Classical Logic with Mendler Induction
A Dual Calculus and its Strong Normalization

Marco Devesas Campos* and Marcelo Fiore**

Computer Laboratory, University of Cambridge, United Kingdom

Abstract. We investigate (co-)induction in Classical Logic under the propositions-as-types paradigm, considering propositional, second-order, and (co-)inductive types. Specifically, we introduce an extension of the Dual Calculus with a Mendler-style (co-)iterator that remains strongly normalizing under head reduction. We prove this using a non-constructive realizability argument.

Keywords: Mendler Induction, Classical Logic, Curry-Howard isomorphism, Dual Calculus, Realizability

1 Introduction

The Curry-Howard Isomorphism The interplay between Logic and Computer Science has a long and rich history. In particular, the Curry-Howard isomorphism, the correspondence between types and theorems, and between typings and proofs, is a long established bridge through which results in one field can fruitfully migrate to the other. One such example, motivating of the research presented herein, is the use of typing systems based on Gentzen's sequent calculus *LK* [10]. At its core, *LK* is a calculus of the dual concepts of necessary assumptions and possible conclusions—which map neatly, on the Computer Science side, to required inputs (or computations) and possible outputs (or continuations).

Classical Calculi The unconventional form of *LK* belies an extreme symmetry and regularity that make it more amenable to analysis than other systems that can be encoded in it. Indeed, Gentzen introduced *LK* as an intermediate step in his proof that Hilbert-style derivation systems and his own system of Natural Deduction, *NK*, were consistent. Curry-Howard descendants of *LK* are Curien and Herbelin's $\lambda\mu\tilde{\mu}$ [6] and Wadler's Dual Calculus [19]. As an example of the kind of analysis that can be done using sequents, these works focused on establishing syntacticly the duality of the two most common evaluations strategies for the lambda-calculus: call-by-name and call-by-value. While originally Classical calculi included only propositional types—i.e. conjunction, disjunction, negation,

* This work was supported by the United Kingdom's Engineering and Physical Sciences Research Council [grant number EP/J500380/1].
** Partially supported by ERC ECSYM.

implication and subtraction (the dual connective of implication)—they were later extended with second-order types [13, 17], and also with *positive* (co-)inductive types [13]; the latter fundamentally depended on the map operation of the underlying type-schemes.

Mendler Induction In continuing with this theme, we turn our attention here to a more general induction scheme due to Mendler [15]. Originally, this induction scheme was merely seen as an ingenious use of polymorphism that allowed induction to occur without direct use of mapping operations. However, it was later shown that with Mendler's iterator one could in fact induct on data-types of *arbitrary variance*—i.e. data-types whose induction variable may also appear negatively [14, 18]. Due to its generality, Mendler Induction has been applied in a number of different contexts, amongst which we find higher-order recursive types [1, 2] and automated theorem proving [12].

Classical Logic and *Mendler Induction* Can one export Mendler Induction to non-functional settings without introducing unexpected side-effects? Specifically, can one extend Classical Logic with Mendler Induction without losing consistency? Note that Classical Logic has been shown to be quite misbehaved if not handled properly [11]; and certain forms of Mendler Induction have been shown to break strong normalization at higher-ranked types [2].

This paper answers both questions affirmatively. In summary, we:

- extend the second-order Dual Calculus with functional types—viz., with arrow and subtractive types (Section 2);
- prove its strong normalization (Section 3) via a realizability argument (a lattice-theoretic distillation of Parigot's proof for the Symmetric Lambda-calculus [3, 16]);
- recall the idea underlying Mendler Induction in the functional setting (Section 4);
- present our extension of the Dual Calculus with Mendler (co-)inductive types and argue why functional types are indispensable to its definition (Section 5); and
- extend the aforementioned realizability argument to give a non-constructive proof that the extension is also strongly normalizing (Section 6).

2 Second-order Dual Calculus

The Base Calculus Our base formalism is Wadler's Dual Calculus [19]—often abbreviated DC. We begin by reviewing the original propositional version extended with second-order types [13] and subtractive types [5, 6]. Tables 1[1], 2, and 3 respectively summarize the syntax, the types and typing rules, and the reduction rules of the calculus.

[1] Unlike Wadler's presentation, we keep the standard practice of avoiding suffix operators; whilst lexical duality is lost, we think it improves readability.

Syntax The sequent calculus LK is a calculus of multiple assumptions and conclusions, as witnessed by the action of the right and left derivation rules. Similarly, the two main components of DC are split into two kinds: *terms* (or *computations*) which, intuitively, produce values; and *co-terms* (or *continuations*), which consume them. However, whereas in the sequent calculus one can mix the different kinds of rules in any order, to keep the computational connection, the term and co-term formation rules are restricted in what phrases they expect—e.g. pairs should combine values, while projections pass the components of a pair to some other continuation. This distinction also forces the existence of two kinds of variables: variables for terms and co-variables for co-terms. We assume that they belong to some disjoint and countably infinite sets Var and $Covar$, respectively.

Terms

$$t := x, y, \ldots \in Var \mid \underbrace{\langle t, t' \rangle \mid i_1\langle t\rangle \mid i_2\langle t\rangle \mid not\langle k\rangle \mid \lambda x.(t) \mid (t \# k) \mid a\langle t\rangle \mid e\langle t\rangle}_{\text{Introductions}} \mid \alpha.(c)$$

Co-terms

$$k := \alpha, \beta, \ldots \in Covar \mid \underbrace{\left[k, k'\right] \mid fst[k] \mid snd[k] \mid not[t] \mid (t \mathbin{@} k) \mid \mu\alpha.(k) \mid a[k] \mid e[k]}_{\text{Eliminations}} \mid x.(c)$$

Cuts

$$c := t \bullet k$$

Table 1. Syntax of the second-order Dual Calculus

Cuts and Abstractions The third and final kind of phrase in the Dual Calculus are *cuts*. Recall the famous dictum of Computer Science:

$$\text{Data-structures} + \text{Algorithms} = \text{Programs} \ .$$

In DC, where terms represent the creation of information and co-terms consume it, we find that cuts, the combination of a term with a continuation, are analogous to programs:

$$\text{Terms} + \text{Co-terms} = \text{Cuts} \ ;$$

they are the entities that are capable of being executed. Given a cut, one can consider the computation that would ensue if given data for a variable or co-variable. The calculus provides a mechanism to express such situations by means of *abstractions* $x.(c)$ and of *co-abstractions* $\alpha.(c)$ on any cut c. Abstractions are continuations—they expect values in order to proceed with some execution—and, dually, co-abstractions are computations.

Subtraction One novelty of this paper is the central role given to subtractive types, $A - B$ [5]. Subtraction is the dual connective to implication; it is to continuations what implication is to terms: it allows one to abstract co-variables in co-terms—and thereby *compose continuations*. Given a continuation k where a co-variable α might appear free, the subtractive abstraction (or catch, due to its connection with exception handling) is defined as $\mu\alpha.(k)$, the idea being that applying (read, cutting) a continuation k' and value t to it, *packed* together as $(t \# k')$, yields a cut of the form $t \bullet k[k'/\alpha]$.

Typing Judgments We present the types and the typing rules in Table 2; we omit the structural rules here but they can be found in the aforementioned paper by Wadler [19]. We have three forms of typing judgments that go hand-in-hand with the three different types of phrases: $\Gamma \vdash t : A \mid \Delta$ for terms, $\Gamma \mid k : A \dashv \Delta$ for co-terms, and $\Gamma \vdash c \dashv \Delta$ for cuts. In all cases, the entailment symbols point to the phrase under judgment, and they appear in the same position as they would appear in the corresponding sequent of LK. *Typing contexts* Γ assign variables to their assumed types; dually, *typing co-contexts* Δ assign co-variables to their types. Tacitly, we assume that they always include the free (co-)variables in the phrase under consideration. Type-schemes $F(X)$ are types in which a distinguished type variable X may appear free; the instantiation of such a type-scheme to a particular type T is simply the substitution of the distinguished X by T and is denoted $F(T)$.

Example: Witness the Lack of Witness We can apply the rules in Table 2 to bear proof of valid formulas in second-order Classical Logic. One such example at the second-order level is $\neg\forall X.T \to \exists X.\neg T$:

$$\mid not[a\langle\alpha.(e\langle not\langle\alpha\rangle\rangle \bullet \beta)\rangle] : \neg\forall X.T \dashv \beta : \exists X.\neg T \ .$$

Note how the existential does not construct witnesses but simply diverts the flow of execution (by use of a co-abstraction).

Head Reduction The final ingredient of the calculus is the set of (head) reduction rules (Table 3). They are non-deterministic—as a cut made of abstractions and co-abstractions can reduce by either one of the abstraction rules—and non-confluent. Confluence can be reestablished by prioritizing the reduction of one type of abstraction over the other; this gives rise to two confluent reduction disciplines that we term *abstraction prioritizing* and *co-abstraction prioritizing*. In any case, reduction of well-typed cuts yields well-typed cuts.[2]

[2] As we are not looking at call-by-name and call-by-value we do not use the same reduction rule for implication as Wadler [19]; the rule here is due to Curien and Herbelin [6].

Types

$$T, A, B := X \mid A \wedge B \mid A \vee B \mid \neg A \mid A \to B \mid A - B \mid \forall X.T \mid \exists X.T$$

Identity

$$\frac{}{x : A \vdash x : A \mid} \qquad \frac{}{\mid \alpha : A \dashv \alpha : A}$$

Abstractions

$$\frac{\Gamma \vdash c \dashv \Delta, \alpha : A}{\Gamma \vdash \alpha.(c) : A \mid \Delta} \qquad \frac{x : A, \Gamma \vdash c \dashv \Delta}{\Gamma \mid x.(c) : A \dashv \Delta}$$

Cut

$$\frac{\Gamma \vdash t : A \mid \Delta \qquad \Gamma \mid k : A \dashv \Delta}{\Gamma \vdash t \bullet k \dashv \Delta}$$

Conjunction

$$\frac{\Gamma \vdash t : A \mid \Delta \qquad \Gamma \vdash t' : B \mid \Delta}{\Gamma \vdash \langle t, t' \rangle : A \wedge B \mid \Delta}$$

$$\frac{\Gamma \mid k : A \dashv \Delta}{\Gamma \mid \mathit{fst}[k] : A \wedge B \dashv \Delta} \qquad \frac{\Gamma \mid k : B \dashv \Delta}{\Gamma \mid \mathit{snd}[k] : A \wedge B \dashv \Delta}$$

Disjunction

$$\frac{\Gamma \vdash t : A \mid \Delta}{\Gamma \vdash i_1\langle t \rangle : A \vee B \mid \Delta} \qquad \frac{\Gamma \vdash t : B \mid \Delta}{\Gamma \vdash i_2\langle t \rangle : A \vee B \mid \Delta}$$

$$\frac{\Gamma \mid k : A \dashv \Delta \qquad \Gamma \mid k' : B \dashv \Delta}{\Gamma \mid [k, k'] : A \vee B \dashv \Delta}$$

Negation

$$\frac{\Gamma \mid k : A \dashv \Delta}{\Gamma \vdash \mathit{not}\langle k \rangle : \neg A \mid \Delta} \qquad \frac{\Gamma \vdash t : A \mid \Delta}{\Gamma \mid \mathit{not}[t] : \neg A \dashv \Delta}$$

Implication

$$\frac{x : A, \Gamma \vdash t : B \mid \Delta}{\Gamma \vdash \lambda x.(t) : A \to B \mid \Delta} \qquad \frac{\Gamma \vdash t : A \mid \Delta \quad \Gamma \mid k : B \dashv \Delta}{\Gamma \mid (t @ k) : A \to B \dashv \Delta}$$

Subtraction

$$\frac{\Gamma \vdash t : A \mid \Delta \quad \Gamma \mid k : B \dashv \Delta}{\Gamma \vdash (t \# k) : A - B \mid \Delta} \qquad \frac{\Gamma \mid k : A \dashv \Delta, \alpha : B}{\Gamma \mid \mu\alpha.(k) : A - B \dashv \Delta}$$

Universal Quantification

$$\frac{\Gamma \vdash t : F(X) \mid \Delta}{\Gamma \vdash a\langle t \rangle : \forall X.F(X) \mid \Delta}\ (X \text{ not free in } \Gamma, \Delta) \qquad \frac{\Gamma \mid k : F(A) \dashv \Delta}{\Gamma \mid a[k] : \forall X.F(X) \dashv \Delta}$$

Existential Quantification

$$\frac{\Gamma \vdash t : F(A) \mid \Delta}{\Gamma \vdash e\langle t \rangle : \exists X.F(X) \mid \Delta} \qquad \frac{\Gamma \mid k : F(X) \dashv \Delta}{\Gamma \mid e[k] : \exists X.F(X) \dashv \Delta}\ (X \text{ not free in } \Gamma, \Delta)$$

Table 2. Typing for the second-order propositional Dual Calculus (with the structural rules omitted).

$$\langle t, t' \rangle \bullet fst[k] \rightsquigarrow t \bullet k \qquad \langle t, t' \rangle \bullet snd[k] \rightsquigarrow t' \bullet k$$
$$i_1\langle t \rangle \bullet [k, k'] \rightsquigarrow t \bullet k \qquad i_2\langle t \rangle \bullet [k, k'] \rightsquigarrow t \bullet k'$$
$$not\langle k \rangle \bullet not[t] \rightsquigarrow t \bullet k$$
$$\lambda x.(t) \bullet (t' @ k) \rightsquigarrow t[t'/x] \bullet k \qquad (t \# k) \bullet \mu\alpha.(k') \rightsquigarrow t \bullet k'[k/\alpha]$$
$$a\langle t \rangle \bullet a[k] \rightsquigarrow t \bullet k \qquad e\langle t \rangle \bullet e[k] \rightsquigarrow t \bullet k$$
$$\alpha.(c) \bullet k \rightsquigarrow c[k/\alpha] \qquad t \bullet x.(c) \rightsquigarrow c[t/x]$$

Table 3. Head reduction for the second-order Dual Calculus

3 Strong Normalization of the Second-order Dual Calculus

The Proof of Strong Normalization Having surveyed the syntax, types and reduction rules of DC, we will now give a proof of its strong normalization—i.e., that all reduction sequences of well-typed cuts terminate in a finite number of steps—for the given *non-deterministic reduction* rules. It will follow, then, that the deterministic sub-calculi, where one prioritizes the reduction of one kind abstraction over the other, are also strongly normalizing.

The proof rests on a realizability interpretation for terms. Similar approaches for the propositional fragment can be found in the literature [17, 9]; however, the biggest influence on our proof was the one by Parigot for the second-order extension of the Symmetric Lambda-Calculus [16]. Our main innovation is the identification of a complete lattice structure with fix-points suitable for the interpretation of (co-)inductive types. We will, in fact, need to consider two lattices: $\mathcal{OP}$ and $\mathcal{ONP}$. In $\mathcal{OP}$, we find, intuitively, all the terms/co-terms of types. In the lattice $\mathcal{ONP}$ we find *only* terms/co-terms that are introductions/eliminations; these correspond, again intuitively, to values/co-values of types. Between these two classes we have type-directed *actions* from $\mathcal{OP}$ to $\mathcal{ONP}$, and a completion operator $\Downarrow$ from $\mathcal{ONP}$ to $\mathcal{OP}$ that generates all terms/co-terms compatible with the given values/co-values.

$$\mathcal{OP} \underset{\Downarrow}{\overset{\wedge,\vee,\neg,\ldots}{\rightleftarrows}} \mathcal{ONP} \tag{1}$$

In this setting, we give (two) mutually induced interpretations for types (one in $\mathcal{ONP}$ and the other in $\mathcal{OP}$, Table 4) and establish an adequacy result (Theorem 4) from which strong normalization follows as a corollary. The development is outlined next.

Sets of Syntax The set of all terms formed using the rules in Table 1 will be denoted by $\mathcal{T}$; similarly, co-terms will be $\mathcal{K}$ and cuts $\mathcal{C}$. We will also need three

special subsets of those sets: $\mathcal{IT}$ for those terms whose outer syntactic form is an introduction; $\mathcal{EK}$, dually, for the co-terms whose outer syntactic form is an eliminator; and $\mathcal{SN}$ for the set of strongly-normalizing cuts.[3]

Syntactic Actions on Sets The syntactic constructors give rise to obvious *actions* on sets of terms, co-terms, and cuts; e.g.

$$- \bullet - : \mathcal{P}(\mathcal{T}) \times \mathcal{P}(\mathcal{K}) \to \mathcal{P}(\mathcal{C}) \ , \quad T \bullet K = \{t \bullet k \mid t \in T, k \in K\} \ .$$

By abuse of notation these operators shall be denoted as their syntactic counterparts; they are basic to our realizability interpretation.

Restriction under Substitution The substitution operation lifts point-wise to the level of sets as a monotone function $(-)[(=)/\phi] : \mathcal{P}(U) \times \mathcal{P}(V) \to \mathcal{P}(U)$ for V the set of terms (resp. co-terms), ϕ a variable (resp. co-variable), and U either the set of terms, co-terms, or cuts. We will make extensive use of the right adjoint $(-)\Big|_{\phi}^{Q}$ to $(-)[Q/\phi]$ characterized by

$$R[Q/\phi] \subseteq P \quad \textit{iff} \quad R \subseteq P\Big|_{\phi}^{Q} \ ,$$

and that we term the *restriction under substitution*. With it we can, e.g., express the set of cuts that are strongly normalizing when free occurrences of a co-variable α are substituted by co-terms from a set K:

$$\mathcal{SN}\Big|_{\alpha}^{K} = \{\, c \in \mathcal{C} \mid \text{for all } k \in K \ .\ c[k/\alpha] \in \mathcal{SN} \,\} \ .$$

Orthogonal Pairs Whenever a term t and a co-term k form a strongly normalizing cut $t \bullet k$, we say that they are *orthogonal*. Similarly, for sets T of terms and K of co-terms, we say that they are orthogonal if $T \bullet K \subseteq \mathcal{SN}$. We call pairs of such sets *orthogonal pairs*, and the set of all such pairs $\mathcal{OP}$. For any orthogonal pair $P \in \mathcal{OP}$, its set of terms is denoted $(P)^{\mathsf{T}}$ and its set of co-terms by $(P)^{\mathsf{K}}$. Note that no type restriction is in play in the definition of orthogonal pairs; e.g. a cut of an injection with a projection is by definition orthogonal as no reduction rule applies.

Lattices Recall that a lattice S is a partially ordered set such that any *non-empty* finite subset $S' \subseteq S$ has a least upper bound (or join, or lub) and a greatest lower-bound (or meet, or glb), respectively denoted by $\bigvee S'$ and $\bigwedge S'$. If the bounds exist for any subset of S one says that the lattice is *complete*. In particular, this entails the existence of a bottom and a top element for the partial order. The powerset $\mathcal{P}(S)$ of a set S is a complete lattice under inclusion; the dual L^{op} of a (complete) lattice L (where we take the opposite order and invert the bounds) is a (complete) lattice, as is the point-wise product of any two (complete) lattices.

[3] A non-terminating, non-well-typed cut: $\alpha.(\mathit{not}\langle\alpha\rangle \bullet \alpha) \bullet \mathit{not}[\alpha.(\mathit{not}\langle\alpha\rangle \bullet \alpha)]$.

Proposition 1 (Lattice Structure of $\mathcal{OP}$). *The set of orthogonal pairs is a sub-lattice of $\mathcal{P}(\mathcal{T}) \times \mathcal{P}(\mathcal{K})^{op}$. Explicitly, for $P, Q \in \mathcal{OP}$,*

$$P \leq Q \quad \textit{iff} \quad (P)^{\mathsf{T}} \subseteq (Q)^{\mathsf{T}} \textit{ and } (P)^{\mathsf{K}} \supseteq (Q)^{\mathsf{K}} \ ;$$

the join and meet of arbitrary non-empty sets $S \subseteq \mathcal{OP}$ are

$$\bigvee S \equiv \left(\bigcup_{P \in S} (P)^{\mathsf{T}}, \bigcap_{P \in S} (P)^{\mathsf{K}} \right) \qquad \bigwedge S \equiv \left(\bigcap_{P \in S} (P)^{\mathsf{T}}, \bigcup_{P \in S} (P)^{\mathsf{K}} \right) \ .$$

Moreover, it is complete with empty join and meet given by $\bot \equiv (\emptyset, \mathcal{K})$ and $\top \equiv (\mathcal{T}, \emptyset)$.

Orthogonal Normal Pairs The other lattice we are interested in is the lattice $\mathcal{ONP}$ of what we call *orthogonal normal pairs*. These are orthogonal pairs which are made out at the outermost level by introductions and eliminators. Logically speaking, they correspond to those proofs whose last derivation is a left or right operational rule. Computationally, they correspond to the narrowest possible interpretations of values and co-values. Orthogonal normal pairs inherit the lattice structure of $\mathcal{OP}$ but for the empty lub and glb which become $\bot \equiv (\emptyset, \mathcal{EK})$ and $\top \equiv (\mathcal{IT}, \emptyset)$.

Type Actions Pairing together the actions of the introductions and eliminations of a given type allows us to construct elements of $\mathcal{ONP}$ whenever we apply them to orthogonal sets—in particular, then, when these sets are the components of elements of $\mathcal{OP}$—as witnessed by the following proposition.

Proposition 2. *For $P, Q \in \mathcal{OP}$ and $S \subseteq \mathcal{OP}$, the following definitions determine elements of $\mathcal{ONP}$:*

$$\begin{aligned}
P \wedge Q &= \Big(\Big\langle (P)^{\mathsf{T}}, (Q)^{\mathsf{T}} \Big\rangle, \mathit{fst}\Big[(P)^{\mathsf{K}} \Big] \cup \mathit{snd}\Big[(Q)^{\mathsf{K}} \Big] \Big) \\
P \vee Q &= \Big(i_1 \Big\langle (P)^{\mathsf{T}} \Big\rangle \cup i_2 \Big\langle (Q)^{\mathsf{T}} \Big\rangle, \Big[(P)^{\mathsf{K}}, (Q)^{\mathsf{K}} \Big] \Big) \\
\neg P &= \Big(\mathit{not} \Big\langle (P)^{\mathsf{K}} \Big\rangle, \mathit{not} \Big[(P)^{\mathsf{T}} \Big] \Big) \\
P \to Q &= \bigvee_{x \in Var} \Big(\lambda x. \Big((Q)^{\mathsf{T}} \Big|_{x}^{(P)^{\mathsf{T}}} \Big), \Big((P)^{\mathsf{T}} @ (Q)^{\mathsf{K}} \Big) \Big) \\
P - Q &= \bigwedge_{\alpha \in Covar} \Big(\Big((P)^{\mathsf{T}} \# (Q)^{\mathsf{K}} \Big), \mu\alpha. \Big((P)^{\mathsf{K}} \Big|_{\alpha}^{(Q)^{\mathsf{K}}} \Big) \Big) \\
\forall S = \bigwedge_{P \in S} \Big(a \Big\langle (P)^{\mathsf{T}} \Big\rangle, a \Big[(P)^{\mathsf{K}} \Big] \Big) \qquad & \exists S = \bigvee_{P \in S} \Big(e \Big\langle (P)^{\mathsf{T}} \Big\rangle, e \Big[(P)^{\mathsf{K}} \Big] \Big)
\end{aligned}$$

Orthogonal Completion Now that we have interpretations for the actions that construct values/co-values of a type in $\mathcal{ONP}$, we need to go the other way (cf. Diagram 1, above) to $\mathcal{OP}$, so that we also include (co-)variables and (co-)abstractions in our interpretations. So, for orthogonal sets of values T and of co-values K, the term and co-term completions of T and K are respectively defined as:

$$[T](L) = \mathit{Var} \cup T \cup \bigcup_{\alpha \in \mathit{Covar}} \alpha.\big(\mathcal{SN}\big|_{\alpha}^{L}\big) \ , \quad [K](U) = \mathit{Covar} \cup K \cup \bigcup_{x \in \mathit{Var}} x.\big(\mathcal{SN}\big|_{x}^{U}\big) \, .$$

Due to the non-determinism associated with the reduction of (co-)abstractions, we need guarantee that all added (co-)abstractions are compatible not only with the starting set of values, but also with any (co-)abstractions that have been added in the process—and vice-versa. In other words, we need to iterate this process by taking the least fix-point:

$$(\underline{\, T \Vdash K}) = \big(\mathit{lfp}([T] \circ [K]) \, , \, [K] \, (\mathit{lfp}([T] \circ [K])) \big) \ .$$

(In fact, as has been remarked elsewhere [3, 16], all one needs is *a* fix-point.)

Theorem 3. *Let $N \in \mathcal{ONP}$ be an orthogonal normal pair; its structural completion $\Vdash N$ is an orthogonal pair:*

$$\Vdash N = \Big(\underline{(N)^{T} \Big\Vert (N)^{K}} \Big) \in \mathcal{OP} \ .$$

Interpretations Given a type T and a (suitable) mapping γ from its free type variables, $\mathrm{ftv}(T)$, to $\mathcal{ONP}$—called the interpretation context—we define (Table 4) two interpretations, as orthogonal pairs and as orthogonal normal pairs, by mutual induction on the structure of T. They both satisfy the weakening and substitution properties. The extension of an interpretation context γ where a type-variable X is mapped to $N \in \mathcal{ONP}$ is denoted by $\gamma[X \mapsto N]$.

Theorem 4 (Adequacy). *Let t, k and c stand for terms, co-terms and cuts of the Dual Calculus. For any typing context Γ and co-context Δ, and type T such that*

$$\Gamma \vdash t : T \mid \Delta \ , \qquad \Gamma \mid k : T \dashv \Delta \ , \qquad \Gamma \vdash c \dashv \Delta \ ,$$

and for any suitable interpretation context γ for Γ, Δ and T, and any substitution σ satisfying

$$(x : A) \in \Gamma \implies \sigma(x) \in (\!|A|\!)(\gamma))^{T} \quad \textit{and} \quad (\alpha : A) \in \Delta \implies \sigma(\alpha) \in (\!|A|\!)(\gamma))^{K},$$

we have that

$$t[\sigma] \in (\!|T|\!)(\gamma))^{T} \ , \quad k[\sigma] \in (\!|T|\!)(\gamma))^{K} \ , \quad c[\sigma] \in \mathcal{SN} \ .$$

Corollary 5 (Strong Normalization). *Every well-typed cut of DC is strongly normalizing.*

$$
\begin{array}{rl}
[\![T]\!](\gamma) : \mathcal{ONP} & (\!|T|\!)(\gamma) : \mathcal{OP} \\
[\![X]\!](\gamma) = \gamma(X) & (\!|T|\!)(\gamma) = \coprod([\![T]\!](\gamma)) \\
[\![A \wedge B]\!](\gamma) = (\!|A|\!)(\gamma) \wedge (\!|B|\!)(\gamma) & \\
[\![A \vee B]\!](\gamma) = (\!|A|\!)(\gamma) \vee (\!|B|\!)(\gamma) & \\
[\![\neg A]\!](\gamma) = \neg(\!|A|\!)(\gamma) & \\
[\![A \to B]\!](\gamma) = (\!|A|\!)(\gamma) \to (\!|B|\!)(\gamma) & \\
[\![A - B]\!](\gamma) = (\!|A|\!)(\gamma) - (\!|B|\!)(\gamma) & \\
[\![\forall\, X\,.\, A]\!](\gamma) = \forall\{(\!|A|\!)(\gamma[X \mapsto N]) \mid N \in \mathcal{ONP}\} & \\
[\![\exists\, X.\, A]\!](\gamma) = \exists\{(\!|A|\!)(\gamma[X \mapsto N]) \mid N \in \mathcal{ONP}\} &
\end{array}
$$

Table 4. Interpretations of the second-order Dual Calculus in $\mathcal{ONP}$ and $\mathcal{OP}$.

4 Mendler Induction

Having covered the first theme of the paper, Classical Logical in its Dual Calculus guise, let us focus in this section on the second theme we are exploring: Mendler Induction. As the concept may be rather foreign, it is best to review it informally in the familiar functional setting.

Inductive Definitions Roughly speaking, an inductive definition of a function is one in which the function being defined can be used in its own definition provided that it is applied only to values of strictly smaller character than the input. The fix-point operator

$$
\begin{array}{l}
\mathit{fix} : \big((\mu X.F(X) \to A) \to \mu X.F(X) \to A\big) \to \mu X.F(X) \to A \\
\mathit{fix}\ f\ x = f\ (\mathit{fix}\ f)\ x
\end{array}
$$

associated to the inductive type $\mu X.F(X)$ arising from a type scheme $F(X)$, clearly violates induction, and indeed breaks strong normalization: one can feed it the identity function to yield a looping term. One may naively attempt to tame this behavior by considering the following modified fix-point operator

$$
\begin{array}{l}
\mathit{fix}' : \big((\mu X.F(X) \to A) \to F\big(\mu X.F(X)\big) \to A\big) \to \mu X.F(X) \to A \\
\mathit{fix}'\ f\ (\mathit{in}\ x') = f\ (\mathit{fix}'\ f)\ x'
\end{array}
$$

in which, for the introduction $\mathit{in} : F\big(\mu X.F(X)\big) \to \mu X.F(X)$, one may regard x' as being of strictly smaller character than $\mathit{in}(x')$. Of course, this is still unsatisfactory as, for instance, we have the looping term $\mathit{fix}'\ (\lambda f.\ f \circ \mathit{in})$. The problem here is that the functional $\lambda f.\ f \circ \mathit{in} : (\mu X.F(X) \to A) \to F\big(\mu X.F(X)\big) \to A$ of which we are taking the fix-point takes advantage of the concrete type $F\big(\mu X.F(X)\big)$ of x' used in the recursive call.

Mendler Induction The ingenuity of Mendler Induction is to ban such perversities by restricting the type of the functionals that the iterator can be applied to: these should not rely on the inductive type but rather be abstract; in other words, be represented by a fresh type variable X as in the typing below[4]:

$$\begin{aligned} &\mathit{mitr} : \big((X \to A) \to F(X) \to A\big) \to \mu X.F(X) \to A \\ &\mathit{mitr}\ f\ (\mathit{min}\ x) = f\ (\mathit{mitr}\ f)\ x \end{aligned}$$

for *min* the introduction $F\big(\mu X.F(X)\big) \to \mu X.F(X)$.

Note that if the type scheme $F(X)$ is endowed with a polymorphic mapping operation $\mathit{map}_F : (A \to B) \to F(A) \to F(B)$, every term $a : F(A) \to A$ has as associated catamorphism $\mathit{cata}(a) \equiv \mathit{mitr}\ \big(\lambda f\,.\ a \circ (\mathit{map}_F\ f)\big) : \mu X.F(X) \to A$. In particular, one has $\mathit{cata}(\mathit{map}_F\ \mathit{min}) : \mu X.F(X) \to F\big(\mu X.F(X)\big)$.

5 Dual Calculus with Mendler Induction

Mendler Induction We shall now formalize Mendler Induction in the Classical Calculus of Section 2. Additionally, we shall also introduce its dual, Mendler co-Induction. This requires: type constructors; syntactic operations corresponding to the introductions and eliminations, and their typing rules; and reduction rules. These are summarized in Table 5. First, we take a type scheme $F(X)$ and represent its inductive type by $\mu X.F(X)$—dually, we represent the associated co-inductive type by $\nu X.F(X)$.

Syntax As usual, the inductive introduction, $\mathit{min}\langle - \rangle$, witnesses that the values of the unfolding of the inductive type $F(\mu X.F(X))$ are injected in the inductive type $\mu X.F(X)$. It is in performing induction that we consume values of inductive type and, hence, the *induction operator* (or *iterator*, or *inductor*), $\mathit{mitr}_{\rho,\alpha}[k,l]$ corresponds to an elimination. It is comprised of an iteration step k, an output continuation l, and two distinct induction co-variables, ρ and α. We postpone the explanation of their significance for the section on reduction below, but note now that the iterator binds ρ and α in the iteration continuation but not in the output continuation; thus, e.g.,

$$\big(\mathit{mitr}_{\rho,\alpha}[k,l]\big)[k'/\rho][l'/\alpha] = \mathit{mitr}_{\rho,\alpha}[k, l[k'/\rho][l'/\alpha]]\ .$$

The co-inductive operators, $\mathit{mcoitr}_{r,x}\langle t,u\rangle$ and $\mathit{mout}[k]$, are obtained via dualization. In particular, the co-inductive eliminator, $\mathit{mout}[k]$, witnesses that the co-values k of type $F(\nu X.F(X))$ translate into co-values of $\nu X.F(X)$.

[4] We note that the original presentation of this inductive operator [15] was in System F and, accordingly, the operator considered instead functionals of type $\forall X.(X \to A) \to F(X) \to A$. Cognoscenti will recognize that this type is the type-theoretic Yoneda reformulation $\forall X.(X \to A) \to T(X)$ of $T(A) = F(A) \to A$ for $T(X) = F(X) \to A$.

Types

$$T := \ldots \mid \mu X.F(X) \mid \nu X.F(X)$$

Syntax

$$t := \ldots \mid \underbrace{\ldots \mid min\langle t\rangle \mid mcoitr_{r,x}\langle t, t'\rangle}_{\text{Introductions}} \mid \ldots$$
$$k := \ldots \mid \underbrace{\ldots \mid mitr_{\rho,\alpha}[k, k'] \mid mout[k]}_{\text{Eliminations}} \mid \ldots$$

Reduction

$$\begin{aligned} min\langle t\rangle \bullet mitr_{\rho,\alpha}[k,l] &\rightsquigarrow t \bullet k[\mu\alpha.(mitr_{\rho,\alpha}[k,\alpha])/\rho][l/\alpha] \\ mcoitr_{r,x}\langle t,u\rangle \bullet mout[k] &\rightsquigarrow t[\lambda x.(mcoitr_{r,x}\langle t,x\rangle)/r][u/x] \bullet k \end{aligned}$$

Typing rules

$$\frac{\Gamma \vdash t : F(\mu X.F(X)) \mid \Delta}{\Gamma \vdash min\langle t\rangle : \mu X.F(X) \mid \Delta}$$

$$\frac{\Gamma \mid k : F(X) \dashv \Delta, \rho : X - A, \alpha : A \qquad \Gamma \mid l : A \dashv \Delta}{\Gamma \mid mitr_{\rho,\alpha}[k,l] : \mu X.F(X) \dashv \Delta}$$

(X not free in Γ, Δ, A)

$$\frac{x : A, r : A \to X, \Gamma \vdash t : F(X) \mid \Delta \qquad \Gamma \vdash u : A \mid \Delta}{\Gamma \vdash mcoitr_{r,x}\langle t,u\rangle : \nu X.F(X) \mid \Delta}$$

(X not free in Γ, Δ, A)

$$\frac{\Gamma \mid k : F(\nu X.F(X)) \dashv \Delta}{\Gamma \mid mout[k] : \nu X.F(X) \dashv \Delta}$$

Table 5. Extension of the second-order Dual Calculus with Mendler Induction

Reduction To reduce an inductive cut $min\langle t\rangle \bullet mitr_{\rho,\alpha}[k,l]$, we start by passing the unwrapped inductive value t to the induction step k. However, in the spirit of Mendler Induction, the induction step must be instantiated with the induction itself *and*, because we are in a Classical calculus, the output continuation—this is where the parameter co-variables come into play. The first co-variable, ρ, receives the induction; the induction step may call this co-variable (using a cut) arbitrarily and it must also be able to capture the output of those calls—in other words, it needs to *compose this continuation* with other continuations; therefore one needs to pass $\mu\alpha.(mitr_{\rho,\alpha}[k,\alpha])$, the induction with the output continuation (subtractively) abstracted. The other co-variable, α, represents in k the output of the induction—which for a call $mitr_{\rho,\alpha}[k,l]$ is l[5]. For co-induction, we dualize—in particular, the co-inductive call expects the lambda-abstraction of the co-inductive step.

Typing Lastly, we have the typing rules that force induction to be well-founded. Recall that this was achieved in the functional setting by forcing the inductive step to take an argument of arbitrary instances of the type scheme $F(X)$. Here we do the same. In typing $mitr_{\rho,\alpha}[k,l]$ for $\mu X.F(X)$ we require k to have type $F(X)$ where X is a variable that appears nowhere in the derivation except in the (input) type of the co-variable ρ.

Example: Naturals Let us look at a concrete example: natural numbers under the abstraction prioritizing strategy. We posit a distinguished type variable $\mathcal{B}$, and from it construct the type $1 \equiv \mathcal{B} \vee \neg\mathcal{B}$, which is inhabited by the witness of the law of the excluded middle, $* \equiv \alpha.(i_2\langle not\langle x.(i_1\langle x\rangle \bullet \alpha)\rangle\rangle \bullet \alpha)$. The base type scheme for the naturals is $F(X) \equiv 1 \vee X$, and the naturals are then defined as $\mathcal{N} \equiv \mu X.F(X)$. Examples of this type are:

$$\text{zero} \equiv min\langle i_1\langle *\rangle\rangle \ , \quad \text{one} \equiv min\langle i_2\langle \text{zero}\rangle\rangle \ , \text{ and } \quad \text{two} \equiv min\langle i_2\langle \text{one}\rangle\rangle \ .$$

For any continuation k on $\mathcal{N}$, the successor "function" is defined as the following continuation for $\mathcal{N}$

$$succk^k \equiv x.(min\langle i_2\langle x\rangle\rangle \bullet k) \qquad (x \notin \text{fv}(k)) \ .$$

Example: Addition The above primitives are all we need to define addition of these naturals. The inductive step "add m to" is

$$Step^m_{\rho,\alpha} \equiv \big[x.(m \bullet \alpha), x.\big((x \# \ succk^\alpha) \bullet \rho\big)\big] \ .$$

Theorem 6. *Let n and m stand for the encoding of two natural numbers and the encoding of their sum be (by abuse of notation) $n+m$. Under the abstraction prioritizing reduction rule,*

$$n \bullet mitr_{\rho,\alpha}\big[Step^m_{\rho,\alpha}, l\big] \rightsquigarrow^* (n+m) \bullet l \ .$$

[5] One may wonder if the output continuation is strictly necessary. As outputs appear on the right of sequents, and the induction is already a left-rule, the only possible alternative would be to add a co-variable to represent it. However, under this rule the system would no longer be closed under substitution [13].

6 Strong Normalization for Mendler Induction

We now come to the main contribution of the paper: the extension of the Orthogonal Pairs realizability interpretation of the second-order Dual Calculus (Section 3) to Mendler Induction, which establishes that the extension is strongly normalizing.

Lattice Structure The extension begins with the reformulation of the sets $\mathcal{SN}$, $\mathcal{T}$, $\mathcal{K}$, $\mathcal{C}$, $\mathcal{IT}$, and $\mathcal{EK}$ so that they accommodate the (co-)inductive operators. Modulo these changes, the definitions of $\mathcal{OP}$ and $\mathcal{ONP}$ remain the same; so do the actions for propositional and second order types, and the orthogonal completion, $\perp\!\!\!\perp$. All that remains, then, is to give suitable definitions for the (co-)inductive actions and the interpretations of (co-)inductive types.

Inductive Restrictions The reduction rule for Mendler Induction is unlike any other of the calculus. When performing an inductive step for $mitr_{\rho,\alpha}[k,l]$, the bound variable ρ will be only substituted by one specific term: $\mu\alpha.(mitr_{\rho,\alpha}[k,\alpha])$. One needs a different kind of restriction to encode this invariant: take K and L to be sets of co-terms (intuitively, where the inductive step and output continuation live) and define the inductive restriction by

$$K \,/^{\rho}_{\alpha}\, L \equiv \{\, k \in \mathcal{K} \mid \text{for all } l \in L,\ k[\mu\alpha.(mitr_{\rho,\alpha}[k,\alpha])/\rho][l/\alpha] \in K \,\} \ ;$$

and also for co-induction, for sets of terms T and U:

$$T \,/^{r}_{x}\, U \equiv \{\, t \in \mathcal{T} \mid \text{for all } u \in U,\ t[\lambda x.(mcoitr_{r,x}\langle t, x\rangle)/r][u/x] \in T \,\} \ .$$

Mendler Pairing Combining the inductive restriction with the inductive introduction/elimination set operations, we can easily create orthogonal normal pairs—much as we did for the propositional actions—from two given orthogonal pairs: one intuitively standing for the interpretation of $F(\mu F.F(X))$ and the other for the output type. However, the interpretation of the inductive type should not depend on a specific choice of output type but should accept *all* instantiations of output, as well as *all* possible induction co-variables; model-wise this corresponds to taking a meet over all possible choices for the parameters:

$$\mathrm{MuP}(P) = \bigwedge_{\substack{Q \in \mathcal{OP} \\ \rho \neq \alpha \in Covar}} \Big(min\Big\langle (P)^{\mathsf{T}} \Big\rangle, mitr_{\rho,\alpha}\Big[(P)^{\mathsf{K}} \Big/^{\rho}_{\alpha} (Q)^{\mathsf{K}}, (Q)^{\mathsf{K}} \Big] \Big) \in \mathcal{ONP} \ ;$$

and similarly for its dual, NuP:

$$\mathrm{NuP}(P) = \bigvee_{\substack{Q \in \mathcal{OP} \\ r \neq x \in Var}} \Big(mcoitr_{r,x}\Big\langle (P)^{\mathsf{T}} \Big/^{r}_{x} (Q)^{\mathsf{T}}, (Q)^{\mathsf{T}} \Big\rangle, mout\Big[(P)^{\mathsf{K}} \Big] \Big) \in \mathcal{ONP} \ .$$

Monotonization The typing constraints on Mendler Induction correspond—model-wise—to a monotonization step. This turns out to be what we need to guarantee that an inductive type can be modeled by a least fix-point; without this step, the interpretation of a type scheme would be a function on lattices that would not necessarily be monotone. There are two possible universal ways to induce monotone endofunctions from a given endofunction f on a lattice: the first one, $\lceil f \rceil$, we call the monotone extension and use it for inductive types, the other one, the monotone restriction $\lfloor f \rfloor$, will be useful for co-inductive types. Their definitions[6] are:

$$\lceil f \rceil\, x \equiv \bigvee_{y \leq x} f\, y \qquad \text{and} \qquad \lfloor f \rfloor\, x \equiv \bigwedge_{x \leq y} f\, y \ .$$

They are, respectively, the least monotone function above and the greatest monotone function below f. Necessarily, by Tarski's fix-point theorem, they both have least and greatest fix-points; in particular we have $\mathit{lfp}(\lceil f \rceil)$ and $\mathit{gfp}(\lfloor f \rfloor)$.

Inductive Actions Combining the above ingredients, one can define the actions corresponding to inductive and to co-inductive types. They are parametrized by functions $f : \mathcal{ONP} \to \mathcal{OP}$,

$$\mu f \equiv \mathit{lfp}(\lceil \mathrm{MuP} \circ f \rceil) \in \mathcal{ONP} \quad \text{and} \quad \nu f \equiv \mathit{gfp}(\lfloor \mathrm{NuP} \circ f \rfloor) \in \mathcal{ONP} \ .$$

Interpretations For (co-)inductive types associated to a type-scheme $F(X)$ and mappings $\rho : \mathrm{ftv}(\mu X.F(X)) \to \mathcal{ONP}$ (the context) we set

$$[\![\mu X.F(X)]\!](\gamma) = \mu(\!|F(X)|\!)(\gamma[X \mapsto -])\,,\ [\![\nu X.F(X)]\!](\gamma) = \nu(\!|F(X)|\!)(\gamma[X \mapsto -])\,;$$

while their *orthogonal interpretation* is as before. These interpretations also satisfy the weakening and substitution properties.

Classically Reasoning about Mendler Induction Mendler's original proof of strong normalization for his induction principle in a functional setting was already classical [15]. For us, this issue centers around the co-term component of the interpretation of inductive types (and, dually, the term component of co-inductive types). Roughly, the induction hypothesis of the adequacy theorem states that for any $N \in \mathcal{ONP}$, $m \in (\!|X - A|\!)(\gamma[X \mapsto N]))^{\mathsf{K}}$, $l \in (\!|A|\!)(\gamma))^{\mathsf{K}}$, and realizability substitution σ we have

$$k[\sigma][m/\rho][l/\alpha] \in (\!|F(X)|\!)(\gamma[X \mapsto N]))^{\mathsf{K}} \ , \tag{2}$$

and if we were to prove that $\mathit{mitr}_{\rho,\alpha}[k[\sigma], l[\sigma]] \in ([\![\mu X.F(X)]\!](\gamma))^{\mathsf{K}}$ just by the fix-point property of the interpretation, we would need to have

$$(k[\sigma])[\mu\alpha.(\mathit{mitr}_{\rho,\alpha}[k[\sigma], \alpha])/\rho][l/\alpha] \in (\!|F(X)|\!)(\gamma[X \mapsto [\![\mu X.F(X)]\!](\gamma)]))^{\mathsf{K}}$$

[6] Cognoscenti will recognize that they are point-wise Kan extensions.

for arbitrary $l \in (\!|A|\!)(\gamma))^{\mathsf{K}}$. Instantiating Formula 2 to the case when N is the interpretation of our fix-point, $[\![\mu X.F(X)]\!](\gamma)$, we see that in order to prove that $mitr_{\rho,\alpha}[k[\sigma], l[\sigma]] \in ([\![\mu X.F(X)]\!](\gamma))^{\mathsf{K}}$ we would need to prove that for any $l' \in (\!|A|\!)(\gamma))^{\mathsf{K}}$ we have that $mitr_{\rho,\alpha}[k[\sigma], l'] \in (\!|\mu X.F(X)|\!)(\gamma))^{\mathsf{K}}$—a circularity!

For ω-complete posets there is an alternative characterization of the least fix-point of a *continuous* function as the least upper bound of a countable chain. The completion operation $\amalg$ used in the definition of the $\mathcal{OP}$ interpretation is not continuous. However, classically, the least fix-point of any monotone function f on a complete lattice lies in the transfinite chain [7]

$$d_{\alpha+1} = f(d_\alpha) \qquad \text{and} \qquad d_\lambda = \bigvee_{\alpha<\lambda} d_\alpha \ \text{ (for limit } \lambda)$$

(and dually for co-induction).

A set (or property) $\mathbf{P} \subseteq \mathcal{ONP}$ is said to be *admissible iff* (i) preserves lubs: $S \subseteq \mathbf{P} \implies \mathbf{P}(\bigvee S)$; and (ii) is downward closed: $a \leq b$ and $\mathbf{P}(b) \implies \mathbf{P}(a)$.

Theorem 7 (Scott Induction for Monotone Extensions of Endofunctions). *Let $f : L \to L$ be an endofunction (not necessarily a homomorphism) and $\mathbf{P}$ be an admissible property on a complete lattice L. If f preserves property $\mathbf{P}$, i.e. $\mathbf{P}(a) \implies \mathbf{P}(f\,a)$, then $\mathbf{P}$ holds for the least fix-point of its monotone extension, i.e. $\mathbf{P}\left(\mathit{lfp}(\lceil f \rceil)\right)$.*

With this proof principle and its dual one shows that the interpretation of DC with Mendler (co-)induction via realizability as orthogonal pairs satisfies the adequacy theorem (Theorem 4), and obtains the following result as a corollary.

Theorem 8 (Strong Normalization). *Every well-typed cut of the Dual Calculus with Mendler Induction is strongly normalizing.*

7 Concluding Remarks

We have investigated Classical Logic with Mendler Induction, presenting a Classical calculus with very general (co-)inductive types. Our work borrows from and generalizes systems based on Gentzen's *LK* under the Curry-Howard correspondence. Despite its generality, and as outlined by means of a realizability interpretation, our Dual Calculus with Mendler Induction is well-behaved in that its well-typed cuts are guaranteed to terminate. We expect—but have yet to fully confirm—that other models fit within our framework for interpreting Mendler Induction; our prime example is based on inflationary fix-points like those used in complexity theory [8] and which also apply to non-monotone functionals.

It is known that *LK*-based calculi can encode various other calculi [6, 19]. Our calculus supports map operations for all positive (co-)inductive types. In an extended version of the paper, we expect to use these to encode Kimura and Tatsuta's extension of the Dual Calculus with positive (co-)inductive types [13].

One avenue of research that remains unexplored is how one may extract proofs from within our system—in previous work, Berardi, et al. [4] showed how,

embracing the non-determinism of reduction inherent in the Symmetric Lambda-calculus (and also present in DC), one could express proof witnesses that behave like processes for a logic based on Peano arithmetic. A further direction would be to direct these investigations into the realm of linear logic, where the connection with processes may be more salient.

Acknowledgments Thanks to Anuj Dawar, Tim Griffin, Ohad Kammar, Andy Pitts, and the anonymous referees for their comments and suggestions.

References

1. Abel, A., Matthes, R., Uustalu, T.: Iteration and coiteration schemes for higher-order and nested datatypes. Theoretical Computer Science 333(1), 3–66 (2005)
2. Ahn, K.Y., Sheard, T.: A hierarchy of Mendler style recursion combinators: Taming inductive datatypes with negative occurrences. In: Proceedings of the 16th ACM SIGPLAN International Conference on Functional Programming. pp. 234–246. ICFP '11, ACM, New York, NY, USA (2011)
3. Barbanera, F., Berardi, S.: A symmetric lambda calculus for classical program extraction. Information and Computation 125(2), 103–117 (1996)
4. Barbanera, F., Berardi, S., Schivalocchi, M.: "Classical" programming-with-proofs in λ_{PA}^{Sym} : An analysis of non-confluence. In: Abadi, M., Ito, T. (eds.) Theoretical Aspects of Computer Software, Lecture Notes in Computer Science, vol. 1281, pp. 365–390. Springer Berlin Heidelberg (1997)
5. Crolard, T.: A formulae-as-types interpretation of subtractive logic. Journal of Logic and Computation 14(4), 529–570 (2004)
6. Curien, P.L., Herbelin, H.: The duality of computation. In: Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming. pp. 233–243. ICFP '00, ACM, New York, NY, USA (2000)
7. Davey, B.A., Priestley, H.A.: Introduction to Lattices and Order. Cambridge University Press (2002)
8. Dawar, A., Gurevich, Y.: Fixed point logics. Bulletin of Symbolic Logic 8(01), 65–88 (2002)
9. Dougherty, D., Ghilezan, S., Lescanne, P., Likavec, S.: Strong normalization of the dual classical sequent calculus. In: Sutcliffe, G., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning, Lecture Notes in Computer Science, vol. 3835, pp. 169–183. Springer Berlin Heidelberg (2005)
10. Gentzen, G.: Investigations into logical deduction. American Philosophical Quarterly 1(4), 288–306 (1964)
11. Harper, B., Lillibridge, M.: ML with callcc is unsound. Post to TYPES mailing list (1991)
12. Hur, C.K., Neis, G., Dreyer, D., Vafeiadis, V.: The power of parameterization in coinductive proof. In: Proceedings of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 193–206. POPL '13, ACM, New York, NY, USA (2013)
13. Kimura, D., Tatsuta, M.: Dual calculus with inductive and coinductive types. In: Treinen, R. (ed.) Rewriting Techniques and Applications, Lecture Notes in Computer Science, vol. 5595, pp. 224–238. Springer Berlin Heidelberg (2009)

14. Matthes, R.: Extensions of System F by Iteration and Primitive Recursion on Monotone Inductive Types. Ph.D. thesis, Ludwig-Maximilians Universität (May 1998)
15. Mendler, N.: Inductive types and type constraints in the second-order lambda calculus. Annals of Pure and Applied Logic 51(1), 159–172 (1991)
16. Parigot, M.: Strong normalization of second order symmetric λ-calculus. In: Kapoor, S., Prasad, S. (eds.) FST TCS 2000: Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, vol. 1974, pp. 442–453. Springer Berlin Heidelberg (2000)
17. Tzevelekos, N.: Investigations on the Dual Calculus. Theoretical Computer Science 360(1), 289–326 (2006)
18. Uustalu, T., Vene, V.: Mendler-style inductive types, categorically. Nord. J. Comput. 6(3), 343 (1999)
19. Wadler, P.: Call-by-value is dual to call-by-name. In: Proceedings of the Eighth ACM SIGPLAN International Conference on Functional Programming. pp. 189–201. ICFP '03, ACM, New York, NY, USA (2003)