\ A}(M)]\!]{:}K & = & \mathsf{out}_{Z.\ \neg\textsc{cpst}[\![A]\!][Z/\neg Z]}(K)\ \textsc{cps}[\![M]\!] \\
[\![\mathsf{iter}(N, u.\ P)]\!]{:}K & = & [\![N]\!]{:}\mathsf{coit}(K, k'.\ \lambda u.\ [\![P]\!]{:}k') \\
[\![\mathsf{in}^{-1}(N)]\!]{:}K & = & [\![N]\!]{:}\mathsf{out}^{-1}(\lambda x.\ x\ K)
\end{array}$$

Figure 10: Alternative CPS translation of $\lambda_{\mu,\nu}$

∇ is its intuitionistic counterpart, Ex falsum quodlibet. In programming terms, $\Delta x.\ x\ N$ may be read as `catch` x `in` N and $\nabla(x\ N)$ as `throw` $x\ N$.

Extending the translation of Sec. 3 with the clause

$$[\![\Delta x.\ N]\!]{:}K \quad = \quad ([\![N]\!]{:}(\lambda n.\ \nabla(n)))[\lambda h.\ h\ \lambda j.\ \lambda i.\ j\ K/x]$$

gives a CPS translation from $\lambda\Delta_{\mathsf{Nat}}$ to λ_{Nat} (with intuitionistic falsum). It is straightforward to prove the correctness of the extended CPS translation. Here, only convertibility is preserved.

PROPOSITION 3.

1. *If* $\Gamma \vdash M : A$, *then* $\text{CPST}[\![\Gamma]\!] \vdash \text{CPS}[\![M]\!] : \text{CPST}[\![A]\!]$.
2. *If* $M =_{\beta\Delta} M'$, *then* $\text{CPS}[\![M]\!] =_{\beta} \text{CPS}[\![M']\!]$.

As an illustration of the applicability of the translation, we show how it is possible to use its correctness to prove that every number-theoretic function representable in $\lambda\Delta_{\mathsf{Nat}}$ is also representable in $\lambda_{\mathsf{Nat},\mathsf{Nat}^{\mathsf{c}}}$.

Definition 1. Let $\sim$ be a congruence on the objects of a typed λ-calculus, A be a closed type and $|.|$ a mapping from $\mathbb{N}$ to closed objects of type A. A function $f : \mathbb{N} \to \mathbb{N}$ is $(\sim, A, |.|)$-representable if there exists a derivable judgment $x : N \vdash e : N$ such that, for every $n \in \mathbb{N}$, we have $e[|n|/x] \sim |f\ n|$.

In $\lambda\Delta_{\mathsf{Nat}}$, the traditional coding for $\mathbb{N}$ is Nat with $\lceil 0 \rceil = \mathsf{o}$ and $\lceil n+1 \rceil = \mathsf{s}(\lceil n \rceil)$, whereas, in $\lambda_{\mathsf{Nat},\mathsf{Nat}^{\mathsf{c}}}$, there is another possible coding, namely $\mathsf{Nat}^{\mathsf{c}}$ with $[\![0]\!] = \mathsf{o}^{\mathsf{c}}$ and $[\![n+1]\!] = \mathsf{s}^{\mathsf{c}}(\lambda k.\ k\ [\![n]\!])$. In order to establish the representability theorem, we must first relate the two representations. In the second item below, we set $\bot$ (an unspecified type thus far) to be Nat.

LEMMA 2.

1. *For every* $n \in \mathbb{N}$, *we have* $\text{CPS}[\![\lceil n \rceil]\!] = [\![n]\!]$;
2. *Let* $\bot = \mathsf{Nat}$. *There exist* $i : \mathsf{Nat}^{\mathsf{c}} \to \mathsf{Nat}$ *and* $j : \mathsf{Nat} \to \mathsf{Nat}^{\mathsf{c}}$ *such that for every* $n \in \mathbb{N}$, *we have* $i[\![n]\!] =_{\beta} \lceil n \rceil$ *and* $j\lceil n \rceil =_{\beta} [\![n]\!]$.

PROOF SKETCH. In (2), the conversion functions i and j are

$$\begin{array}{rcl} i & = & \lambda x.\ \mathsf{niter}^{\mathsf{c}}(x, \mathsf{o}, u.\ \mathsf{s}\ (u\ (\lambda z.\ z))) \\ j & = & \lambda x.\ \mathsf{niter}(x, \mathsf{o}^{\mathsf{c}}, u.\ \mathsf{s}^{\mathsf{c}}\ (\lambda k.\ k\ u)) \end{array}$$

□

We now turn to the representation theorem.

THEOREM 1. *Every function* $f : \mathbb{N} \to \mathbb{N}$ *that is* $(=_{\beta\Delta}, \mathsf{Nat}, \lceil . \rceil)$*-representable in* $\lambda\Delta_{\mathsf{Nat}}$ *is* $(=_{\beta}, \mathsf{Nat}, \lceil . \rceil)$*-representable in* $\lambda_{\mathsf{Nat},\mathsf{Nat}^{\mathsf{c}}}$.

PROOF SKETCH. The proof proceeds in two steps. First one shows that every function $f : \mathbb{N} \to \mathbb{N}$ that is $(=_{\beta\Delta}, \mathsf{Nat}, \lceil . \rceil)$-representable in $\lambda\Delta_{\mathsf{Nat}}$ is $(=_{\beta}, \mathsf{Nat}^{\mathsf{c}}, [\![.]\!])$-representable in $\lambda_{\mathsf{Nat},\mathsf{Nat}^{\mathsf{c}}}$ Then it remains to show that, in $\lambda_{\mathsf{Nat},\mathsf{Nat}^{\mathsf{c}}}$, every function that is $(=_{\beta}, \mathsf{Nat}^{\mathsf{c}}, [\![.]\!])$-representable in $\lambda_{\mathsf{Nat},\mathsf{Nat}^{\mathsf{c}}}$ is also $(=_{\beta}, \mathsf{Nat}, \lceil . \rceil)$-representable. □

7. CALCULUS OF INDUCTIVE CONSTRUCTIONS

In [8], Barthe, Hatcliff and Sørensen showed how the typed version of Plotkin's call-by-name CPS translation scales up for a large class of Pure Type Systems including the Calculus of Constructions (CC) and therefore also all smaller systems of Barendregt's λ-cube [5]. For technical simplicity, most of their work is cast in the framework of *domain-free* Pure Type Systems [9], a variant of Pure Type Systems where λ-abstractions do not carry the domain of their bound variable, i.e., are of the form $\lambda x.\ M$, in opposition to the standard, *domain-full* variant.

In this section, we examine whether the translation of CC from [8] and Sections 3, 4 are compatible and can be combined and generalized to yield, in particular, a CPS translation for the Calculus of Inductive Constructions (CIC) [40, 46], the base system of the Coq proof assistant. It turns out that the translations are compatible and can be combined, but only yield a CPS translation for a fragment of CIC. In particular, we show that small Σ-types and sum types with dependent eliminations cannot be CPS translated.

7.1 Calculus of Constructions

We adopt a stratified presentation of domain-free CC where a distinction is made between (raw) kinds, constructors, and objects at the syntax definition level. The language of domain-free CC (kinds, constructors, objects) is given by the grammar

$$\begin{array}{lcl} S, T & ::= & \star \mid \Pi x : A.\ T \mid \Pi X : S.\ T \\ A, B, D & ::= & X \mid \lambda x.\ A \mid B\ P \mid \lambda X.\ A \mid B\ D \\ & & \mid \Pi x : A.\ B \mid \Pi X : S.\ B \\ M, N, P & ::= & x \mid \lambda x.\ M \mid N\ P \mid \lambda X.\ M \mid N\ D \end{array}$$

The typing rules are given in Fig. 11. Because of the stratification, the presentation is slightly more verbose than those based on Pure Type Systems [5], but nevertheless equivalence between the two can be shown easily.

The CPS translation of domain-free CC from [8] is given in Fig. 12.

The translation preserves typing and reductions.

PROPOSITION 4 ([8]).

1. *If* $\Gamma \vdash a : b$, *then* $\text{CPST}[\![\Gamma]\!] \vdash \text{CPS}[\![a]\!] : \text{CPST}[\![b]\!]$;
2. *If* $a \twoheadrightarrow_{\beta} a'$, *then* $\text{CPS}[\![a]\!] \twoheadrightarrow_{\beta} \text{CPS}[\![a']\!]$.

It is straightforward to show that the correctness of the translation is preserved, if CC is extended with sums, products, positive inductive and coinductive types in the "non-dependent" forms introduced in previous sections. Stronger, "dependent" versions of these constructions, in contrast, raise a number of difficulties. These are discussed below.

7.2 Sigma-types

First, it is not possible to CPS translate small Σ-types along the same lines as product types. Consider CC extended with small Σ-types. The extended calculus features a new constructor form $\Sigma x : A.\ B$ and new object forms

$$\frac{\Gamma \vdash a : s}{\Gamma, x : a \vdash x : a} \qquad \frac{\Gamma \vdash a : b \quad \Gamma \vdash c : s}{\Gamma, x : c \vdash a : b} \qquad \frac{\Gamma \vdash a : b \quad \Gamma \vdash b' : s \quad b =_\beta b'}{\Gamma \vdash a : b'} \qquad \frac{}{\diamond \vdash \star : \Box}$$

$$\frac{\Gamma, x : A \vdash B : \star}{\Gamma \vdash (\Pi x : A.\ B) : \star} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.\ M : \Pi x : A.\ B} \qquad \frac{\Gamma \vdash N : (\Pi x : A.\ B) \quad \Gamma \vdash P : A}{\Gamma \vdash N\ P : B[P/x]}$$

$$\frac{\Gamma, X : S \vdash B : \star}{\Gamma \vdash (\Pi X : S.\ B) : \star} \qquad \frac{\Gamma, X : S \vdash M : B}{\Gamma \vdash \lambda X.\ M : \Pi X : S.\ B} \qquad \frac{\Gamma \vdash N : (\Pi X : S.\ B) \quad \Gamma \vdash D : S}{\Gamma \vdash N\ D : B[D/X]}$$

$$\frac{\Gamma, x : A \vdash T : \Box}{\Gamma \vdash (\Pi x : A.\ T) : \Box} \qquad \frac{\Gamma, x : A \vdash A' : T}{\Gamma \vdash \lambda x.\ A' : \Pi x : A.\ T} \qquad \frac{\Gamma \vdash B : (\Pi x : A.\ T) \quad \Gamma \vdash P : A}{\Gamma \vdash B\ P : T[P/x]}$$

$$\frac{\Gamma, X : S \vdash T : \Box}{\Gamma \vdash (\Pi X : S.\ T) : \Box} \qquad \frac{\Gamma, X : S \vdash A : T}{\Gamma \vdash \lambda X.\ A : \Pi X : S.\ T} \qquad \frac{\Gamma \vdash B : (\Pi X : S.\ T) \quad \Gamma \vdash D : S}{\Gamma \vdash B\ D : T[D/X]}$$

Figure 11: Typing rules for domain-free Calculus of Constructions

Objects

$$\begin{array}{rcl}
\textsc{cps}[\![M]\!] & = & \lambda k.\ [\![M]\!]{:}k \\
[\![x]\!]{:}K & = & x\ K \\
[\![\lambda x.\ M]\!]{:}K & = & K\ (\lambda x.\ \textsc{cps}[\![M]\!]) \\
[\![N\ P]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ n\ \textsc{cps}[\![P]\!]\ K) \\
[\![\lambda X.\ M]\!]{:}K & = & K\ (\lambda X.\ \textsc{cps}[\![M]\!]) \\
[\![N\ D]\!]{:}K & = & [\![N]\!]{:}(\lambda n.\ n\ \textsc{cps}[\![D]\!]\ K)
\end{array}$$

Constructors

$$\begin{array}{rcl}
\textsc{cpst}[\![A]\!] & = & \neg\neg\textsc{cps}[\![A]\!] \\
\textsc{cps}[\![X]\!] & = & X \\
\textsc{cps}[\![\lambda x.\ A]\!] & = & \lambda x.\ \textsc{cps}[\![A]\!] \\
\textsc{cps}[\![B\ P]\!] & = & \textsc{cps}[\![B]\!]\ \textsc{cps}[\![P]\!] \\
\textsc{cps}[\![\lambda X.\ A]\!] & = & \lambda X.\ \textsc{cps}[\![A]\!] \\
\textsc{cps}[\![B\ D]\!] & = & \textsc{cps}[\![B]\!]\ \textsc{cps}[\![D]\!] \\
\textsc{cps}[\![\Pi x : A.\ B]\!] & = & \Pi x : \textsc{cpst}[\![A]\!].\ \textsc{cpst}[\![B]\!] \\
\textsc{cps}[\![\Pi X : S.\ B]\!] & = & \Pi X : \textsc{cpst}[\![S]\!].\ \textsc{cpst}[\![B]\!]
\end{array}$$

Kinds

$$\begin{array}{rcl}
\textsc{cpst}[\![S]\!] & = & \textsc{cps}[\![S]\!] \\
\textsc{cps}[\![\star]\!] & = & \star \\
\textsc{cps}[\![\Pi x : A.\ T]\!] & = & \Pi x : \textsc{cpst}[\![A]\!].\ \textsc{cpst}[\![T]\!] \\
\textsc{cps}[\![\Pi X : S.\ T]\!] & = & \Pi X : \textsc{cpst}[\![S]\!].\ \textsc{cpst}[\![T]\!]
\end{array}$$

Box

$$\textsc{cpst}[\![\Box]\!] \quad = \quad \Box$$

Contexts

$$\begin{array}{rcl}
\textsc{cpst}[\![\diamond]\!] & = & \bot : \star \\
\textsc{cpst}[\![\Gamma, x{:}C]\!] & = & \textsc{cpst}[\![\Gamma]\!], x{:}\textsc{cpst}[\![C]\!] \\
\textsc{cpst}[\![\Gamma, X{:}K]\!] & = & \textsc{cpst}[\![\Gamma]\!], X{:}\textsc{cpst}[\![K]\!]
\end{array}$$

Figure 12: CPS translation of domain-free Calculus of Constructions

$\langle M_1, M_2\rangle$, $\mathsf{fst}(N)$, $\mathsf{snd}(N)$. The typing rules are

$$\frac{\Gamma \vdash A : \star \quad \Gamma, x : A \vdash B : \star}{\Gamma \vdash \Sigma x : A.\ B : \star}$$

$$\frac{\Gamma \vdash M_1 : A \quad \Gamma \vdash M_2 : B[M_1/x]}{\Gamma \vdash \langle M_1, M_2\rangle : \Sigma x : A.\ B}$$

$$\frac{\Gamma \vdash N : \Sigma x : A.\ B}{\Gamma \vdash \mathsf{fst}(N) : A} \qquad \frac{\Gamma \vdash N : \Sigma x : A.\ B}{\Gamma \vdash \mathsf{snd}(N) : B[\mathsf{fst}(N)/x]}$$

and the β-rules are the obvious ones.

We would like to set

$$\textsc{cps}[\![\Sigma x : A.\ B]\!] \quad = \quad \Sigma x : \textsc{cpst}[\![A]\!].\ \textsc{cpst}[\![B]\!]$$

and

$$\begin{aligned}
[\![\langle M_1, M_2\rangle]\!]{:}K &= K\ \langle \textsc{cps}[\![M_1]\!], \textsc{cps}[\![M_2]\!]\rangle \\
[\![\mathsf{fst}(N)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \mathsf{fst}(n)\ K) \\
[\![\mathsf{snd}(N)]\!]{:}K &= [\![N]\!]{:}(\lambda n.\ \mathsf{snd}(n)\ K)
\end{aligned}$$

It is routine to check that the translation preserves reduction. But it does not preserve typing and the problem arises with the second projection. Indeed, consider a rule application

$$\frac{\Gamma \vdash N : \Sigma x : A.\ B}{\Gamma \vdash \mathsf{snd}(N) : B[\mathsf{fst}(N)/x]}$$

and assume that the translation property holds for the premiss, i.e.,

$$\textsc{cpst}[\![\Gamma]\!] \vdash \textsc{cps}[\![N]\!] : \neg\neg\Sigma x : \textsc{cpst}[\![A]\!].\ \textsc{cpst}[\![B]\!]$$

The translation property for the conclusion is

$$\textsc{cpst}[\![\Gamma]\!] \vdash \lambda k.\ \textsc{cps}[\![N]\!]\ (\lambda n.\ \mathsf{snd}(n)\ k) : \textsc{cpst}[\![B[\mathsf{fst}(N)/x]]\!]$$

For this to hold, we must (by successive applications of generation) have

$$\begin{aligned}
&\textsc{cpst}[\![\Gamma]\!], k : \neg\textsc{cps}[\![B[\mathsf{fst}(N)/x]]\!], n : \Sigma x : \textsc{cpst}[\![A]\!].\ \textsc{cpst}[\![B]\!] \\
&\quad \vdash \mathsf{snd}(n) : \textsc{cpst}[\![B[\mathsf{fst}(N)/x]]\!]
\end{aligned}$$

which is not the case.

The difficulty of CPS translating Σ-types seems closely related to the difficulties of extracting the constructive content of the axiom of choice, see, e.g., [10].

Small Σ-types with small non-dependent split instead of projections, on the other hand, are unproblematic. Instead of object forms $\mathsf{fst}(N)$, $\mathsf{snd}(N)$, they come with an object form $\mathsf{split}(N, (x, u).\ P)$ with the typing rule

$$\frac{\Gamma \vdash N : \Sigma x : A.\ B \quad \Gamma \vdash C : \star \quad \Gamma, x : A, u : B \vdash P : C}{\Gamma \vdash \mathsf{split}(N, (x, u).\ P) : C}$$

and the β-reduction rule

$$\mathsf{split}(\langle M_1, M_2\rangle, (x, u).\ P) \quad \to \quad P[M_1, M_2/x, u]$$

The CPS translation is:

$$\begin{aligned}
&[\![\mathsf{split}(N, (x, u).\ P)]\!]{:}K \\
&\quad = \quad [\![N]\!]{:}(\lambda n.\ \mathsf{split}(n, (x, u).\ \textsc{cps}[\![P]\!])\ K)
\end{aligned}$$

Also unproblematic for CPS translation are large Σ-types, i.e. types $\Sigma x : A.T$. (Types $\Sigma x : T.\ B$ are inconsistent.)

7.3 Sum types

Another extension for which CPS translating fails is sum types in their dependent version, i.e., with dependent eliminations. More generally, this failure applies to the dependent version of any type constructions with case-like eliminations (colimit-like types), e.g., natural numbers with dependent primitive recursion or Σ-types with dependent split.

Let CC_+ stand for CC extended with a constructor form $A{+}B$ and object forms $\mathsf{inl}(M)$, $\mathsf{inr}(M)$, $\mathsf{case}^\star(N, u.\ P_1, u.\ P_2)$ and $\mathsf{case}^\Box(N, u.\ P_1, u.\ P_2)$. The typing rules are

$$\frac{\Gamma \vdash A : \star \quad \Gamma \vdash B : \star}{\Gamma \vdash A + B : \star}$$

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \mathsf{inl}(M) : A + B} \qquad \frac{\Gamma \vdash M : B}{\Gamma \vdash \mathsf{inr}(M) : A + B}$$

$$\frac{\begin{array}{l}\Gamma \vdash N : A + B \\ \Gamma, x : A + B \vdash C : \star \\ \Gamma, u : A \vdash P_1 : C[\mathsf{inl}(u)/x] \\ \Gamma, u : B \vdash P_2 : C[\mathsf{inr}(u)/x]\end{array}}{\Gamma \vdash \mathsf{case}^\star(N, u.\ P_1, u.\ P_2) : C[N/x]}$$

$$\frac{\begin{array}{l}\Gamma \vdash N : A + B \\ \Gamma, x : A + B \vdash T : \Box \\ \Gamma, u : A \vdash P_1 : T[\mathsf{inl}(u)/x] \\ \Gamma, u : B \vdash P_2 : T[\mathsf{inr}(u)/x]\end{array}}{\Gamma \vdash \mathsf{case}^\Box(N, u.\ P_1, u.\ P_2) : T[N/x]}$$

and the β-reduction rules are the obvious ones.

Just as in the previous subsection with Σ-types, it is easy to verify that the obvious candidate CPS translation ignoring large case fails to be type-correct; moreover, for large case, even constructing a reasonable candidate CPS counterpart fails. But one can actually prove a crisp result: no type-correct CPS translation for CC_+ can exist extending that for CC.

Recall that in CC, a classical operator is inconsistent with dependently eliminated sum types, see e.g., [14, 21]. Indeed, such a combination allows one to construct a retract pair from $\star$ to the type $1 + 1$ of booleans. In $\mathrm{CC}\Delta_+$, the type $\Pi X : \star.\ X + (X \to 0)$ is inhabited by

$$\mathsf{lem} \quad = \quad \lambda X.\ \Delta x.\ x\ \mathsf{inr}(\lambda z.\ x\ \mathsf{inl}(z))$$

The two functions $\epsilon : \mathsf{Bool} \to \star$, $E : \star \to \mathsf{Bool}$ forming the retract pair are

$$\begin{aligned}
\epsilon &= \lambda x.\ \mathsf{case}^\Box(x, u.\ 1, u.\ 0) \\
E &= \lambda X.\ \mathsf{case}^\star(\mathsf{lem}\ X, u.\ \mathsf{inl}(\langle\rangle), u.\ \mathsf{inr}(\langle\rangle))
\end{aligned}$$

The proofs $p_1 : \Pi X : \star.\ X \to \epsilon\ (E\ X)$, $p_2 : \Pi X : \star.\ \epsilon\ (E\ X) \to X$ showing that ϵ and E give a retract pair indeed are

$$\begin{aligned}
p_1 &= \lambda X.\ \lambda x.\ \mathsf{case}^\star(\mathsf{lem}\ X, u.\ \langle\rangle, u.\ u\ x) \\
p_2 &= \lambda X.\ \mathsf{case}^\star(\mathsf{lem}\ X, u.\ \lambda x.\ u, u.\ \lambda x.\ \nabla(x))
\end{aligned}$$

Further, T. Coquand [14] shows that retract pairs from $\star$ to a small type yield an inconsistency in CC. Hence $\mathrm{CC}\Delta_+$ is inconsistent.

Because of the inconsistency, there is a closed object M in $\mathrm{CC}\Delta_+$ such that $\vdash M : 0$. Now, if a type-correct CPS translation existed for CC_+ and therefore also for $\mathrm{CC}\Delta_+$, then, in CC_+, we would have

$$\bot : \star \vdash \textsc{cps}[\![M]\!](\lambda z.\ \nabla(z)) : \bot$$

which contradicts its consistency.

In a recent paper [22], H. Geuvers proved a similar result, namely the non-derivability of induction principles in $\lambda P2$. Despite being close in spirit, the results are unrelated, since we work in a type system with inductive types and Geuvers does not. In fact, he shows explicitly that his techniques cannot be adapted immediately to yield non-derivability results for type systems with inductive types and dependent case analysis.

7.4 Summary

Summing up, the CPS translation of arrow types scales smoothly up for Π-types, but those of product and sum do not extend for small Σ-types in their strong version and sum types with dependent eliminations. Further we have shown that the CPS translation for the λ-cube does not admit any type-correct extension to sum types with dependent elimination. To conclude this section, we would like to contrast our results with [47], where H. Xi and C. Schürmann prove the correctness of a CPS translation for DML, and with [43], where Z. Shao and coauthors prove the correctness of a CPS translation for TL. In these papers, the type systems under consideration have considerable expressive power, but are not "deeply" dependently typed. In particular, DML indexes have no computational role and DML functions cannot be defined by dependent case analysis.

8. CONCLUSION

This paper highlights a number of important characteristics of CPS translations for (co)inductive types.

On the positive side, we showed that feasible CPS translations exist for (co)inductive types in simply typed λ-calculi and, moreover, they function in the presence of control operators and generalize for dependently typed calculi where case-like eliminations are only allowed in non-dependent forms.

On the negative side, we showed that no CPS translation along the standard lines can be defined for the full Calculus of Inductive Constructions, as the projects fails for small Σ-types and sum types with dependent case analysis.

Issues for further type-theoretical study include CPS translations for alternative formulations of (co)inductive types such as (co)inductive types with recursors à la Mendler, commuting reductions, direct-style (DS) translations and correspondences in the style of [7], but also the implications of our study for program extraction.

The practical implications of our results remain to be investigated, but we believe that our analysis can be of interest for practitioners interested in the use and design and (dependently) typed programming languages.

Acknowledgments

We are grateful to our four anonymous referees for a number of remarks and suggestions which helped us improve the paper.

The second author was supported by the Portuguese Foundation for Science and Technology under grant No. PRAXIS XXI/C/EEI/14172/98 and by the Estonian Science Foundation under grant No. 4155. The authors also acknowledge support from the cooperation program ICCTI/INRIA.

9. REFERENCES

[1] S. Abramsky, ed. *Proc. of 5th Int. Conf. on Typed Lambda Calculi and Appl., TLCA'01*, vol. 2044 of *Lect. Notes in Comp. Sci.*. Springer-Verlag, 2001.

[2] T. Altenkirch. Representations of first order function types as terminal coalgebras. In [1], pp. 8–21.

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

[4] F. Barbanera and S. Berardi. A symmetric lambda calculus for classical program extraction. *Information and Computation*, 125(2):103–117, 1996.

[5] H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaums, eds., *Handbook of Logic in Computer Science, Vol. 2: Background: Computational Structures*, pp. 117–309. Clarendon Press, 1992.

[6] G. Barthe, J. Hatcliff, and M. H. Sørensen. A notion of classical pure type system (preliminary version). In S. Brookes and M. Mislove, eds., *Proc. of 13th Ann. Conf. on Math. Found. of Programming Semantics, MFPS'97*, vol. 6 of *Elect. Notes in Theor. Comp. Sci.*. Elsevier, 1997.

[7] G. Barthe, J. Hatcliff, and M. H. Sørensen. Reflections on reflections. In H. Glaser, P. Hartel, and H. Kuchen, eds., *Proc. of 9th Int. Symp. on Programming Languages: Implementations, Logics, and Programs, PLILP'97*, vol. 1292 of *Lect. Notes in Comp. Sci.*, pp. 241–258. Springer-Verlag, 1997.

[8] G. Barthe, J. Hatcliff, and M. H. Sørensen. CPS-translations and applications: the cube and beyond. *Higher-Order and Symbolic Computation*, 12(2):125–170, 1999.

[9] G. Barthe and M. H. Sørensen. Domain-free pure type systems. *J. of Funct. Prog.*, 10(5):417–452, 2000.

[10] S. Berardi, M. Bezem, and T. Coquand. A realization of the negative interpretation of the axiom of choice. In [16], pp. 47–62.

[11] U. Berger. A constructive interpretation of positive inductive definitions. Unpublished draft, 1995.

[12] U. Berger and H. Schwichtenberg. Program extraction from classical proofs. In D. Leivant, ed., *Selected Papers from Int. Workshop on Logical and Computational Complexity, LCC'94*, vol. 960 of *Lect. Notes in Comp. Sci.*, pp. 77–97. Springer-Verlag, 1995.

[13] R. Constable and C. R. Murthy. Finding computational content in classical proofs. In G. Huet and G. Plotkin, eds., *Logical Frameworks*, pp. 341–362. Cambridge University Press, 1990.

[14] T. Coquand. Metamathematical investigations of a calculus of constructions. In [37], pp. 91–122.

[15] T. Coquand and C. Paulin. Inductively defined types (preliminary version). In P. Martin-Löf and G. Mints, eds., *Proc. of Int. Conf. on Computer Logic, COLOG'88*, vol. 417 of *Lect. Notes in Comp. Sci.*, pp. 50–66. Springer-Verlag, 1990.

[16] M. Dezani-Ciancaglini and G. Plotkin, eds. *Proc. of 2nd Int. Conf. on Typed Lambda Calculi and Appl., TLCA'95*, vol. 902 of *Lect. Notes in Comp. Sci.*. Springer-Verlag, 1995.

[17] M. Felleisen, D. P. Friedman, E. Kohlbecker, and B. Duba. *Theor. Comp. Sci.*, 52(3):205–237, 1987.

[18] M. Felleisen and A. Sabry. Continuations in programming practice: Introduction and survey.

Manuscript, 1999.

[19] FLINT project. `http://flint.cs.yale.edu`

[20] H. Geuvers. Inductive and coinductive types with iteration and recursion. In B. Nordström, K. Pettersson and G. Plotkin, eds., *Proc. of Workshop on Types for Proofs and Programs, TYPES'92*, pp. 193-217. Dept. of Comp. Sci., Chalmers Univ. of Techn. and Göteborg Univ, 1992.

[21] H. Geuvers. Inconsistency of classical logic in type theory. Unpublished note, 2001.

[22] H. Geuvers. Induction is not derivable in second order dependent type theory. In [1], pp. 166–181.

[23] T. G. Griffin. The formulae-as-types notion of control. In *Conf. Record of 17th Annual ACM Symp. on Principles of Programming Languages, POPL'90*, pp. 47–57. ACM Press, 1990.

[24] P. de Groote. A CPS-Translation of the $\lambda\mu$-Calculus. In S. Tison, ed., *Proc. of 19th Int. Coll. on Trees in Algebra and Programming, CAAP'94*, vol. 787 of *Lect. Notes in Comp. Sci.*, pp. 85–99. Springer-Verlag, 1994.

[25] P. de Groote. A simple calculus of exception handling. In [16], pp. 201–215.

[26] P. de Groote. On the strong normalization of natural deduction with permutation- conversions. In P. Narendran and M. Rusinowitch, eds., *Proc. of 10th Int. Conf. on Rewriting Techniques and Appl., RTA'99*, vol. 1631 of *Lect. Notes in Comp. Sci.*, pp. 45–59. Springer-Verlag, 1999.

[27] R. Harper and M. Lillibridge. Polymorphic type assignment and CPS conversion. *LISP and Symbolic Computation*, 6(4):361–380, 1993.

[28] R. Harper and M. Lillibridge. Explicit polymorphism and CPS conversion. In *Conf. Record of 20th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'93*, pp. 206–219. ACM Press, 1993.

[29] D. Leivant. Contracting proofs to programs. In [37], pp. 279–327.

[30] R. Matthes. *Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types.* PhD thesis, Ludwig-Maximilians-Universität München, 1998.

[31] N. P. Mendler. Inductive types and type constraints in the second-order lambda-calculus. *Ann. of Pure and Appl. Logic*, 51(1–2):159–172, 1991.

[32] A. R. Meyer and M. Wand. Continuation semantics in typed lambda-calculi (summary). In R. Parikh, ed., *Proc. of 3rd Workshop on Logics of Programs*, vol. 193 of *Lect. Notes in Comp. Sci.*, pp. 219–224. Springer-Verlag, 1985.

[33] C. R. Murthy. An evaluation semantics for classical proofs. In *Proc. of 6th Ann. IEEE Symp. on Logic in Computer Science, LICS'91*, pp. 96–107. IEEE CS Press, 1991.

[34] H. Nakano. A constructive logic behind the catch and throw mechanism. *Ann. of Pure and Appl. Logic*, 69(2–3):269–301, 1994.

[35] C.-H. L. Ong. A semantic view of classical proofs: type-theoretic, categorical, and denotational characterizations (preliminary extended abstract). In *Proc. of 11th Ann. IEEE Symp. on Logic in Computer Science, LICS'96*, pp. 230–241. IEEE CS Press, 1996.

[36] C.-H. L. Ong and C. A. Stewart. A Curry-Howard foundation for functional computation with control. In *Conf. Record of 24th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'97*, pp. 215–227, ACM Press, 1997.

[37] P. Odifreddi, ed. *Logic and Computer Science*, vol. 31 of *APIC Studies in Data Processing.* Academic Press, 1990.

[38] M. Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In A. Voronkov, ed., *Proc. of Int. Conf. on Logic Programming and Automated Reasoning, LPAR'92*, vol. 624 of *Lect. Notes in Comp. Sci.*, pp. 190–201. Springer-Verlag, 1992.

[39] M. Parigot. Classical proofs as programs. In G. Gottlob, A. Leitsch, and D. Mundici, eds., *Proc. of 3rd Kurt Gödel Colloquium, KGC'93*, vol. 713 of *Lecture Notes in Computer Science*, pp. 263–276. Springer-Verlag, 1993.

[40] C. Paulin-Mohring. Inductive definitions in the system Coq: Rules and properties. In M. Bezem and J. F. Groote, eds., *Proc. of 1st Int. Conf. on Typed Lambda Calculi and Appl., TLCA'93*, vol. 664 of *Lect. Notes in Computer Science*, pp. 328–345. Springer-Verlag, 1993.

[41] G. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theor. Comp. Sci.*, 1(2):125–159, 1975.

[42] N. J. Rehof and M. H. Sørensen. The λ_Δ calculus. In M. Hagiya and J. Mitchell, eds., *Proc. of 2nd Int. Symp. on Theor. Aspects of Comp. Sci., TACS'94*, vol. 789 of *Lect. Notes in Comp. Sci.*, pp. 516–542. Springer-Verlag, 1994.

[43] Z. Shao, B. Saha, V. Trifonov, and N. Papaspyrou. A type system for certified binaries. To appear in *Conf. Record of 29th Ann. ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'02.* ACM Press, 2002.

[44] TAL project. `http://www.cs.cornell.edu/talc`

[45] T. Uustalu and V. Vene. Least and greatest fixedpoints in intuitionistic natural deduction. *Theor. Comp. Sci.*, to appear.

[46] B. Werner. *Méta-théorie du Calcul des Constructions Inductives.* PhD thesis, Université Paris 7, 1994.

[47] H. Xi and C. Schürmann. CPS transform and type derivations for Dependent ML. In *Proc. of 8th Workshop on Logic, Language, Information and Computation, WoLLIC'01*, 2001.