Least and greatest fixed points in intuitionistic natural deduction ⋆

Tarmo Uustalu [a,1,2] and Varmo Vene [b,1]

[a] *Institute of Cybernetics at Tallinn Technical University, Akadeemia tee 21, EE-12618 Tallinn, Estonia*

[b] *Institute of Computer Science, University of Tartu, J. Liivi 2, EE-50409 Tartu, Estonia*

Received 1 December 1998; revised 1 December 1999; accepted 21 March 2000

Abstract

This paper is a comparative study of a number of (intensional-semantically distinct) least and greatest fixed point operators that natural-deduction proof systems for intuitionistic logics can be extended with in a proof-theoretically defendable way. Eight pairs of such operators are analysed. The exposition is centered around a cube-shaped classification where each node stands for an axiomatization of one pair of operators as logical constants by intended proof and reduction rules and each arc for a proof- and reduction-preserving encoding of one pair in terms of another. The three dimensions of the cube reflect three orthogonal binary options: conventional-style vs. Mendler-style, basic ("[co]iterative") vs. enhanced ("primitive-[co]recursive"), simple vs. course-of-value [co]induction. Some of the axiomatizations and encodings are well-known; others, however, are novel; the classification into a cube is also new. The differences between the least fixed point operators considered are illustrated on the example of the corresponding natural number types.

Key words: least and greatest fixed points, natural deduction, (co)inductive types, typed lambda calculi, coding styles, schemes of (total) (co)recursion

---

⋆ A version of a talk presented at the Mathematical Society of Japan Meeting on Theories for Types and Proofs, TTP-Tokyo'97 (Tokyo, Japan, 8–19 Sept. 1997).

[1] Partially supported by the Estonian Science Foundation under grant No. 2976.

[2] On leave to Department of Informatics, University of Minho, Campus de Gualtar, P-4710-057, Braga, Portugal.

1 Introduction

This paper is a comparative study of a number of least and greatest fixed point operators, or inductive and coinductive definition operators, that natural-deduction (N.D.) proof systems for intuitionistic logics (typed lambda calculi with product and sum types) can be extended with as logical constants (type-language constants), either by an axiomatization by intended proof and reduction rules ("implicit definition") or by a proof- and reduction-preserving encoding in terms of some logical constants already present ("explicit definition"). One of the reasons why such logical or type-language constants are interesting lies in their useful programming interpretation: inductive types behave as data types, their introductions as data constructors and eliminations as recursors; coinductive types may be viewed as codata types, their introductions as corecursors and eliminations as codata destructors. In the literature, a fairly large number of axiomatizations and encodings of both particular [co]inductively defined types and general [co]inductive definition operators can be found, see e.g., [1,14,19,20,24,25,15,7]. The paper grew out of a wish to better understand their individual properties and their relations to each other.

The contribution of the paper consists in a coordinated analysis of eight intensional-semantically distinct pairs of [co]inductive definition operators, arranged into a cube-shaped taxonomy, which resulted from an attempt to fit the various known axiomatizations and encodings into a single picture and to find fillers for the holes. Each node of the cube stands for an axiomatization by proof and reduction rules of one pair of logical constants and each arc for a proof- and reduction-preserving encoding of one pair in terms of another. Some axiomatizations and encodings rely on the presence in the system of certain other logical constants (the standard propositional connectives, 2nd-order quantifiers, or a "retractive" recursive definition operator σ). The three dimensions of the cube reflect three orthogonal binary choices: conventional-style vs. Mendler-style, basic ("[co]iterative") vs. enhanced ("primitive-[co]recursive"), simple vs. course-of-value [co]induction.

The cube looks as follows:

$$
\begin{tikzcd}
 & \mu^{\shortparallel\star}, \nu^{\shortparallel\star} \arrow[rr] & & \mathrm{M}^{\shortparallel\star}, \mathrm{N}^{\shortparallel\star} \\
\mu^{\shortparallel}, \nu^{\shortparallel} \arrow[rr] \arrow[ur] & & \mathrm{M}^{\shortparallel}, \mathrm{N}^{\shortparallel} \arrow[ur] & \\
 & \mu^{\star}, \nu^{\star} \arrow[rr] \arrow[uu] & & \mathrm{M}^{\star}, \mathrm{N}^{\star} \arrow[uu] \\
\mu, \nu \arrow[rr] \arrow[ur] \arrow[uu] & & \mathrm{M}, \mathrm{N} \arrow[ur] \arrow[uu] &
\end{tikzcd}
$$

μ and ν (with optional superscripts) are conventional-style inductive and coinductive definition operators; M and N (with optional superscripts) are Mendler-style operators. The superscript '$^{\shortparallel}$' marks the "enhanced" feature, the superscript '*' indicates the "course-of-value" feature.

The distinctions between basic and enhanced, simple and course-of-value [co]induction are distinctions between essentially different forms of [co]induction, with different associating schemes of (total) [co]recursion. Basic [co]induction gives [co]iteration, enhanced [co]induction gives (full) primitive [co]recursion. All axiomatizations and encodings we have found in the literature deal with simple forms of [co]induction. The axiomatizations and encodings for course-of-value [co]induction in this paper are ours, we think.

The difference between conventional- and Mendler-style [co]induction (named after Mendler [19,20]) is more technical and harder to spell out informally, but not shallow. A conventional-style [co]inductive definition operator applies to a proposition-function only if it is positive; the associating reduction rule refers then to a proof of its monotonicity (all positive proposition-functions are monotonic wrt the preorder of inclusion). Mendler-style operators apply to any proposition-functions. The axiomatizations of enhanced and course-of-value conventional-style operators rely on the presence in the system of other logical constants, those of Mendler-style operators do not. Thus, in more than one sense, Mendler-style operators are more uniform than conventional-style operators; resorting to programming jargon, one might for instance want to say that the Mendler-style operators are generic, whereas the conventional-style ones are only polytypic. These uniformity features have a price though: the proof rules of the Mendler-style operators involve implicit ("external") 2nd-order quantification at the level of premisses.

Throughout the paper, the semantics that we keep in mind is intensional, so we only consider β-reduction, not $\beta\eta$-conversion.

Some remarks are in order regarding the technical machinery that we use. By natural deduction, we mean a proof system style where instead of axioms involving implications and universal quantifications we systematically prefer to have proof rules involving hypothetical and schematic judgements ("externalized" implications and universal quantifications), in sharp contrast to the Hilbert style of proof systems. For us therefore, natural deduction is really the "extended" natural deduction of Schroeder-Heister [30,31]: we allow proof rules to be of order higher than two: not only may conclusions have premisses and these have premisses in their turn, but even the latter may be hypothetical. This choice makes axiomatizations of different logical constants very compact, but on the expense of certain added complexity in their encodings in terms of other logical constants.

In order to compactify the notation and to get around the technicalities related to α-conversion and substitution, we use a simple meta-syntax, a higher-oder abstract syntax derived from logical frameworks such as de Bruijn's AUT-PI and AUT-QE [5], Martin-Löf's system of arities [22, chapter 3], and Harper, Honsell, and Plotkin's LF [12]. $(x_1, \ldots, x_n)s$ denotes the schematization of s wrt. $x_1, \ldots, x_n$. $s(t_1, \ldots, t_n)$ denotes the instantiation of s with $t_1, \ldots, t_n$. Schematization and instantiation are stipulated to satisfy the following rules: $((x_1, \ldots, x_n)s)(t_1, \ldots, t_n) \equiv s[t_1/x_1, \ldots, t_n/x_n]$ and, if $x_1, \ldots, x_n$ are not free in s, then $(x_1, \ldots, x_n)(s(x_1, \ldots, x_n)) \equiv s$. ($\equiv$ denotes syntactic identity.)

We have made an effort to make the paper self-contained; for the omitted details, we refer to Uustalu [35]. A preliminary report of the present work appeared as [37]. We also refer to Matthes [17], an in-depth study of extensions of system F with constructors of basic and enhanced conventional- and Mendler-style inductive types, which in regard to the clarification of the relationship between the conventional- and Mendler-style induction builds partly upon our work.

The paper is organized as follows. In section 2, we lay down our starting point: it is given by systems that we denote $\mathcal{NI}$ and $\mathcal{NI}^2$, the N.D. proof systems for 1st- and 2nd-order intuitionistic propositional logics, optionally extended with a "retractive" recursive definition operator σ. Then, in section 3, we first present the basic [co]induction operators, both in conventional and Mendler-style and then continue with their encodings in terms of the 2nd-order quantifiers and each other. In sections 4 and 5, we describe enhanced [co]induction and course-of-value [co]induction operators respectively and their encodings via the operators of the basic kind. In section 6, we give a survey of related work on inductive and coinductive types. Finally, in section 7, we conclude and mention some directions for future work.

2 Preliminaries

In principle, the [co]inductive definition operators described below can be added to the N.D. proof system of any intuitionistic propositional logic. (They also admit a straightforward generalization for predicate logics.) The most natural base system for such extensions however is $\mathcal{NI}$, the standard N.D. proof system for (full) 1st-order intuitionistic propositional logic. The logical constants of $\mathcal{NI}$ are $\wedge$ (conjuction), $\vee$ (disjunction), $\top$ (verum), $\perp$ (falsum), and $\rightarrow$ (implication). These propositional connectives are axiomatized by the proof and reduction rules listed in Figure 1. (To save space, the reduction rules are given not for proofs, but for (untyped) term codes of proofs; the reduction rules for proofs are easy to recover. The reduction relation on terms satisfies subject reduction.)

$$\frac{c_L : A_L \quad c_R : A_R}{\langle c_L, c_R\rangle : A_L \wedge A_R}\wedge\mathsf{I} \qquad \frac{c : A_L \wedge A_R}{\mathsf{fst}(c) : A_L}\wedge\mathsf{E_L} \quad \frac{c : A_L \wedge A_R}{\mathsf{snd}(c) : A_R}\wedge\mathsf{E_R}$$

$$\mathsf{fst}(\langle c_L, c_R\rangle) \quad \rhd^\beta \quad c_L \qquad\qquad \mathsf{snd}(\langle c_L, c_R\rangle) \quad \rhd^\beta \quad c_R$$

$$\frac{c : A_L}{\mathsf{inl}(c) : A_L \vee A_R}\vee\mathsf{I_L} \quad \frac{c : A_R}{\mathsf{inr}(c) : A_L \vee A_R}\vee\mathsf{I_R} \qquad \frac{c : A_L \vee A_R \quad \dfrac{\xi : A_L}{e_L(\xi) : C} \quad \dfrac{\xi : A_R}{e_R(\xi) : C}}{\mathsf{case}(c, e_L, e_R) : C}\vee\mathsf{E}$$

$$\mathsf{case}(\mathsf{inl}(c), e_L, e_R) \quad \rhd^\beta \quad e_L(c) \qquad\qquad \mathsf{case}(\mathsf{inr}(c), e_L, e_R) \quad \rhd^\beta \quad e_R(c)$$

$$\frac{}{\langle\rangle : \top}\top\mathsf{I} \qquad \frac{c : \perp}{\mathsf{case}(c) : C}\perp\mathsf{E}$$

$$\frac{\dfrac{\xi : B}{c(\xi) : A}}{\lambda(c) : B \rightarrow A}\rightarrow\mathsf{I} \qquad \frac{c : B \rightarrow A \quad e : B}{c \cdot e : A}\rightarrow\mathsf{E}$$

$$\lambda(c) \cdot e \quad \rhd^\beta \quad c(e)$$

Figure 1: Proof and reduction rules for standard propositional connectives.

Another important base system is $\mathcal{NI}^2$, the N.D. proof system for 2nd-order intuitionistic propositional logic. This system extends $\mathcal{NI}$ with $\forall^2$ and $\exists^2$, the standard 2nd-order quantifiers. The proof rules for $\forall^2$ and $\exists^2$ are presented in Figure 2.

In the encodings of enhanced [co]induction in terms of basic [co]induction, we shall need a logical constant σ, a "retractive" recursive definition operator. This is a proposition-valued operator on proposition-functions that are positive. The proof and reduction rules for σ appear in Figure 3. The introduction and elimination rules for σ behave as an embedding-retraction pair. The

$$\frac{\begin{array}{c} c : F(Y) \\ {\scriptstyle [Y]} \end{array}}{c : \forall^2(F)}\ \forall^2\mathsf{I} \qquad\qquad \frac{c : \forall^2(F)}{c : F(Q)}\ \forall^2\mathsf{E}(Q)$$

$$c \quad \equiv \quad c$$

$$\frac{c : F(Q)}{c : \exists^2(F)}\ \exists^2\mathsf{I}(Q) \qquad\qquad \frac{c : \exists^2(F) \qquad \begin{array}{c} \dfrac{\xi : F(Y)}{e(\xi) : R} \\ {\scriptstyle [Y]} \end{array}}{e(c) : R}\ \exists^2\mathsf{E}$$

$$e(c) \quad \equiv \quad e(c)$$

Figure 2: Proof and reduction rules for $\forall^2$, $\exists^2$.

extensions of $\mathcal{NI}$ and $\mathcal{NI}^2$ with σ will be denoted by $\mathcal{NI}(\sigma)$ and $\mathcal{NI}^2(\sigma)$. Of importance for us is the fact that $\mathcal{NI}^2(\sigma)$ is strongly normalizing (i.e., every proof of $\mathcal{NI}^2(\sigma)$ is strongly normalizing); consult Mendler [19,20] and Urzyczyn [34].

$$\frac{c : F(\sigma(F))}{\mathsf{i}(c) : \sigma(F)}\ \sigma\mathsf{I} \qquad\qquad \frac{c : \sigma(F)}{\mathsf{o}(c) : F(\sigma(F))}\ \sigma\mathsf{E}$$

$$\mathsf{o}(\mathsf{i}(c)) \quad \rhd^{\beta} \quad c$$

Figure 3: Proof and reduction rules for σ.

The syntactic concepts of positivity and negativity of proposition-functions are system-dependent. For any particular system, these concepts are defined by mutual structural induction on proposition-functions definable in this system. In $\mathcal{NI}$ and its extensions considered in this paper, a proposition-function $(X)F$ is defined to be positive [negative] if every occurrence of X in F appears within an even [odd] number of antecedents of implications. Also for any particular system and by a similar induction, explicit definitions can be given for the derivable proof rules M^- and M^+ establishing that positive [negative] proposition-functions are monotonic [antimonotonic] wrt. the preorder of proposition inclusion. These proof rules appear in Figure 4.

$$\frac{c : F(Q') \qquad \dfrac{\zeta : Q'}{d(\zeta) : Q''}}{\mathsf{map}^+_F(c, d) : F(Q'')}\ \mathsf{M}^+ \qquad\qquad \frac{c : F(Q') \qquad \dfrac{\zeta : Q''}{d(\zeta) : Q'}}{\mathsf{map}^-_F(c, d) : F(Q'')}\ \mathsf{M}^-$$

$$F \text{ positive} \qquad\qquad\qquad F \text{ negative}$$

Figure 4: Derivable proof rules M^- and M^+.

As an example, we shall consider the proposition-function N defined by setting

$$\mathsf{N}(R) \quad \equiv \quad \top \vee R$$

N is obviously positive. The corresponding monotonicity witness $\mathsf{map}^+_{\mathsf{N}}$ is defined as follows:

$$\mathsf{map}^+_{\mathsf{N}}(c,d) \quad\equiv\quad \mathsf{case}(c,(\xi)\mathsf{inl}(\xi),(\xi)\mathsf{inr}(d(\xi)))$$

3 Basic [co]induction

The logical constants from the two lower front nodes of the cube provide the most fundamental forms of [co]inductive definition of propositions, viz. the *basic* (in other words, *"[co]iterative"*) forms of conventional- and Mendler-style [co]inductive definition. μ and ν are operators of conventional-style induction and coinduction and apply to positive proposition-functions only; $\textsc{m}$ and $\textsc{n}$ are their Mendler-style counterparts applicable without restrictions to any proposition-functions. Their proof and reduction rules are given in Figures 5 and 6. The proof rules for $\textsc{m}$ and $\textsc{n}$ are more complex than those for μ and ν, but their reduction rules, in compensation, are simpler and more uniform: their right-hand sides do not refer to the M^+ proof rule.

$$\frac{c : F(\mu(F))}{\mathsf{wrap}_F(c) : \mu(F)}\ \mu\mathsf{I} \qquad\qquad \frac{c : \mu(F) \quad \dfrac{\gamma : F(R)}{e(\gamma) : R}}{\mathsf{cata}_F(c,e) : R}\ \mu\mathsf{E}$$

$$\mathsf{cata}_F(\mathsf{wrap}_F(c),e) \quad \rhd^\beta \quad e(\mathsf{map}^+_F(c,(\zeta)\mathsf{cata}_F(\zeta,e)))$$

$$\frac{c : R \quad \dfrac{\gamma : R}{e(\gamma) : F(R)}}{\mathsf{ana}_F(c,e) : \nu(F)}\ \nu\mathsf{I} \qquad\qquad \frac{c : \nu(F)}{\mathsf{open}_F(c) : F(\nu(F))}\ \nu\mathsf{E}$$

$$\mathsf{open}_F(\mathsf{ana}_F(c,e)) \quad \rhd^\beta \quad \mathsf{map}^+_F(e(c),(\zeta)\mathsf{ana}_F(\zeta,e))$$

Figure 5: Proof and reduction rules for μ and ν.

$$\frac{c : F(Q) \quad \dfrac{\zeta : Q}{d(\zeta) : \textsc{m}(F)}}{\mathsf{mapwrap}(c,d) : \textsc{m}(F)}\ \textsc{m}\mathsf{I}(Q) \qquad\qquad \frac{c : \textsc{m}(F) \quad \underset{[Y]}{\dfrac{\gamma : F(Y) \quad \dfrac{\zeta : Y}{\delta(\zeta) : R}}{e(\gamma,\delta) : R}}}{\mathsf{iter}(c,e) : R}\ \textsc{m}\mathsf{E}$$

$$\mathsf{iter}(\mathsf{mapwrap}(c,d),e)) \quad \rhd^\beta \quad e(c,(\zeta)\mathsf{iter}(d(\zeta),e))$$

$$\frac{c : R \quad \underset{[Y]}{\dfrac{\gamma : R \quad \dfrac{\zeta : R}{\delta(\zeta) : Y}}{e(\gamma,\delta) : F(Y)}}}{\mathsf{coit}(c,e) : \textsc{n}(F)}\ \textsc{n}\mathsf{I} \qquad\qquad \frac{c : \textsc{n}(F) \quad \dfrac{\zeta : \textsc{n}(F)}{d(\zeta) : Q}}{\mathsf{mapopen}(c,d) : F(Q)}\ \textsc{n}\mathsf{E}(Q)$$

$$\mathsf{mapopen}(\mathsf{coit}(c,e),d) \quad \rhd^\beta \quad e(c,(\zeta)d(\mathsf{coit}(\zeta,e)))$$

Figure 6: Proof and reduction rules for $\textsc{m}$ and $\textsc{n}$.

From the algebraic semantics point of view, μF is a least prefixed point of F wrt. the inclusion preorder of propositions: it is both itself a prefixed point of F (by the I-rule) and a lower bound of the set of all prefixed points of F (by the E-rule). (Recall that R is said to be a prefixed point of F, if $F(R)$ is less than R.) νF, dually, is a greatest postfixed point of F.[3] Since a least [greatest] prefixed [postfixed] point of a monotonic function is also its least [greatest] fixed point, μF and νF are also least and greatest fixed points of F.

In a similar fashion, $\mathrm{M}F$ can be thought of as a least robustly prefixed point of F: it is both itself a robustly prefixed point of F and a lower bound of all robustly prefixed points of F. Here, R is considered to be a robustly prefixed point of F, if not only is $F(R)$ less than R, but $F(Y)$ is less than R for all Y's less than R. But $\mathrm{M}F$ is also a least (ordinary) prefixed point of a function F^{e} [$F^{\mathsf{e}}(R) \equiv \exists^2((Y)(Y \to R) \wedge F(Y))$] sending any R to a supremum of the set of all $F(Y)$'s such that Y is less than R. F^{e} (which is always positive) appears to be a least monotonic majorant of F wrt. the pointwise "lifting" of the inclusion preorder of propositions to a preorder of proposition-functions. If F is monotonic, then F and F^{e} are equivalent (pointwise). The dualization is obvious: $\mathrm{N}F$ is a greatest robustly postfixed point of F and a greatest (ordinary) postfixed point of a function F^{a} [$F^{\mathsf{a}}(R) \equiv \forall^2((Y)(R \to Y) \to F(Y))$] sending any R to an infimum of the set of all $F(Y)$'s such that Y is greater than R.

Under the programming interpretation, μF is a data type, with wrap_F a data constructor and cata_F an iterator, and νF is a codata type, with ana_F a coiterator and open_F a codata destructor, in the most standard sense. $\mathrm{M}F$, with $\mathsf{mapwrap}$ and iter, and $\mathrm{N}F$, with coit and $\mathsf{mapopen}$, are Mendler-style versions of these things. This is best explained on an example.

The type of standard natural numbers Nat, with zero and succ the constant zero and the successor function and $\mathsf{natcata}$ the iterator, is normally axiomatized as follows:

$$\mathsf{zero} : \mathsf{Nat} \qquad \frac{c : \mathsf{Nat}}{\mathsf{succ}(c) : \mathsf{Nat}} \qquad \frac{c : \mathsf{Nat} \quad e_z : R \quad \dfrac{\gamma : R}{e_s(\gamma) : R}}{\mathsf{natcata}(c, e_z, e_s) : R}$$

$$\begin{array}{rcl} \mathsf{natcata}(\mathsf{zero}, e_z, e_s) & \rhd^\beta & e_z \\ \mathsf{natcata}(\mathsf{succ}(c), e_z, e_s) & \rhd^\beta & e_s(\mathsf{natcata}(c, e_z, e_s)) \end{array}$$

These typing and reduction rules are essentially nothing else than those for

[3] Note here that, in a preorder (also in a Heyting algebra), it may turn out that all monotonic functions have least [greatest] prefixed [postfixed] points; hence allowing μ and ν to apply to any positive F should not lead to inconsistencies (the encodability of μ, ν in terms of $\forall^2$, $\exists^2$ demonstrates that this is the case indeed).

conventional basic induction with N as the underlying proposition-function. Indeed, making the following definitions ensures the required typing and reduction properties:

$$\begin{array}{rcl}
\mathsf{Nat} & \equiv & \mu(\mathsf{N}) \\
\mathsf{zero} & \equiv & \mathsf{wrap}_{\mathsf{N}}(\mathsf{inl}(\langle\rangle)) \\
\mathsf{succ}(c) & \equiv & \mathsf{wrap}_{\mathsf{N}}(\mathsf{inr}(c)) \\
\mathsf{natcata}(c, e_z, e_s) & \equiv & \mathsf{cata}_{\mathsf{N}}(c, (\gamma)\mathsf{case}(\gamma, (\xi)e_z, (\xi)e_s(\xi)))
\end{array}$$

This suggests a similar specialization of Mendler-style basic induction for N by the following definitions:

$$\begin{array}{rcl}
\textsc{Nat} & \equiv & \textsc{m}(\mathsf{N}) \\
\mathsf{mapzero}(d) & \equiv & \mathsf{mapwrap}(\mathsf{inl}(\langle\rangle), d) \\
\mathsf{mapsucc}(c, d) & \equiv & \mathsf{mapwrap}(\mathsf{inr}(c), d) \\
\mathsf{natiter}(c, e_z, e_s) & \equiv & \mathsf{iter}(c, (\gamma, \delta)\mathsf{case}(\gamma, (\xi)e_z(\delta), (\xi)e_s(\xi, \delta)))
\end{array}$$

The type $\textsc{Nat}$ of Mendler-style natural numbers, with $\mathsf{mapzero}$, $\mathsf{mapsucc}$ and $\mathsf{natiter}$ the Mendler-style constant zero, successor function, and iterator, obeys the following typing and reduction rules.

$$\frac{\dfrac{\zeta : Q}{d(\zeta) : \textsc{Nat}}}{\mathsf{mapzero}(d) : \textsc{Nat}} \qquad \frac{c : Q \quad \dfrac{\zeta : Q}{d(\zeta) : \textsc{Nat}}}{\mathsf{mapsucc}(c, d) : \textsc{Nat}} \qquad \frac{c : \textsc{Nat} \quad \underset{[Y]}{\dfrac{\dfrac{\zeta : Y}{\delta(\zeta) : R}}{e_z(\delta) : R}} \quad \underset{[Y]}{\dfrac{\gamma : Y \quad \dfrac{\zeta : Y}{\delta(\zeta) : R}}{e_s(\gamma, \delta) : R}}}{\mathsf{natiter}(c, e_z, e_s) : R}$$

$$\begin{array}{rcl}
\mathsf{natiter}(\mathsf{mapzero}(d), e_z, e_s) & \rhd^{\beta} & e_z((\zeta)\mathsf{natiter}(d(\zeta), e_z, e_s)) \\
\mathsf{natiter}(\mathsf{mapsucc}(c, d), e_z, e_s) & \rhd^{\beta} & e_s(c, (\zeta)\mathsf{natiter}(d(\zeta), e_z, e_s))
\end{array}$$

Here, it may be helpful to think of Q as some chosen type of representations for naturals and d as a method for converting representations of this type to naturals. A natural, hence, is constructed from nothing or a representation (for its predecessor), together with a method for converting representations to naturals. Using $\textsc{Nat}$ as Q, the standard constructors of naturals are definable as follows:

$$\begin{array}{rcl}
\mathsf{zero} & \equiv & \mathsf{mapzero}((\zeta)\zeta) \\
\mathsf{succ}(c) & \equiv & \mathsf{mapsucc}(c, (\zeta)\zeta)
\end{array}$$

$\mathsf{natcata}$ and $\mathsf{natiter}$ are iterators. Iteration is a very simple form of total recursion: the result of an iteration on a given natural is only dependent of the result on the predecessor. If the "straightforward" definition of a function follows some more complex form of recursion, then definitions by iteration can get clumsy. The factorial of a given natural, for instance, depends not only on the factorial of its predecessor, but also on the predecessor itself. An iterative definition of the factorial has to define both the factorial and the identity function "in parallel" and then project the factorial component out.

$$\mathsf{fact}(c) \equiv \mathsf{fst}(\mathsf{natcata}(c, \left\langle \begin{array}{l} \mathsf{one}, \\ \mathsf{zero} \end{array} \right\rangle, (\gamma) \left\langle \begin{array}{l} \mathsf{mult}(\mathsf{fst}(\gamma), \mathsf{succ}(\mathsf{snd}(\gamma))), \\ \mathsf{succ}(\mathsf{snd}(\gamma)) \end{array} \right\rangle))$$

$$\mathsf{fact}(c) \equiv \mathsf{fst}(\mathsf{natiter}(c, (\delta) \left\langle \begin{array}{l} \mathsf{one}, \\ \mathsf{zero} \end{array} \right\rangle, (\gamma, \delta) \left\langle \begin{array}{l} \mathsf{mult}(\mathsf{fst}(\delta(\gamma)), \mathsf{succ}(\mathsf{snd}(\delta(\gamma)))), \\ \mathsf{succ}(\mathsf{snd}(\delta(\gamma))) \end{array} \right\rangle))$$

Exactly the same trick of "tupling" is also needed to program the Fibonacci function: the Fibonacci of a given natural number depends not only on the Fibonacci of its predecessor, but also on the Fibonacci of its pre-predecessor. An iterative definition of Fibonacci has to define both Fibonacci and the "one-step-behind Fibonacci" "in parallel".

$$\mathsf{fibo}(c) \equiv \mathsf{fst}(\mathsf{natcata}(c, \left\langle \begin{array}{l} \mathsf{zero}, \\ \mathsf{inl}(\langle\rangle) \end{array} \right\rangle, (\gamma) \left\langle \begin{array}{l} \mathsf{case}\left(\mathsf{snd}(\gamma), \begin{array}{l} (\xi')\mathsf{one}, \\ (\xi')\mathsf{add}(\mathsf{fst}(\gamma), \xi') \end{array} \right), \\ \mathsf{inr}(\mathsf{fst}(\gamma)) \end{array} \right\rangle))$$

$$\mathsf{fibo}(c) \equiv \mathsf{fst}(\mathsf{natiter}(c, (\delta) \left\langle \begin{array}{l} \mathsf{zero}, \\ \mathsf{inl}(\langle\rangle) \end{array} \right\rangle, (\gamma, \delta) \left\langle \begin{array}{l} \mathsf{case}\left(\mathsf{snd}(\delta(\gamma)), \begin{array}{l} (\xi')\mathsf{one}, \\ (\xi')\mathsf{add}(\mathsf{fst}(\delta(\gamma)), \xi') \end{array} \right), \\ \mathsf{inr}(\mathsf{fst}(\delta(\gamma))) \end{array} \right\rangle))$$

These examples show how other forms of recursion can be captured by iteration using "tupling". Such modelling is not without drawbacks, however. First, it is more transparent to define a function using its "native" form of recursion. Second, the intensional behavior of iterative definitions is not always satisfactory. It is well known, for instance, that the predecessor function can be programmed using iteration, but the programs take linear time to compute (and only work as desirable on numerals, i.e., closed natural number terms).

$$\mathsf{pred}(c) \equiv \mathsf{cata}_{\mathsf{N}}(c, (\gamma)\mathsf{map}^{+}_{\mathsf{N}}(\gamma, (\zeta)\mathsf{wrap}_{\mathsf{N}}(\zeta)))$$

$$\mathsf{pred}(c) \equiv \mathsf{iter}(c, (\gamma, \delta)\mathsf{map}^{+}_{\mathsf{N}}(\gamma, (\zeta)\mathsf{mapwrap}(\delta(\zeta), (\zeta)\zeta)))$$

The more complex forms of induction considered in the following sections remedy these problems by offering more advanced forms of recursion.

Basic [co]induction vs. 2nd-order quantifiers

Both μ, ν and M, N can be encoded in terms of $\forall^2$, $\exists^2$ in a proof- and reduction-preserving manner.

Proposition 1 *The following is a proof- and reduction-preserving encoding of μ, ν in terms of $\forall^2$, $\exists^2$:*

$$\begin{array}{rcl} \mu^\sharp(F) & \equiv & \forall^2((X)(F(X)\to X)\to X) \\ \mathsf{wrap}^\sharp_F(c) & \equiv & \lambda((\chi)\chi\cdot\mathsf{map}^+_F(c,(\zeta)\zeta\cdot\chi)) \\ \mathsf{cata}^\sharp_F(c,e) & \equiv & c\cdot\lambda((\gamma)e(\gamma)) \\ \nu^\sharp(F) & \equiv & \exists^2((X)(X\to F(X))\wedge X) \\ \mathsf{ana}^\sharp_F(c,e) & \equiv & \langle\lambda((\gamma)e(\gamma)),c\rangle \\ \mathsf{open}^\sharp_F(c) & \equiv & \mathsf{map}^+_F(\mathsf{fst}(c)\cdot\mathsf{snd}(c),(\zeta)\langle\mathsf{fst}(c),\zeta\rangle) \end{array}$$

This encoding is a proof theory recapitulation of the Knaster–Tarski fixed point theorem [33] stating that an infimum [supremum] of the set of all prefixed [postfixed] points of a monotonic function is its least [greatest] prefixed [postfixed] point. In its general form, the encoding seems to be a piece of folklore. For the special case of "polynomial" proposition-functions (such as N), essentially the same encoding was first given by Böhm and Berarducci [1] and Leivant [14]. For naturals, our encoding specializes to the following:

$$\begin{array}{rcl} \mathsf{Nat}^\sharp & \equiv & \forall^2((X)(\top\vee X\to X)\to X) \\ \mathsf{zero}^\sharp & \equiv & \lambda((\chi)\chi\cdot\mathsf{inl}(\langle\rangle)) \\ \mathsf{succ}^\sharp(c) & \equiv & \lambda((\chi)\chi\cdot\mathsf{inr}(c\cdot\chi)) \\ \mathsf{natcata}^\sharp(c,e_z,e_s) & \equiv & c\cdot\lambda((\gamma)\mathsf{case}(\gamma,(\xi)e_z,(\xi)e_s(\xi))) \end{array}$$

(In Böhm and Berarducci's encoding, $\mathsf{Nat}^\sharp \equiv \forall^2((X)X \to (X \to X) \to X)$, $\mathsf{zero}^\sharp \equiv \lambda((\chi_z)\lambda((\chi_s)\chi_z))$, $\mathsf{succ}^\sharp(c) \equiv \lambda((\chi_z)\lambda((\chi_s)\chi_s{\cdot}(c{\cdot}\chi_z{\cdot}\chi_s)))$, $\mathsf{natcata}^\sharp(c,e_z,e_s) \equiv c\cdot e_z\cdot e_s$.)

Proposition 2 *The following is a proof- and reduction-preserving encoding of M, N in terms of $\forall^2$, $\exists^2$:*

$$\mathrm{M}^\sharp(F) \quad\equiv\quad \forall^2((X)\forall^2((Y)(Y\to X)\to(F(Y)\to X))\to X)$$

$$
\begin{aligned}
\mathsf{mapwrap}^\sharp(c,d) &\equiv \lambda((\chi)\chi\cdot\lambda((\zeta)d(\zeta)\cdot\chi)\cdot c)\\
\mathsf{iter}^\sharp(c,e) &\equiv c\cdot\lambda((\delta)\lambda((\gamma)e(\gamma,(\zeta)\delta\cdot\zeta)))\\
\mathrm{N}^\sharp(F) &\equiv \exists^2((X)\forall^2((Y)(X\to Y)\to(X\to F(Y)))\wedge X)\\
\mathsf{coit}^\sharp(c,e) &\equiv \langle\lambda((\delta)\lambda((\gamma)e(\gamma,(\zeta)\delta\cdot\zeta))),c\rangle\\
\mathsf{mapopen}^\sharp(c,d) &\equiv \mathsf{fst}(c)\cdot\lambda((\zeta)d(\langle\mathsf{fst}(c),\zeta\rangle))\cdot\mathsf{snd}(c)
\end{aligned}
$$

This encoding builds on the following robust analog of the Knaster-Tarski fixed point theorem: an infimum [supremum] of the set of all robustly prefixed [postfixed] points of any function (monotonic or not) is its least [greatest] robustly prefixed [postfixed] point.

Corollary 3 *$\mathcal{NI}^2(\sigma)$ (and also its any fragment, including $\mathcal{NI}$) extended with operators μ,ν or M,N is strongly normalizing and confluent.*

Mendler-style vs. conventional [co]induction

It is also possible to encode μ, ν in terms of M, N and vice versa. For the encoding in the latter direction, $\forall^2$, $\exists^2$ have to be available.

Proposition 4 *The following is a proof- and reduction-preserving encoding of μ, ν in terms of M, N:*

$$
\begin{aligned}
\mu^\sharp(F) &\equiv \mathrm{M}(F)\\
\mathsf{wrap}^\sharp_F(c) &\equiv \mathsf{mapwrap}(c,(\zeta)\zeta)\\
\mathsf{cata}^\sharp_F(c,e) &\equiv \mathsf{iter}(c,(\gamma,\delta)e(\mathsf{map}^+_F(\gamma,\delta))\\
\nu^\sharp(F) &\equiv \mathrm{N}(F)\\
\mathsf{ana}^\sharp_F(c,e) &\equiv \mathsf{coit}(c,(\gamma,\delta)\mathsf{map}^+_F(e(\gamma),\delta))\\
\mathsf{open}^\sharp_F(c) &\equiv \mathsf{mapopen}(c,(\zeta)\zeta)
\end{aligned}
$$

Proposition 5 *The following is a proof- and reduction-preserving encoding of M, N in terms of μ, ν in the presence of $\exists^2$, $\forall^2$:*

$$
\begin{aligned}
F^{\mathbf{e}}(R) &\equiv \exists^2((Y)(Y\to R)\wedge F(Y))\\
\mathrm{M}^\sharp(F) &\equiv \mu(F^{\mathbf{e}})\\
\mathsf{mapwrap}^\sharp(c,d) &\equiv \mathsf{wrap}_{F^{\mathbf{e}}}(\langle\lambda(d),c\rangle)\\
\mathsf{iter}^\sharp(c,e) &\equiv \mathsf{cata}_{F^{\mathbf{e}}}(c,(\gamma)e(\mathsf{snd}(\gamma),(\zeta)\mathsf{fst}(\gamma)\cdot\zeta))\\
F^{\mathbf{a}}(R) &\equiv \forall^2((Y)(R\to Y)\to F(Y))\\
\mathrm{N}^\sharp(F) &\equiv \nu(F^{\mathbf{a}})\\
\mathsf{coit}^\sharp(c,e) &\equiv \mathsf{ana}_{F^{\mathbf{a}}}(c,(\gamma)\lambda((\delta)e(\gamma,(\zeta)\delta\cdot\zeta)))
\end{aligned}
$$

$$\mathsf{mapopen}^{\sharp}(c, d) \quad \equiv \quad \mathsf{open}_{F^{\mathbf{a}}}(c) \cdot \lambda(d)$$

The encoding of M, N in terms of μ, ν is a proof-theoretic version of the observation that a least [greatest] prefixed [postfixed] point of $F^{\mathbf{e}}$ [$F^{\mathbf{a}}$] is a least [greatest] robustly prefixed [postfixed] point of F.

4 Enhanced [co]induction

The logical constants from the two upper front nodes of the cube capture the *enhanced* (in other words, *"primitive-[co]recursive"*) forms of conventional- and Mendler-style [co]inductive definition. $\mu^{\shortparallel}$ and $\nu^{\shortparallel}$ are operators of enhanced induction and coinduction; $\mathrm{M}^{\shortparallel}$ and $\mathrm{N}^{\shortparallel}$ are their Mendler-style counterparts. Their proof and reduction rules are given in Figures 7 and 8. Adding $\mu^{\shortparallel}$, $\nu^{\shortparallel}$ to a proof system presupposes the presence of $\wedge$, $\vee$; there is no corresponding restriction governing the addition of $\mathrm{M}^{\shortparallel}$, $\mathrm{N}^{\shortparallel}$.

$$\frac{c : F(\mu^{\shortparallel}(F) \wedge \mu^{\shortparallel}(F))}{\mathsf{wrap}^{\shortparallel}_{F}(c) : \mu^{\shortparallel}(F)}\ \mu^{\shortparallel}\mathsf{I} \qquad \frac{c : \mu^{\shortparallel}(F) \quad \dfrac{\gamma : F(R \wedge \mu^{\shortparallel}(F))}{e(\gamma) : R}}{\mathsf{para}_{F}(c, e) : R}\ \mu^{\shortparallel}\mathsf{E}$$

$$\mathsf{para}_{F}(\mathsf{wrap}^{\shortparallel}_{F}(c), e) \quad \rhd^{\beta} \quad e(\mathsf{map}^{+}_{F}(c, (\zeta)\left\langle \begin{matrix} \mathsf{para}_{F}(\mathsf{fst}(\zeta), e), \\ \mathsf{snd}(\zeta) \end{matrix} \right\rangle))$$

$$\frac{c : R \quad \dfrac{\gamma : R}{e(\gamma) : F(R \vee \nu^{\shortparallel}(F))}}{\mathsf{apo}_{F}(c, e) : \nu^{\shortparallel}(F)}\ \nu^{\shortparallel}\mathsf{I} \qquad \frac{c : \nu^{\shortparallel}(F)}{\mathsf{open}^{\shortparallel}_{F}(c) : F(\nu^{\shortparallel}(F) \vee \nu^{\shortparallel}(F))}\ \nu^{\shortparallel}\mathsf{E}$$

$$\mathsf{open}^{\shortparallel}_{F}(\mathsf{apo}_{F}(c, e)) \quad \rhd^{\beta} \quad \mathsf{map}^{+}_{F}(e(c), (\zeta)\mathsf{case}\left(\zeta, \begin{matrix} (\xi)\mathsf{inl}(\mathsf{apo}_{F}(\xi, e)), \\ (\xi)\mathsf{inr}(\xi) \end{matrix} \right))$$

Figure 7: Proof and reduction rules for $\mu^{\shortparallel}$ and $\nu^{\shortparallel}$.

From the algebraic semantics point-of-view, $\mu^{\shortparallel}F$ is a least "recursive" prefixed point of a given (necessarily monotonic) F, i.e., a least element of the set of all R's such that $F(R \wedge \mu^{\shortparallel}F)$ is less than R (note the recurrent occurrence of $\mu^{\shortparallel}F$ here!). $\nu^{\shortparallel}F$ is a greatest "recursive" postfixed point of F.

$\mathrm{M}^{\shortparallel}F$ is a least "recursive" robustly prefixed point of a given F, ie., a least element of the set of all R's such that $F(Y)$ is less than R for all Y's less than not only R but also $\mathrm{M}^{\shortparallel}F$ (note again the circularity!). $\mathrm{N}^{\shortparallel}F$, dually, is a greatest "recursive" robustly postfixed point of F.

For programming, $\mu^{\shortparallel}F$ is a "recursive" data type, with $\mathsf{wrap}^{\shortparallel}_{F}$ a "recursive" data constructor and para_{F} a primitive recursor, and $\nu^{\shortparallel}F$ is a "recursive" codata type, with apo_{F} a primitive corecursor and $\mathsf{open}^{\shortparallel}_{F}$ a "recursive" codata

$$\frac{c : F(Q) \quad \dfrac{\zeta : Q}{d(\zeta) : \mathrm{M}''(F)} \quad \dfrac{\zeta : Q}{i(\zeta) : \mathrm{M}''(F)}}{\mathsf{mapwrap}''(c,d,i) : \mathrm{M}''(F)}\ \mathsf{M''I}(Q) \qquad \frac{c : \mathrm{M}''(F) \quad \overset{[Y]}{\dfrac{\gamma : F(Y) \quad \dfrac{\zeta : Y}{\delta(\zeta) : R} \quad \dfrac{\zeta : Y}{\iota(\zeta) : \mathrm{M}''(F)}}{e(\gamma,\delta,\iota) : R}}}{\mathsf{rec}(c,e) : R}\ \mathsf{M''E}$$

$$\mathsf{rec}(\mathsf{mapwrap}''(c,d,i),e) \quad \rhd^{\beta} \quad e(c,(\zeta)\mathsf{rec}(d(\zeta),e),(\zeta)i(\zeta))$$

$$\frac{c : R \quad \overset{[Y]}{\dfrac{\gamma : R \quad \dfrac{\zeta : R}{\delta(\zeta) : Y} \quad \dfrac{\zeta : \mathrm{N}''(F)}{\iota(\zeta) : Y}}{e(\gamma,\delta,\iota) : F(Y)}}}{\mathsf{cor}(c,e) : \mathrm{N}''(F)}\ \mathsf{N''I} \qquad \frac{c : \mathrm{N}''(F) \quad \dfrac{\zeta : \mathrm{N}''(F)}{d(\zeta) : Q} \quad \dfrac{\zeta : \mathrm{N}''(F)}{i(\zeta) : Q}}{\mathsf{mapopen}''(c,d,i) : F(Q)}\ \mathsf{N''E}(Q)$$

$$\mathsf{mapopen}''(\mathsf{cor}(c,e),d,i) \quad \rhd^{\beta} \quad e(c,(\zeta)d(\mathsf{cor}(\zeta,e)),(\zeta)i(\zeta))$$

Figure 8: Proof and reduction rules for M'' and N''.

destructor. $\mathrm{M}''F$, with $\mathsf{mapwrap}''$ and rec, and $\mathrm{N}''F$, with cor and $\mathsf{mapopen}''$, are their Mendler-style equivalents.

Returning to our running example of naturals, specializing enhanced induction for N yields the type Nat'' of "recursive" natural numbers, with zero'', succ'' and $\mathsf{natpara}$ the "recursive" constant zero, "recursive" successor function and primitive recursor.

$$\begin{aligned}
\mathsf{Nat}'' &\equiv \mu''(\mathsf{N}) \\
\mathsf{zero}'' &\equiv \mathsf{wrap}''_{\mathsf{N}}(\mathsf{inl}(\langle\rangle)) \\
\mathsf{succ}''(c) &\equiv \mathsf{wrap}''_{\mathsf{N}}(\mathsf{inr}(c)) \\
\mathsf{natpara}(c,e_z,e_s) &\equiv \mathsf{para}_{\mathsf{N}}(c,(\gamma)\mathsf{case}(\gamma,(\xi)e_z,(\xi)e_s(\xi)))
\end{aligned}$$

The typing and reduction rules for Nat'' are the following:

$$\mathsf{zero}'' : \mathsf{Nat}'' \qquad \frac{c : \mathsf{Nat}'' \wedge \mathsf{Nat}''}{\mathsf{succ}''(c) : \mathsf{Nat}''} \qquad \frac{c : \mathsf{Nat}'' \quad e_z : R \quad \dfrac{\gamma : R \wedge \mathsf{Nat}''}{e_s(\gamma) : R}}{\mathsf{natpara}(c,e_z,e_s) : R}$$

$$\begin{aligned}
\mathsf{natpara}(\mathsf{zero}'',e_z,e_s) \quad &\rhd^{\beta} \quad e_z \\
\mathsf{natpara}(\mathsf{succ}''(c),e_z,e_s) \quad &\rhd^{\beta} \quad e_s(\langle \mathsf{natpara}(\mathsf{fst}(c),e_z,e_s),\mathsf{snd}(c)\rangle)
\end{aligned}$$

Note that a non-zero "recursive" natural is constructed from a pair of naturals. In the reduction rule, the first of them is used as the argument for the recurrent applications of the function being defined, while the second one is used directly. In principle, the two naturals can be unrelated, but the normal usage of the construction is that the second natural is equal to the first (the predecessor), so the standard successor function is recovered by duplicating its argument.

$$
\begin{aligned}
\mathsf{zero} &\equiv \mathsf{zero}^{\shortparallel} \\
\mathsf{succ}(c) &\equiv \mathsf{succ}^{\shortparallel}(\langle c, c\rangle)
\end{aligned}
$$

The type $\mathrm{NAT}^{\shortparallel}$ of “recursive” Mendler-style naturals is defined as follows:

$$
\begin{aligned}
\mathrm{NAT}^{\shortparallel} &\equiv \mathrm{M}^{\shortparallel}(\mathsf{N}) \\
\mathsf{mapzero}^{\shortparallel}(d, i) &\equiv \mathsf{mapwrap}^{\shortparallel}(\mathsf{inl}(\langle\rangle), d, i) \\
\mathsf{mapsucc}^{\shortparallel}(c, d, i) &\equiv \mathsf{mapwrap}^{\shortparallel}(\mathsf{inr}(c), d, i) \\
\mathsf{natrec}(c, e_z, e_s) &\equiv \mathsf{rec}(c, (\gamma, \delta, \iota)\mathsf{case}(\gamma, (\xi)e_z(\delta, \iota), (\xi)e_s(\xi, \delta, \iota)))
\end{aligned}
$$

$\mathrm{NAT}^{\shortparallel}$ obeys the following typing and reduction rules:

$$
\dfrac{\dfrac{\zeta : Q}{d(\zeta) : \mathrm{NAT}^{\shortparallel}} \quad \dfrac{\zeta : Q}{i(\zeta) : \mathrm{NAT}^{\shortparallel}}}{\mathsf{mapzero}^{\shortparallel}(d, i) : \mathrm{NAT}^{\shortparallel}}
\qquad
\dfrac{c : Q \quad \dfrac{\zeta : Q}{d(\zeta) : \mathrm{NAT}^{\shortparallel}} \quad \dfrac{\zeta : Q}{i(\zeta) : \mathrm{NAT}^{\shortparallel}}}{\mathsf{mapsucc}^{\shortparallel}(c, d, i) : \mathrm{NAT}^{\shortparallel}}
$$

$$
\dfrac{c : \mathrm{NAT}^{\shortparallel} \quad \underset{[Y]}{\dfrac{\dfrac{\zeta : Y}{\delta(\zeta) : R} \quad \dfrac{\zeta : Y}{\iota(\zeta) : \mathrm{NAT}^{\shortparallel}}}{e_z(\delta, \iota) : R}} \quad \underset{[Y]}{\dfrac{\gamma : Y \quad \dfrac{\zeta : Y}{\delta(\zeta) : R} \quad \dfrac{\zeta : Y}{\iota(\zeta) : \mathrm{NAT}^{\shortparallel}}}{e_s(\gamma, \delta, \iota) : R}}}{\mathsf{natrec}(c, e_z, e_s) : R}
$$

$$
\begin{aligned}
\mathsf{natrec}(\mathsf{mapzero}^{\shortparallel}(d, i), e_z, e_s) &\rhd^{\beta} e_z((\zeta)d(\mathsf{natrec}(\zeta, e_z, e_s)), i) \\
\mathsf{natrec}(\mathsf{mapsucc}^{\shortparallel}(c, d, i), e_z, e_s) &\rhd^{\beta} e_s(c, (\zeta)d(\mathsf{natrec}(\zeta, e_z, e_s)), i)
\end{aligned}
$$

A non-zero “recursive” Mendler-style natural is constructed from a representation (for the predecessor), a method for converting representations to naturals and another function from representations to naturals. In the normal usage of the construction, the second method is also a conversion method. Choosing $\mathrm{NAT}^{\shortparallel}$ as the type of representations, the standard constructors are obtained as follows:

$$
\begin{aligned}
\mathsf{zero} &\equiv \mathsf{mapzero}^{\shortparallel}((\zeta)\zeta, (\zeta)\zeta) \\
\mathsf{succ}(c) &\equiv \mathsf{mapsucc}^{\shortparallel}(c, (\zeta)\zeta, (\zeta)\zeta)
\end{aligned}
$$

On “recursive” naturals constructed using the standard constructors, natpara and natrec capture standard primitive recursion. The factorial function, for instance, can be programmed as follows:

$$
\mathsf{fact}(c) \equiv \mathsf{natpara}(c, \mathsf{one}, (\gamma)\mathsf{mult}(\mathsf{fst}(\gamma), \mathsf{succ}(\mathsf{snd}(\gamma)))))
$$

$$
\mathsf{fact}(c) \equiv \mathsf{natrec}(c, (\delta, \iota)\mathsf{one}, (\gamma, \delta, \iota)\mathsf{mult}(\delta(\gamma), \mathsf{succ}(\iota(\gamma))))
$$

A degenerate application of primitive recursion, which only uses the "direct-access" predecessors of non-zero naturals, gives a fast (constant time) program for the predecessor function:

$$\mathsf{pred}(c) \equiv \mathsf{natpara}(c, \mathsf{inl}(\langle\rangle), (\gamma)\mathsf{inr}(\mathsf{snd}(\gamma)))$$

$$\mathsf{pred}(c) \equiv \mathsf{natrec}(c, (\delta, \iota)\mathsf{inl}(\langle\rangle), (\gamma, \delta, \iota)\mathsf{inr}(\iota(\gamma)))$$

Enhanced vs. basic [co]induction

Both μ, ν and M, N can be encoded in terms of μ'', ν'' and M'', N''. The converse is also true, but only if the retractive recursive definition operator σ is available.

Proposition 6 *The following is a proof- and reduction-preserving encoding of μ, ν in terms of μ'', ν'':*

$$\begin{aligned}
\mu^\sharp(F) &\equiv \mu''(F)\\
\mathsf{wrap}^\sharp_F(c) &\equiv \mathsf{wrap}''_F(\mathsf{map}^+_F(c, (\zeta)\langle\zeta, \zeta\rangle))\\
\mathsf{cata}^\sharp_F(c, e) &\equiv \mathsf{para}_F(c, (\gamma)e(\mathsf{map}^+_F(\gamma, \mathsf{fst})))\\
\nu^\sharp(F) &\equiv \nu''(F)\\
\mathsf{ana}^\sharp_F(c, e) &\equiv \mathsf{apo}_F(c, (\gamma)\mathsf{map}^+_F(e(\gamma), \mathsf{inl}))\\
\mathsf{open}^\sharp_F(c) &\equiv \mathsf{map}^+_F(\mathsf{open}''_F(c), (\zeta)\mathsf{case}(\zeta, (\xi)\xi, (\xi)\xi))
\end{aligned}$$

Proposition 7 *The following is a proof- and reduction-preserving encoding of M, N in terms of M'', N'':*

$$\begin{aligned}
\mathrm{M}^\sharp(F) &\equiv \mathrm{M}''(F)\\
\mathsf{mapwrap}^\sharp(c, d) &\equiv \mathsf{mapwrap}''(c, d, d)\\
\mathsf{iter}^\sharp(c, e) &\equiv \mathsf{rec}(c, (\gamma, \delta, \iota)e(\gamma, \delta))\\
\mathrm{N}^\sharp(F) &\equiv \mathrm{N}''(F)\\
\mathsf{coit}^\sharp(c, e) &\equiv \mathsf{cor}(c, (\gamma, \delta, \iota)e(\gamma, \delta))\\
\mathsf{mapopen}^\sharp(c, d) &\equiv \mathsf{mapopen}''(c, d, d)
\end{aligned}$$

Proposition 8 *The following is a proof- and reduction-preserving encoding of μ'', ν'' in terms of μ, ν in the presence of σ:*

$$\begin{aligned}
\mu''^\sharp(F) &\equiv \sigma((Z)\mu((X)F(X \wedge Z)))\\
F''(R) &\equiv F(R \wedge \mu''^\sharp(F))\\
\mathsf{wrap}''^\sharp_F(c) &\equiv \mathsf{i}(\mathsf{wrap}_{F''}(\mathsf{map}^+_{F''}(c, (\zeta)\mathsf{o}(\zeta))))
\end{aligned}$$

$$
\begin{array}{rcl}
\mathsf{para}^{\sharp}_{F}(c,e) & \equiv & \mathsf{cata}_{F''}(\mathsf{o}(c),e) \\
\nu''^{\sharp}(F) & \equiv & \sigma((Z)\nu((X)F(X \vee Z))) \\
F''(R) & \equiv & F(R \vee \nu''^{\sharp}(F)) \\
\mathsf{apo}^{\sharp}_{F}(c,e) & \equiv & \mathsf{i}(\mathsf{ana}_{F''}(c,e)) \\
\mathsf{open}''^{\sharp}_{F} & \equiv & \mathsf{map}^{+}_{F''}(\mathsf{open}_{F''}(\mathsf{o}(c)),(\zeta)\mathsf{i}(\zeta))
\end{array}
$$

Proposition 9 *The following is a proof- and reduction-preserving encoding of* M'', N'' *in terms of* M, N *in the presence of* σ*:*

$$
\begin{array}{rcl}
\mathrm{M}''^{\sharp}(F) & \equiv & \sigma((Z)\mathrm{M}((X)(X \to Z) \wedge F(X)) \\
F''(R) & \equiv & (R \to \mathrm{M}''^{\sharp}(F)) \wedge F(R) \\
\mathsf{mapwrap}''^{\sharp}(c,d,i) & \equiv & \mathsf{i}(\mathsf{mapwrap}(\langle\lambda(i),c\rangle,(\zeta)\mathsf{o}(d(\zeta)))) \\
\mathsf{rec}^{\sharp}(c,e) & \equiv & \mathsf{iter}(\mathsf{o}(c),(\gamma,\delta)e(\mathsf{snd}(\gamma),(\zeta)\delta(\zeta),(\zeta)\mathsf{fst}(\gamma)\cdot\zeta)) \\
\mathrm{N}''^{\sharp}(F) & \equiv & \sigma((Z)\mathrm{N}((X)(Z \to X) \to F(X)) \\
F''(R) & \equiv & (\mathrm{N}''^{\sharp}(F) \to R) \to F(R) \\
\mathsf{cor}^{\sharp}(c,e) & \equiv & \mathsf{i}(\mathsf{coit}(c,(\gamma,\delta)\lambda((\iota)e(\gamma,(\zeta)\delta(\zeta),(\zeta)\iota\cdot\zeta)))) \\
\mathsf{mapopen}''^{\sharp}(c,d,i) & \equiv & \mathsf{mapopen}(\mathsf{o}(c),(\zeta)d(\mathsf{i}((\zeta)))\cdot\lambda(i)
\end{array}
$$

In the last two encodings, we would really like to define $\mu''^{\sharp}(F) \equiv \mu((X)F(X \wedge \mu''^{\sharp}(F)))$ and $\mathrm{M}''^{\sharp}(F) \equiv \mathrm{M}((X)(X \to \mathrm{M}''^{\sharp}(F)) \wedge F(X))$, but cannot (because of the circularity). Resorting to σ is a way to overcome this obstacle. From the result in [32], it follows that using σ is a necessity, one cannot possibly do without it.

The first of these encodings is implicit in [25] and [15]. It also appears in [7]. The second seems to be new.

Corollary 10 $\mathcal{NI}^2(\sigma)$ *(and also its any fragment, including* $\mathcal{NI}$*) extended with operators* μ'', ν'' *or* $\mathrm{M}'', \mathrm{N}''$ *is strongly normalizing and confluent.*

5 Course-of-value [co]induction

The logical constants from the two lower rear nodes of the cube capture the *course-of-value* forms of conventional- and Mendler-style [co]inductive definition. $\mu^\star$ and $\nu^\star$ are operators of course-of-value induction and coinduction; $\mathrm{M}^\star$ and $\mathrm{N}^\star$ are their Mendler-style counterparts. Their proof and reduction rules are given in Figures 9 and 10. Adding $\mu^\star$, $\nu^\star$ to a proof system presupposes the presence of $\wedge$, ν, $\vee$, μ ; there is no corresponding restriction governing the addition of $\mathrm{M}^\star$, $\mathrm{N}^\star$.

$$(R \triangle F)(P) \quad\equiv\quad R \wedge F(P)$$

$$\frac{c : F(\nu(\mu^\star(F) \triangle F))}{\mathsf{wrap}^\star_F(c) : \mu^\star(F)}\ \mu^\star\mathsf{I} \qquad\qquad \frac{c : \mu^\star(F) \qquad \dfrac{\gamma : F(\nu(R \triangle F))}{e(\gamma) : R}}{\mathsf{cvcata}_F(c, e) : R}\ \mu^\star\mathsf{E}$$

$$\begin{array}{ll}\mathsf{cvcata}_F(\mathsf{wrap}^\star_F(c), e) & \\ \quad \triangleright^\beta & e(\mathsf{map}^+_F(c, (\zeta)\mathsf{ana}_{\_\triangle F}(\zeta, (\gamma)\left\langle \begin{array}{l}\mathsf{cvcata}_F(\mathsf{fst}(\mathsf{open}_{\_\triangle F}(\gamma)), e),\\ \mathsf{snd}(\mathsf{open}_{\_\triangle F}(\gamma))\end{array}\right\rangle)))\end{array}$$

$$(R \triangledown F)(P) \quad\equiv\quad R \vee F(P)$$

$$\frac{c : R \qquad \dfrac{\gamma : R}{e(\gamma) : F(\mu(R \triangledown F))}}{\mathsf{cvana}_F(c, e) : \nu^\star(F)}\ \nu^\star\mathsf{I} \qquad\qquad \frac{c : \nu^\star(F)}{\mathsf{open}^\star_F(c) : F(\mu(\nu^\star(F) \triangledown F))}\ \nu\mathsf{E}$$

$$\begin{array}{ll}\mathsf{open}^\star_F(\mathsf{cvana}_F(c, e)) & \\ \quad \triangleright^\beta & \mathsf{map}^+_F(e(c), (\zeta)\mathsf{cata}_{\_\triangledown F}(\zeta, (\gamma)\mathsf{wrap}_{\_\triangledown F}(\mathsf{case}\left(\gamma, \begin{array}{l}(\xi)\mathsf{inl}(\mathsf{cvana}_F(\xi, e)),\\ (\xi)\mathsf{inr}(\xi)\end{array}\right)))\end{array}$$

Figure 9: Proof and reduction rules for $\mu^\star$ and $\nu^\star$.

$$\frac{c : F(Q) \qquad \dfrac{\zeta : Q}{d(\zeta) : \mathrm{M}^\star(F)} \qquad \dfrac{\zeta : Q}{k(\zeta) : F(Q)}}{\mathsf{mapwrap}^\star(c, d, k) : \mathrm{M}^\star(F)}\ \mathrm{M}^\star\mathsf{I}(Q) \qquad \frac{c : \mathrm{M}^\star(F) \qquad \dfrac{\gamma : F(Y) \qquad \dfrac{\zeta : Y}{\delta(\zeta) : R} \qquad \dfrac{\zeta : Y}{\kappa(\zeta) : F(Y)}}{e(\gamma, \delta, \kappa) : R}\,[Y]}{\mathsf{cviter}(c, e) : R}\ \mathrm{M}^\star\mathsf{E}$$

$$\mathsf{cviter}(\mathsf{mapwrap}^\star(c, d, k), e) \quad \triangleright^\beta \quad e(c, (\zeta)\mathsf{cviter}(d(\zeta), e), (\zeta)k(\zeta))$$

$$\frac{c : R \qquad \dfrac{\gamma : R \qquad \dfrac{\zeta : R}{\delta(\zeta) : Y} \qquad \dfrac{\zeta : F(Y)}{\kappa(\zeta) : Y}}{e(\gamma, \delta, \kappa) : F(Y)}\,[Y]}{\mathsf{cvcoit}(c, e) : \mathrm{N}^\star(F)}\ \mathrm{N}^\star\mathsf{I} \qquad \frac{c : \mathrm{N}^\star(F) \qquad \dfrac{\zeta : \mathrm{N}^\star(F)}{d(\zeta) : Q} \qquad \dfrac{\zeta : F(Q)}{k(\zeta) : Q}}{\mathsf{mapopen}^\star(c, d, k) : F(Q)}\ \mathrm{N}^\star\mathsf{E}(Q)$$

$$\mathsf{mapopen}^\star(\mathsf{cvcoit}(c, e), d, k) \quad \triangleright^\beta \quad e(c, (\zeta)d(\mathsf{cvcoit}(\zeta, e)), (\zeta)k(\zeta))$$

Figure 10: Proof and reduction rules for $\mathrm{M}^\star$ and $\mathrm{N}^\star$.

From the algebraic semantics point-of-view, $\mu^\star F$ is a least course-of-value prefixed point of a given (necessarily monotonic) F, i.e., a least element of the set of all R's such that $F(\nu(Z)R \wedge F(Z))$ is less than R. $\nu^\star F$ is a greatest course-of-value postfixed point of F.

$\mathrm{M}^{\shortparallel}F$ is a least course-of-value robustly prefixed point of a given F, i.e., a least element of the set of all R's such that $F(Y)$ is less than R for all Y's less than not only R but also $F(Y)$. $\text{N}^\star F$, dually, is a greatest course-of-value robustly postfixed point of F.

For programming, $\mu^\star F$ is a course-of-value data type, with $\mathsf{wrap}^\star_F$ a course-of-value data constructor and cvcata_F a course-of-value iterator, and $\nu^\star F$ is a course-of-value codata type, with cvana_F a course-of-value iterator and $\mathsf{open}^\star_F$ a course-of-value codata destructor. $\text{M}^\star F$, with $\mathsf{mapwrap}^\star$ and cviter, and $\text{N}^\star F$, with cvcoit and $\mathsf{mapopen}^\star$, are their Mendler-style equivalents.

Specializing course-of-value induction for N yields the type $\mathsf{Nat}^\star$ of "course-of-value" natural numbers, with $\mathsf{zero}^\star$, $\mathsf{succ}^\star$ and $\mathsf{natcviter}$ the "course-of-value" versions of constant zero, successor function and iterator respectively.

$$
\begin{array}{rcl}
\mathsf{Nat}^\star & \equiv & \mu^\star(\mathsf{N}) \\
\mathsf{zero}^\star & \equiv & \mathsf{wrap}^\star_{\mathsf{N}}(\mathsf{inl}(\langle\rangle)) \\
\mathsf{succ}^\star(c) & \equiv & \mathsf{wrap}^\star_{\mathsf{N}}(\mathsf{inr}(c)) \\
\mathsf{natcvcata}(c, e_z, e_s) & \equiv & \mathsf{cvcata}_{\mathsf{N}}(c, (\gamma)\mathsf{case}(\gamma, (\xi)e_z, (\xi)e_s(\xi)))
\end{array}
$$

The specialized typing and reduction rules for these constants are the following:

$$
\mathsf{zero}^\star : \mathsf{Nat}^\star \qquad \frac{c : \nu((Z)\mathsf{Nat}^\star \wedge \mathsf{N}(Z))}{\mathsf{succ}^\star(c) : \mathsf{Nat}^\star} \qquad \frac{c : \mathsf{Nat}^\star \quad e_z : R \quad \dfrac{\gamma : \nu((Z)R \wedge \mathsf{N}(Z))}{e_s(\gamma) : R}}{\mathsf{natcvcata}(c, e_z, e_s) : R}
$$

$$
\begin{array}{rcl}
\mathsf{natcvcata}(\mathsf{zero}^\star, e_z, e_s) & \rhd^\beta & e_z \\
\mathsf{natcvcata}(\mathsf{succ}^\star(c), e_z, e_s) & \rhd^\beta & e_s(\mathsf{ana}(c, (\gamma)\left\langle \begin{array}{l} \mathsf{natcvcata}(\mathsf{fst}(\mathsf{open}(\gamma)), e_z, e_s), \\ \mathsf{snd}(\mathsf{open}(\gamma)) \end{array} \right\rangle))
\end{array}
$$

Similarly to the "recursive" case, non-zero "course-of-value" naturals are not constructed from a single preceding natural. The argument of the "course-of-value" successor function is a colist-like structure of naturals. The coiteration in the reduction rule applies the function being defined recurrently to every element of the colist. In principle, again, the naturals in the colist can be unrelated. The normal usage, however, is that the tail of the colist is the ancestral of its head (the predecessor of the natural being constructed). (By the ancestral of a natural, we mean the colist of all lesser naturals in the descending order.) The standard successor function for naturals is therefore easily recovered from the "course-of-value" successor function by first coiteratively applying the predecessor function to its argument.

$$
\begin{array}{rcl}
\mathsf{zero} & \equiv & \mathsf{zero}^\star \\
\mathsf{succ}(c) & \equiv & \mathsf{succ}^\star(\mathsf{ana}(c, (\gamma)\langle\gamma, \mathsf{pred}(\gamma)\rangle))
\end{array}
$$

The predecessor function, however, does not admit a very straightforward definition (this is a problem that vanishes in the case of course-of-value primitive recursion). But it is definable in terms of the ancestral function, which itself is definable by course-of-value iteration in the same way as the predecessor function is definable by simple iteration.

$$\begin{array}{rcl} \mathsf{pred}(c) & \equiv & \mathsf{map}^{+}_{\mathsf{N}}(\mathsf{pred}^{\star}(c), (\zeta)\mathsf{fst}(\mathsf{open}(\zeta))) \\ \mathsf{pred}^{\star}(c) & \equiv & \mathsf{cvcata}_{\mathsf{N}}\left(c, (\gamma)\mathsf{map}^{+}_{\mathsf{N}}\left(\gamma, (\zeta)\mathsf{ana}\left(\zeta, (\gamma')\left\langle \begin{array}{l} \mathsf{wrap}^{\star}_{\mathsf{N}}(\mathsf{fst}(\mathsf{open}(\gamma'))) \\ \mathsf{snd}(\mathsf{open}(\gamma')) \end{array}\right\rangle\right)\right)\right) \end{array}$$

The specialization of course-of-value Mendler-style induction for N yields the type $\textsc{Nat}^{\star}$ of "course-of-value" Mendler-style naturals.

$$\begin{array}{rcl} \textsc{Nat}^{\star} & \equiv & \textsc{m}^{\star}(\mathsf{N}) \\ \mathsf{mapzero}^{\star}(d,k) & \equiv & \mathsf{mapwrap}^{\star}(\mathsf{inl}(\langle\rangle), d, k) \\ \mathsf{mapsucc}^{\star}(c,d,k) & \equiv & \mathsf{mapwrap}^{\star}(\mathsf{inr}(c), d, k) \\ \mathsf{natcviter}(c, e_z, e_s) & \equiv & \mathsf{cviter}(c, (\gamma,\delta,\kappa)\mathsf{case}(\gamma, (\xi)e_z(\delta,\kappa), (\xi)e_s(\xi,\delta,\kappa))) \end{array}$$

The derived typing and reduction rules for the above-defined constants are the following:

$$\frac{\dfrac{\zeta : Q}{d(\zeta) : \textsc{Nat}^{\star}} \quad \dfrac{\zeta : Q}{k(\zeta) : \mathsf{N}(Q)}}{\mathsf{mapzero}^{\star}(d,k) : \textsc{Nat}^{\star}} \qquad \frac{c : Q \quad \dfrac{\zeta : Q}{d(\zeta) : \textsc{Nat}^{\star}} \quad \dfrac{\zeta : Q}{k(\zeta) : \mathsf{N}(Q)}}{\mathsf{mapsucc}^{\star}(c,d,k) : \textsc{Nat}^{\star}}$$

$$\frac{c : \textsc{Nat}^{\star} \quad \underset{[Y]}{\dfrac{\dfrac{\zeta : Y}{\delta(\zeta) : R} \quad \dfrac{\zeta : Y}{\kappa(\zeta) : \mathsf{N}(Y)}}{e_z(\delta,\kappa) : R}} \quad \underset{[Y]}{\dfrac{\gamma : Y \quad \dfrac{\zeta : Y}{\delta(\zeta) : R} \quad \dfrac{\zeta : Y}{\kappa(\zeta) : \mathsf{N}(Y)}}{e_s(\gamma,\delta,\kappa) : R}}}{\mathsf{natcviter}(c, e_z, e_s) : R}$$

$$\begin{array}{rcl} \mathsf{natcviter}(\mathsf{mapzero}^{\star}(d,k), e_z, e_s) & \rhd^{\beta} & e_z((\zeta)d(\mathsf{natcviter}(\zeta, e_z, e_s)), k) \\ \mathsf{natcviter}(\mathsf{mapsucc}^{\star}(c,d,k), e_z, e_s) & \rhd^{\beta} & e_s(c, (\zeta)d(\mathsf{natcviter}(\zeta, e_z, e_s)), k) \end{array}$$

A non-zero "course-of-value" Mendler-style natural is constructed from three components. The first two are the same as in the case of simple Mendler-style naturals: a representation for a natural (the predecessor) and a method to convert representations to naturals. The additional third component gives a method for converting a representation (for some natural) into nothing or another representation (normally for the predecessor of this natural). So, using $\textsc{Nat}^{\star}$ as the type of representations, we obtain the standard constructors of naturals as follows:

$$\begin{array}{rcl} \mathsf{zero} & \equiv & \mathsf{mapzero}^\star((\zeta)\zeta, \mathsf{pred}) \\ \mathsf{succ}(c) & \equiv & \mathsf{mapsucc}^\star(c, (\zeta)\zeta, \mathsf{pred}) \end{array}$$

To define the predecessor function, we again need also the ancestral function.

$$\begin{array}{rcl} \mathsf{pred}(c) & \equiv & \mathsf{map}^+_{\mathsf{N}}(\mathsf{pred}^\star(c), (\zeta)\mathsf{fst}(\mathsf{open}(\zeta))) \\ \mathsf{pred}^\star(c) & \equiv & \mathsf{cviter}(c, (\gamma)\mathsf{map}^+_{\mathsf{N}}(\gamma, (\zeta)\mathsf{ana}(\zeta, \\ & & \qquad (\gamma')\left\langle \begin{array}{l} \mathsf{mapwrap}(\delta(\gamma'), (\zeta')\mathsf{fst}(\mathsf{open}(\zeta')), (\zeta')\mathsf{snd}(\mathsf{open}(\zeta'))), \\ \kappa(\gamma') \end{array} \right\rangle))) \end{array}$$

On “course-of-value” naturals constructed using the standard constructors, natcvcata and natcviter capture standard course-of-value iteration. The Fibonacci function, for instance, can be programmed using natcvcata as follows.

$$\mathsf{fibo}(c) \equiv \mathsf{natcvcata}(c, \mathsf{zero}, (\gamma)\mathsf{case}\left(\mathsf{snd}(\mathsf{open}(\gamma)), \begin{array}{l} (\xi')\mathsf{one} \\ (\xi')\mathsf{add}(\mathsf{fst}(\mathsf{open}(\gamma)), \mathsf{fst}(\mathsf{open}(\xi'))) \end{array}\right))$$

Using natcviter, the definition of the Fibonacci function becomes even more straightforward, as, instead of having to manipulate an intermediate colist of values that Fibonacci returns, we can “roll back” on inputs to it.

$$\mathsf{fibo}(c) \equiv \mathsf{natcviter}(c, (\delta, \kappa)\mathsf{zero}, (\gamma, \delta, \kappa)\mathsf{case}\left(\kappa(\gamma), \begin{array}{l} (\xi')\mathsf{one} \\ (\xi')\mathsf{add}(\delta(\gamma), \delta(\xi')) \end{array}\right))$$

Course-of-value vs. basic [co]induction

Encoding μ, ν and M, N in terms of $\mu^\star$, $\nu^\star$ and $\mathrm{M}^\star$, $\mathrm{N}^\star$ is very similar to encoding these constants in terms of μ'', ν'' and M'', N''. Also encoding in the opposite direction is analogous and, in fact, even simpler (as σ in not needed).

Proposition 11 *The following is a proof- and reduction-preserving encoding of μ, ν in terms of $\mu^\star$, $\nu^\star$:*

$$\begin{array}{rcl} F^{\mathbf{i}}(R) & \equiv & F(R) \vee R \\ \mu^\sharp(F) & \equiv & \mu^\star(F^{\mathbf{i}}) \end{array}$$

$$
\begin{array}{rcl}
\mathsf{wrap}^{\sharp}_{F}(c) & \equiv & \mathsf{wrap}^{\star}_{F^{\mathbf{i}}}(\mathsf{inl}(\mathsf{map}^{+}_{F}(c,(\zeta)\mathsf{ana}_{\_\triangle F^{\mathbf{i}}}(\zeta,(\gamma')\langle\gamma',\mathsf{inr}(\gamma')\rangle)))) \\
\mathsf{cata}^{\sharp}_{F}(c,e) & \equiv & \mathsf{cvcata}_{F}(c,(\gamma)\mathsf{case}\left(\gamma,\begin{array}{l}(\xi)e(\mathsf{map}^{+}_{F}(\xi,(\zeta)\mathsf{fst}(\mathsf{open}_{\_\triangle F^{\mathbf{i}}}(\zeta)))),\\ (\xi)\mathsf{fst}(\mathsf{open}_{\_\triangle F^{\mathbf{i}}}(\xi))\end{array}\right)) \\
F^{\mathbf{o}}(R) & \equiv & F(R)\wedge R \\
\nu^{\sharp}(F) & \equiv & \nu^{\star}(F^{\mathbf{o}}) \\
\mathsf{ana}^{\sharp}_{F}(c,e) & \equiv & \mathsf{cvana}_{F}(c,(\gamma)\left\langle\begin{array}{l}\mathsf{map}^{+}_{F}(e(\gamma),(\zeta)\mathsf{wrap}_{\_\triangledown F^{\mathbf{o}}}(\mathsf{inl}(\zeta))),\\ \mathsf{wrap}_{\_\triangledown F^{\mathbf{o}}}(\mathsf{inl}(\gamma))\end{array}\right\rangle) \\
\mathsf{open}^{\sharp}_{F}(c) & \equiv & \mathsf{map}^{+}_{F}(\mathsf{fst}(\mathsf{open}^{\star}_{F^{\mathbf{o}}}(c)),(\zeta)\mathsf{cata}_{\_\triangledown F^{\mathbf{o}}}(\zeta,(\gamma')\mathsf{case}(\gamma',(\xi)\xi,(\xi)\mathsf{snd}(\xi))))
\end{array}
$$

Proposition 12 *The following is a proof- and reduction-preserving encoding of* M, N *in terms of* $\mathrm{M}^{\star}$, $\mathrm{N}^{\star}$*:*

$$
\begin{array}{rcl}
F^{\mathbf{i}}(R) & \equiv & F(R)\vee R \\
\mathrm{M}^{\sharp}(F) & \equiv & \mathrm{M}^{\star}(F^{\mathbf{i}}) \\
\mathsf{mapwrap}^{\sharp}(c,d) & \equiv & \mathsf{mapwrap}^{\star}(\mathsf{inl}(c),d,\mathsf{inr}) \\
\mathsf{iter}^{\sharp}(c,e) & \equiv & \mathsf{cviter}(c,(\gamma,\delta,\kappa)\mathsf{case}(\gamma,(\xi)e(\xi,\delta),(\xi)\delta(\xi))) \\
F^{\mathbf{o}}(R) & \equiv & F(R)\wedge R \\
\mathrm{N}^{\sharp}(F) & \equiv & \mathrm{N}^{\star}(F^{\mathbf{o}}) \\
\mathsf{coit}^{\sharp}(c,e) & \equiv & \mathsf{cvcoit}(c,(\gamma,\delta,\kappa)\langle e(\gamma,\delta),\delta(\gamma)\rangle) \\
\mathsf{mapopen}^{\sharp}(c,d) & \equiv & \mathsf{fst}(\mathsf{mapopen}^{\star}(c,d,\mathsf{snd})
\end{array}
$$

Proposition 13 *The following is a proof- and reduction-preserving encoding of* $\mu^{\star}$, $\nu^{\star}$ *in terms of* μ, ν*:*

$$
\begin{array}{rcl}
F^{\star}(R) & \equiv & F(\nu(R\triangle F)) \\
\mu^{\star\sharp}(F) & \equiv & \mu(F^{\star}) \\
\mathsf{wrap}^{\star\sharp}_{F} & \equiv & \mathsf{wrap}_{F^{\star}}(c) \\
\mathsf{cvcata}^{\sharp}_{F}(c,e) & \equiv & \mathsf{cata}_{F^{\star}}(c,e) \\
F^{\star}(R) & \equiv & F(\mu(R\triangledown F)) \\
\nu^{\star\sharp}(F) & \equiv & \nu(F^{\star}) \\
\mathsf{cvana}^{\sharp}_{F}(c,e) & \equiv & \mathsf{ana}_{F^{\star}}(c,e) \\
\mathsf{open}^{\star\sharp}_{F} & \equiv & \mathsf{open}_{F^{\star}}(c)
\end{array}
$$

Proposition 14 *The following is a proof- and reduction-preserving encoding of* $\mathrm{M}^{\star}$, $\mathrm{N}^{\star}$ *in terms of* M, N*:*

$$
\begin{array}{rcl}
F^{\star}(R) & \equiv & (R\rightarrow F(R))\wedge F(R) \\
\mathrm{M}^{\star\sharp}(F) & \equiv & \mathrm{M}(F^{\star})
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{mapwrap}^{\star\sharp}(c,d,k) & \equiv & \mathsf{mapwrap}(\langle\lambda(k),c\rangle,(\zeta)d(\zeta)) \\
\mathsf{cviter}^{\sharp}(c,e) & \equiv & \mathsf{iter}(c,(\gamma,\delta)e(\mathsf{snd}(\gamma),(\zeta)\delta(\zeta),(\zeta)\mathsf{fst}(\gamma)\cdot\zeta)) \\
F^{\star}(R) & \equiv & (F(R)\to R)\to F(R) \\
\mathrm{N}^{\star\sharp}(F) & \equiv & \mathrm{N}(F^{\star}) \\
\mathsf{cvcoit}^{\sharp}(c,e) & \equiv & \mathsf{coit}(c,(\gamma,\delta)\lambda((\kappa)e(\gamma,(\zeta)\delta(\zeta),(\zeta)\kappa\cdot\zeta))) \\
\mathsf{mapopen}^{\star\sharp}(c,d,k) & \equiv & \mathsf{mapopen}(c,(\zeta)d(\zeta))\cdot\lambda(k)
\end{array}
$$

Corollary 15 *$\mathcal{NI}^2(\sigma)$ (and also its any fragment, including $\mathcal{NI}$) extended with operators $\mu^\star, \nu^\star$ or $\mathrm{M}^\star, \mathrm{N}^\star$ is strongly normalizing and confluent.*

6 Related work

The first author to extend an intuitionistic N.D. system with (basic conventional-style) inductively defined predicates uniformly by axiomatization was Martin-Löf, with his "theory of iterated inductive definitions" [21].

Böhm and Berarducci [1] and Leivant [14] were the first authors to describe how to encode "polynomial" (basic conventional-style) inductive types in 2nd-order simply typed lambda calculus (Girard and Reynold's system F; the N.D. proof system for the $\rightarrow,\forall^2$-fragment of 2nd-order intuitionistic propositional logic). This method is often referred to as *the* impredicative encoding of inductive types (keeping in mind only basic conventional-style induction). Mendler [19] described the extension by axiomatization of 2nd-order simply typed lambda calculus with enhanced inductive and coinductive types of his style. Mendler [20] discussed a similar system with basic Mendler-style inductive and coinductive types. Extensions of the N.D. proof systems for 2nd-order intuitionistic predicate logic with constructors of (basic) conventional- and Mendler-style inductive predicates were described in Leivant's [15], a paper on extracting programs in (extensions of) 2st-order simply typed lambda calculus from proofs in (extensions of) the N.D. proof system for 2nd-order intuitionistic predicate logic. Parigot's work [24,25] on realizability-based "programming with proofs" bears connection to both Leivant's and Mendler's works.

Greiner [10] and Howard [13, chapter 3] considered programming in an extension of 1st-order simply typed lambda calculus with axiomatized constructors of conventional-style (co)inductive types with (co)iteration and data destruction (codata construction). Both had their motivation in Hagino's category-theoretic work cited below and studied thus not barely β-reduction, but even $\beta\eta$-conversion, driven by definite semantic considerations. Howard implemented his system in a programming language Lemon. Geuvers [7] carried out a comparative study of basic vs. enhanced, conventional- vs. Mendler-style inductive and coinductive types in extensions of 2nd-order simply typed lambda calculus.

In the spirit of Leivant, Paulin-Mohring [26] extracted programs in Girard's F_ω from proofs in Coquand and Huet's CC (calculus of constructions). The milestone papers on inductive type families in extensions of CC and Luo's ECC (extended calculus of constructions, a combination of CC and Martin-Löf's type theory) are Pfenning and Paulin-Mohring [28], Coquand and Paulin-Mohring [4] and Ore [23]. Paulin-Mohring [27] formulated the calculus of inductive constructions, which extends CC with inductive type families with primitive recursion by axiomatization. The Coq proof development system developed at INRIA-Rocqencourt and ENS-Lyon is an implementation of this last system.

In category theory, (basic conventional-style) inductive and coinductive types are modelled by initial algebras and terminal coalgebras for covariant functors. Hagino [11] designed a typed functional language CPL based on distributive categories with initial algebras and terminal coalgebras for strong covariant functors. The implemented Charity language by Cockett et al. [3] is a similar programming language.

The "program calculation" community is rooted in the Bird-Meertens formalism or Squiggol [2], which, originally, was an equational theory of programming with the parametric data type of lists. Malcolm [16] made the community aware of Hagino's work, and studied program calculation based on bi-Cartesian closed categories with initial algebras and terminal coalgebras for ω-cocontinuous resp. ω-continuous covariant functors. Meertens [18] was the first author to give a treatment of primitive-recursion in this setting. Some classic references in the area are Fokkinga's [6] and Sheard and Fegaras' [29].

7 Conclusion and Future Work

In this paper, we studied least and greatest fixed point operators that intuitionistic N.D. systems can be extended with. We described eight pairs of such operators whose eliminations and introductions behave as recursors and corecursors of meaningful kinds.

We intend to continue this research with a study of the perspectives of the utility of intuitionistic N.D. systems with least and greatest fixed point operators in program construction from specifications; this concerns both specification methodology and computer assistance in synthesis. We have also started to study the relating categorical deduction systems (typed combinatory logics à la Curien), their utility in "program calculation" and the relevant categorical theory [38,36,39,40]. We also intend to find out the details of the apparent close relationship of enhanced course-of-value Mendler-style (co)recursion to Giménez' new formulation of guarded (co)recursion [9] (for systems with sub- and supertyping and quantification with upper and lower bounds; radically different from the older, very syntactical formulation of [8]).

Acknowledgements

We are thankful to our anonymous referees for a number of helpful comments and suggestions, especially in regards to matters of presentation. The proof figures and diagrams appearing in the paper were typeset using the `proof.sty` LaTeX2ε macro by Makoto Tatsuta and the XYpic generic TeX macro package by Kristoffer C. Rose, respectively.

References

[1] C. Böhm and A. Berarducci, Automatic synthesis of typed Λ-programs on term algebras, *Theoretical Computer Science* **39** (1985) 135–154.

[2] R. S. Bird, An introduction to the theory of lists, in: M. Broy, ed., *Logic of Programming and Calculi of Discrete Design*, *NATO ASI Series F*, Vol. 36 (Springer-Verlag, Berlin, 1987) 3–42.

[3] R. Cockett and T. Fukushima, About Charity, Yellow Series Report 92/480/18, Dept. of Computer Science, Univ. of Calgary (1992).

[4] T. Coquand and C. Paulin, Inductively defined types (preliminary version), in: P. Martin-Löf and G. Mints, eds., *Proceedings Int. Conf. on Computer Logic, COLOG'88 (Tallinn, USSR, Dec. 1988)*, *Lecture Notes in Computer Science*, Vol. 417 (Springer-Verlag, Berlin, 1990) 50–66.

[5] N. G. de Bruijn, A survey of the project AUTOMATH, in: J. P. Seldin and J. R. Hindley, eds., *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism* (Acabdemic Press, London, 1980) 579–606.

[6] M. M. Fokkinga, Law and order in algorithmics, Ph.D. Thesis, Dept. of Informatics, Univ. of Twente (1992).

[7] H. Geuvers, Inductive and coinductive types with iteration and recursion, in: B. Nordström, K. Pettersson, and G. Plotkin, eds., *Informal Proceedings Workshop on Types for Proofs and Programs (Båstad, Sweden, June 1992)*, Dept. of Computing Science, Chalmers Univ. of Technology and Göteborg Univ. (1992) 193–217. ftp://ftp.cs.chalmers.se/pub/cs-reports/baastad.92/proc.ps.Z.

[8] E. Giménez, Codifying guarded definitions with recursion schemes, in: P. Dybjer and B. Nordström, eds., *Selected Papers 2nd Int. Workshop on Types for*

Proofs and Programs, TYPES'94 (Båstad, Sweden, June 1994), *Lecture Notes in Computer Science*, Vol. 996 (Springer-Verlag, Berlin, 1995) 39–59.

[9] E. Giménez, Structural recursive definitions in type theory, in: K. G. Larsen, S. Skyum, and G. Winskel, eds., *Proceedings 25th Int. Coll. on Automata, Languages and Programming, ICALP'98 (Aalborg, Denmark, July 1998)*, *Lecture Notes in Computer Science*, Vol. 1443 (Springer-Verlag, Berlin, 1998) 397–408.

[10] J. Greiner, Programming with inductive and co-inductive types, Tech. Report CMU-CS-92-109, School of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, USA (1992).

[11] T. Hagino, A categorical programming language, Ph.D. Thesis CST-47-87, Lab. for Foundations of Computer Science, Dept. of Computer Science, Univ. of Edinburgh (1987).

[12] R. Harper, F. Honsell, and G. Plotkin, A framework for defining logics, *Journal of the ACM* **40** (1993) 143–184.

[13] B. Howard, Fixed points and extensionality in typed functional programming languages, Ph.D. Thesis, Tech. Report STAN-CS-92-1455, Computer Science Dept., Stanford Univ., CA. (1992).

[14] D. Leivant, Reasoning about functional programs and complexity classes associated with type disciplines, in: *Proceedings 24th Annual IEEE Symp. on Foundations of Computer Science, FOCS'83 (Tucson, AZ, USA, Nov. 1983)* (IEEE CS Press, Los Alamitos, CA, 1983) 460–469.

[15] D. Leivant, Contracting proofs to programs, in: P. Odifreddi, ed., *Logic and Computer Science*, *APIC Studies in Data Processing*, Vol. 31 (Academic Press, London, 1990) 279–327.

[16] G. Malcolm, Data structures and program transformation, *Science of Computer Programming* **14** (1990) 255–279.

[17] R. Matthes, Extensions of system F by iteration and primitive recursion on monotone inductive types, Ph.D. Thesis, Fachbereich Mathematik, Ludwig-Maximilians-Universität München (1998).

[18] L. Meertens, Paramorphisms, *Formal Aspects of Computing* **4** (1992) 413–424.

[19] N. P. Mendler, Recursive types and type constraints in second-order lambda-calculus, in: *Proceedings 2nd Annual IEEE Symp. on Logic in Computer Science, LICS'87 (Ithaca, NY, USA, June 1987)* (IEEE CS Press, Los Alamitos, CA, 1987) 30–36.

[20] N. P. Mendler, Inductive types and type constraints in the second-order lambda-calculus, *Annals of Pure and Applied Logic* **51** (1991) 159–172.

[21] P. Martin-Löf, Hauptsatz for the intuitionistic theory of iterated inductive definitions, in: J. E. Fenstad, ed., *Proceedings 2nd Scandinavian Logic Symp. (Oslo, Norway, June 1970)*, *Studies in Logic and the Foundations of Mathematics*, Vol. 63 (North-Holland, Amsterdam, 1971) 179–216.

[22] B. Nordström, K. Petersson, and J. M. Smith, *Programming in Martin-Löf's Type Theory: An Introduction*, *Int. Series of Monographs on Computer Science*, Vol. 7 (Clarendon Press, Oxford, 1990).

[23] C.-E. Ore, The extended calculus of constructions (ECC) with inductive types, *Information and Computation* **89** (1992) 231–264.

[24] M. Parigot, Programming with proofs: a second order type theory, in: H. Ganziger, ed., *Proceedings 2nd European Symp. on Programming, ESOP'88 (Nancy, France, March 1988)*, *Lecture Notes in Computer Science*, Vol. 300 (Springer-Verlag, Berlin, 1988) 145–159.

[25] M. Parigot, Recursive programming with proofs, *Theoretical Computer Science* **94** (1992) 335–356.

[26] C. Paulin-Mohring, Extracting Fω's programs from proofs in the calculus of constructions, in: *Conf. Record 16th Annual ACM Symp. on Principles of Programming Languages, POPL'89 (Austin, TX, USA, Jan. 1989)* (ACM Press, New York, 1989) 89–104.

[27] C. Paulin-Mohring, Inductive definitions in the system Coq: rules and properties, in: M. Bezem and J. F. Groote, eds., *Proceedings 1st Int. Conf. on Typed Lambda Calculi and Applications, TLCA'93 (Utrecht, The Netherlands, March 1993)*, *Lecture Notes in Computer Science*, Vol. 664 (Springer-Verlag, Berlin, 1993) 328–345.

[28] F. Pfenning and C. Paulin-Mohring, Inductively defined types in the calculus of constructions, in: M. Main, A. Melton, M. Mislove, and D. Schmidt, eds., *Proceedings 5th Int. Conf. on Math. Foundations of Programming Semantics, MFPS'89 (New Orleans, LA, USA, March/Apr. 1989)*, *Lecture Notes in Computer Science*, Vol. 442 (Springer-Verlag, Berlin, 1990) 209–228.

[29] T. Sheard and L. Fegaras, A fold for all seasons, in: *Proceedings 6th ACM SIGPLAN/SIGARCH Int. Conf. on Functional Programming Languages and Computer Architecture, FPCA'93 (Copenhagen, Denmark, June 1993)* (ACM Press, New York, 1993) 233–242.

[30] P. Schroeder-Heister, A natural extension of natural deduction, *Journal of Symbolic Logic* **49** (1984) 1284–1300.

[31] P. Schroeder-Heister, Generalized rules for quantifiers and the completeness of the intuitionistic operators &, $\vee$, $\supset$, $\perp$, $\forall$, $\exists$, in: M. M. Richter et al., eds., *Proceedings Logic Colloquium '83 (Aachen, FRG, July 1983), Vol. 2: Computation and Proof Theory*, *Lecture Notes in Mathematics*, Vol. 1104 (Springer-Verlag, Berlin, 1984) 399–426.

[32] Z. Spławski and P. Urzyczyn, Type fixpoints: iteration vs. recursion, in: *Proceedings 4th ACM SIGPLAN Int. Conf. on Functional Programming, ICFP'99 (Paris, France, Sept. 1999)* (ACM Press, New York, 1999) 102–113.

[33] A. Tarski, A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Math.* **5** (1955) 285–309.

[34] P. Urzyczyn, Positive recursive type assignment, in: J. Wiedermann and P. Hajék, eds., *Proceedings 20th Int. Symp. on Math. Found. of Computer Science, MFCS'95 (Prague, Czech Rep., Aug./Sept. 1995)*, *Lecture Notes in Computer Science*, Vol. 969 (Springer-Verlag, Berlin, 1995) 382–391.

[35] T. Uustalu, Natural deduction for intuitionistic least and greatest fixedpoint logics, with an application to program construction, Ph.D. Thesis, Dissertation TRITA-IT AVH 98:03, Dept. of Teleinformatics, Royal Inst. of Technology, Stockholm (1998).

[36] T. Uustalu, Multi-(co)iteration, categorically, in: J. Penjam, ed., *Proceedings 6th Fenno-Ugric Symposium on Software Technology, FUSST'99 (Sagadi, Estonia, Aug. 1999)*, Report CS 104/99, Inst. of Cybernetics, Tallinn (1999) 259–267.

[37] T. Uustalu and V. Vene, A cube of proof systems for the intuitionistic predicate μ, ν-logic, in: M. Haveraaen and O. Owe, eds., *Selected Papers 8th Nordic Workshop on Programming Theory, NWPT'96 (Oslo, Norway, Dec. 1996)*, Research Report 248, Dept. of Informatics, Univ. of Oslo (1997) 237–246.

[38] T. Uustalu and V. Vene, Primitive (co)recursion and course-of-value (co)iteration, categorically, *INFORMATICA* **10** (1999) 5–26.

[39] T. Uustalu and V. Vene, Mendler-style inductive types, categorically, *Nordic Journal of Computing* **6** (1999) 343–361.

[40] T. Uustalu and V. Vene, Coding recursion à la Mendler, in: *Proceedings 2nd Workshop on Generic Programming, WGP'00 (Ponte de Lima, Portugal, July 2000)*, Tech. Report, Computer Science Dept., Utrecht Univ., to appear.