A Cube of Proof Systems for the Intuitionistic Predicate μ,ν-Logic

Tarmo Uustalu*
Royal Institute of Technology, Stockholm, Sweden

Varmo Vene†
University of Tartu, Estonia

Abstract

This paper is an attempt at a systematizing study of the proof theory of the intuitionistic predicate μ,ν-logic (conventional intuitionistic predicate logic extended with logical constants μ and ν for the least and greatest fixpoint operators on positive predicate transformers). We identify eight proof-theoretically interesting natural-deduction calculi for this logic and propose a classification of these into a cube on the basis of the embeddibility relationships between these.

1 Introduction

μ,ν-logics, i.e. logics with logical constants μ and ν for the least and greatest fixpoint operators on positive predicate transformers, have turned out to be a useful formalism in a number of computer science areas.

The *classical* 1st-order predicate μ,ν-logic can been used as a logic of (non-deterministic) imperative programs and as a database query language. It is also one of the relation description languages studied in descriptive complexity theory (finite model theory) (for a survey on this highly actual application, see [EF95, chapter 7]). Its modal propositional fragment is applicable as a specification language for state machines and as a terminological logic. The typical problems of interest in these applications are *decision* (given a sentence, is it valid?) and *finite model checking* (given a sentence, does a given finite structure satisfy it?).

Viable natural-deduction (N.D.) calculi for *intuitionistic* μ,ν-logics, at the same time, can be fruitfully thought of as typed functional programming languages supporting inductive and coinductive types. In this application, the typical problems of interest are *derivation normalization* (given a derivation, find a normal form for it; computation) together with *derivation checking* (given a sentence, does a given derivation establish it?; program verification) and *derivation finding* (given a sentence, find a derivation that establishes it, if there is any; program construction).

In comparison to the enormous attention that problems centering around the notions of classical validity and satisfaction (especially finite model checking for verification of finite state machines) have received in the research literature on μ,ν-logics, the interest in problems concerned with the notion of intuitionistic proof has been modest. This paper is an attempt at a systematizing study of the proof theory of the intuitionistic predicate μ,ν-logic, building to a due extent upon what we have learned from the literature. We identify eight proof-theoretically interesting N.D. calculi for this logic and propose a classification of these into a cube on the basis of the embeddibility

*Institutionen för Teleinformatik, Kungliga Tekniska Högskolan, Electrum 204, S-164 40 Kista, Sweden; tarmo@it.kth.se

†Arvutiteaduse Instituut, Tartu Ülikool, J. Liivi 2, EE-2484 Tartu, Estonia; varmo@cs.ut.ee

relationships between these. We also discuss the application of the calculi of the cube in program construction.

This paper is organized as follows. In section 2, we prepare the ground for the subsequent exposition by laying down the particular fashion of treating intuitionistic N.D. calculi as typed functional programming languages that we employ in this paper. For reference in the next section, we do this on the example of the standard N.D. calculus for the 2nd-order intuitionistic predicate logic extended with a logical constant σ for the *non-deterministic* arbitrary fixpoint operator on positive predicate transformers. This calculus is denoted $\mathbf{NI}_{\mathrm{P}}$ in this paper. We also fix our terminology and notation. In section 3, we introduce the cube: we define the eight extensions of $\mathbf{NI}_{\mathrm{P}}$ with μ, ν, and show how these work as typed functional programming languages. We also state the embeddibility relationships between the calculi. Section 4 is a discussion of the results presented in section 3 and contains an exposition of two toy examples of program construction. In section 5, we present our conclusions and point out some directions for future work.

2 The setting

The most obvious and orthodox way of seeing an intuitionistic N.D. calculus as a typed functional programming language relies on a direct interpretation of its sentences as types and derivations as algorithms, best formalized by defining an *iso*morphism (i.e. a homomorphism with an inverse that is a homomorphism, too) between the N.D. calculus and a typed λ-calculus[1]—the Curry and Howard isomorphism [How80]. The codomain of the isomorphism for the *the sentential fragment* of $\mathbf{NI}_{\mathrm{P}}$ is an extension of Girard and Reynold's system F [Gir71, Rey74], a nice simply typed λ-calculus. The codomain of the isomorphism for the *full* $\mathbf{NI}_{\mathrm{P}}$, in contrast, is a dependently typed λ-calculus, too heavy to serve as a practical functional programming language.

A more general approach, first put forth by Leivant [Lei83], is to consider a certain *part* of the structure present in sentences and derivations to be essential for typing and computation, and to relate a typed λ-calculus to the N.D. calculus by means of an appropriate *"contracting"* homomorphism. In this paper, we choose a contraction of the following kind: in sentences, suppress occurrences of individuals and the 1st-order quantifiers; in derivations, accordingly, suppress occurrences of individuals and the rules for the 1st-order quantifiers, but also predicates and the rules for the 2nd-order quantifiers, and the rules for σ. The idea is to interpret sentences in their full glory as specifications of functions, so that the N.D. calculus becomes a medium for program construction, while the λ-calculus is its stand-alone component for program "exploitation" (a programming language simpliciter). The codomain of this "contracting" homomorphism for the full $\mathbf{NI}_{\mathrm{P}}$ is an extension of a variant of system F with implicit type coercions, here denoted $\boldsymbol{\lambda}$.

The syntax we employ derives from Martin-Löf's theory of expressions [NPS90, chapter 3]. $s(t_1, \ldots, t_n)$ denotes application of s to $t_1, \ldots, t_n$. $(x_1, \ldots, x_n)s$ denotes abstraction of $x_1, \ldots, x_n$ from s. $s_1 \equiv s_2$ signifies that s_1 is definitionally equal to s_2. A key postulate of the theory says that $((x_1, \ldots, x_n)s)(t_1, \ldots, t_n) \equiv s[t_1/x_1, \ldots, t_n/x_n]$. The operators of expression application and abstraction must not be identified with the operators app and λ of λ-calculi. $\mathsf{app}(\lambda((\zeta)c), d) \rhd^{\beta} ((\zeta)c)(d) \equiv c[d/\zeta]$. Confusing an application of an open λ-term to a λ-term with an application of the operator app to two λ-terms corresponds to confusing plugging a derivation into an open derivation with plugging two derivations into the rule $\supset \mathcal{E}$ ("modus ponens"). For derivations, we use a syntax essentially due to Schroeder-Heister [SH84]. This syntax is a notational variant of the general syntax.

The following is a part of the definition of $\mathbf{NI}_{\mathrm{P}}$, accompanied by the respective part of the definition of $\boldsymbol{\lambda}$. For the sake of brevity, we have left out the logical constants $\top$, $\perp$, $\wedge$, $\vee$, $=^1$, $\exists^1$, and $\exists^2$, and the associating inference and contraction rules. (In fact, these are not too essential, in the virtue of the well-known homomorphism from the full $\mathbf{NI}_{\mathrm{P}}$ to the fragment without them.)

[1] A homomorphism between a N.D. calculus and a typed λ-calculus is a mapping that sends sentences to type terms and derivations to λ-terms such that, if a derivation establishes a sentence, then the corresponding λ-term types with the corresponding type term, and, if one derivation reduces to another, then the λ-term corresponding to the first reduces to this that corresponds to the other.

$$\mathsf{Pred}^{(n)} \equiv \underbrace{\mathsf{Indiv}\otimes\ldots\otimes\mathsf{Indiv}}_{n \text{ times}} \twoheadrightarrow \mathsf{Sent}$$

$$\supset: \mathsf{Sent}\otimes\mathsf{Sent}\twoheadrightarrow\mathsf{Sent} \qquad \forall^1 : (\mathsf{Indiv}\twoheadrightarrow\mathsf{Sent})\twoheadrightarrow\mathsf{Sent}$$

$$\forall^2 : (\mathsf{Pred}^{(n)}\twoheadrightarrow\mathsf{Sent})\twoheadrightarrow\mathsf{Sent} \qquad \sigma : (\mathsf{Pred}^{(n)}\twoheadrightarrow\mathsf{Pred}^{(n)})\twoheadrightarrow\mathsf{Pred}^{(n)}$$

$$\supset\mathcal{I}: \frac{\dfrac{A}{B}}{A\supset B} \qquad \supset\mathcal{E}: \frac{A\supset B \quad A}{B} \qquad \forall^1\mathcal{I}: \frac{(x)\ P(x)}{\forall^1(P)} \qquad \forall^1\mathcal{E}: (x)\,\frac{\forall^1(P)}{P(x)}$$

$$\forall^2\mathcal{I}: \frac{(X)\ F(X)}{\forall^2(F)} \qquad \forall^2\mathcal{E}: (X)\,\frac{\forall^2(F)}{F(X)} \qquad \sigma\mathcal{I}: \frac{F(\sigma(F))}{\sigma(F)} \qquad \sigma\mathcal{E}: \frac{\sigma(F)(s)}{F(\sigma(F))(s)}$$

$$\cfrac{\cfrac{\begin{array}{c}A\\ \vdots\ \Gamma\\ B\end{array}}{A\supset B}\supset\mathcal{I} \qquad \begin{array}{c}\vdots\ \mathrm{E}\\ A\end{array}}{B}\supset\mathcal{E} \quad \rhd^\beta \quad \begin{array}{c}\vdots\ \mathrm{E}\\ A\\ \vdots\ \Gamma\\ B\end{array} \qquad \cfrac{\cfrac{\begin{array}{c}(x)\ \vdots\ \Gamma\\ P(x)\end{array}}{\forall^1 P}\forall^1\mathcal{I}}{P(t)}\forall^1\mathcal{E}(t) \quad \rhd^\beta \quad \begin{array}{c}\vdots\ \Gamma(t)\\ P(t)\end{array}$$

$$\cfrac{\cfrac{\begin{array}{c}(X)\ \vdots\ \Gamma\\ F(X)\end{array}}{\forall^2 F}\forall^2\mathcal{I}}{F(R)}\forall^2\mathcal{E}(R) \quad \rhd^\beta \quad \begin{array}{c}\vdots\ \Gamma(R)\\ F(R)\end{array} \qquad \cfrac{\cfrac{\begin{array}{c}\vdots\ \Gamma\\ F(\sigma(F))(s)\end{array}}{\sigma(F)(s)}\sigma\mathcal{I}}{F(\sigma(F))(s)}\sigma\mathcal{E} \quad \rhd^\beta \quad \begin{array}{c}\vdots\ \Gamma\\ F(\sigma(F))(s)\end{array}$$

$$\supset: \mathsf{Type}\otimes\mathsf{Type}\twoheadrightarrow\mathsf{Type} \qquad \forall^2 : (\mathsf{Type}\twoheadrightarrow\mathsf{Type})\twoheadrightarrow\mathsf{Type} \qquad \sigma : (\mathsf{Type}\twoheadrightarrow\mathsf{Type})\twoheadrightarrow\mathsf{Type}$$

$$\lambda : (\mathsf{Term}^A\twoheadrightarrow\mathsf{Term}^B)\twoheadrightarrow\mathsf{Term}^{A\supset B} \qquad \mathsf{app} : \mathsf{Term}^{A\supset B}\otimes\mathsf{Term}^A\twoheadrightarrow\mathsf{Term}^B$$

$$(\zeta)\zeta : \mathbf{\Pi}((X)\mathsf{Term}^{F(X)})\twoheadrightarrow\mathsf{Term}^{\forall^2(F)} \qquad (\zeta)\zeta : \mathbf{\Pi}((X)\mathsf{Term}^{\forall^2(F)}\twoheadrightarrow\mathsf{Term}^{F(X)})$$

$$(\zeta)\zeta : \mathsf{Term}^{F(\sigma(F))}\twoheadrightarrow\mathsf{Term}^{\sigma(F)} \qquad (\zeta)\zeta : \mathsf{Term}^{\sigma(F)}\twoheadrightarrow\mathsf{Term}^{F(\sigma(F))}$$

$$\mathsf{app}(\lambda(c), e) \rhd^\beta c(e)$$

3 The cube

We shall now introduce our cube of eight extensions of $\mathbf{NI}_{\mathrm{P}}$ with μ and ν. In parallel to the N.D. calculi, we also introduce the correspondent λ-calculi.

From the functional programming point-of-view, in each of the N.D. calculi, μ is an inductive type former and the μ-elimination rule is a recursion operator; ν, dually, is a coinductive type former and the ν-introduction rule is a corecursion operator. Each N.D. calculus of the cube is characterized by its own inference and contraction rules for μ and ν. The kind of recursion and corecursion operators captured by the inference rules $\mu\mathcal{E}$ and $\nu\mathcal{I}$ thus varies. In the calculi at the bottom front nodes ($\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ and $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$), these are iteration and coiteration; in the calculi at the top front nodes ($\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\mathbf{r}}$ and $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\mathbf{r}}$), —(primitive) recursion and corecursion; in the calculi at the bottom back nodes ($\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\star}$ and $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\star}$), —courses-of-values iteration and coiteration; in the calculi at the top back nodes ($\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\mathbf{r}\star}$ and $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\mathbf{r}\star}$), —courses-of-values recursion and corecursion. Courses-of-values iteration is a function definition scheme where the output of a function on a given input is defined in terms of not only the outputs on the inputs that immediately precede the given one, as in simple iteration, but in terms of the outputs on all inputs that precede the given one. (Cf. the relation of courses-of-values induction on naturals to simple mathematical induction.)

In order to save space, we only treat μ below; ν is in all aspects dual to μ. We also give the definitions of just four of the total of eight N.D. calculi in the cube.

$$\begin{tikzcd}
& \mathbf{NI}\mathrm{P}^{\mu,\nu}_{\mathbf{r}\star} \arrow[rr] & & \textsc{m}\mathbf{NI}\mathrm{P}^{\mu,\nu}_{\mathbf{r}\star} \\
\mathbf{NI}\mathrm{P}^{\mu,\nu}_{\mathbf{r}} \arrow[rr] \arrow[ur] & & \textsc{m}\mathbf{NI}\mathrm{P}^{\mu,\nu}_{\mathbf{r}} \arrow[ur] & \\
& \mathbf{NI}\mathrm{P}^{\mu,\nu}_{\star} \arrow[rr] \arrow[uu] & & \textsc{m}\mathbf{NI}\mathrm{P}^{\mu,\nu}_{\star} \arrow[uu] \\
\mathbf{NI}\mathrm{P}^{\mu,\nu} \arrow[rr] \arrow[ur] \arrow[uu] & & \textsc{m}\mathbf{NI}\mathrm{P}^{\mu,\nu} \arrow[ur] \arrow[uu] &
\end{tikzcd}$$

3.1 The calculi

Ideally, μ should be a logical constant for the least fixpoint operator on predicate transformers.

$$\mu : (\mathsf{Pred}^{(n)} \twoheadrightarrow \mathsf{Pred}^{(n)}) \twoheadrightarrow \mathsf{Pred}^{(n)}$$

$$\mu : (\mathsf{Type} \twoheadrightarrow \mathsf{Type}) \twoheadrightarrow \mathsf{Type}$$

However, not every predicate transformer has a least fixpoint. In the calculi of the cube, $\mu(F)$ faithfully expresses the least fixed point of F, if the predicate transformer F is *positive*. A predicate transformer $(X)P$ is called positive iff every occurrence of X in P appears within an even number of antecedents of implications and within no σ. Positivity is a simple sufficient condition for monotonicity, which, in turn, is a sufficient condition for the existence of least and greatest fixpoints.

As regards to applications of μ to non-positive predicate transformers, two approaches are used in the cube. In the calculi on the left, these applications are simply forbidden. In the calculi on the right (whose names contain the letter M, for Mendler), in constrast, they are allowed, and there, for any predicate transformer F, $\mu(F)$ actually expresses the least fixpoint of a certain positive majorant of F. In the particular case, if F itself happens to be monotone, the majorant is equivalent to F.

The calculi on the left

In these calculi, the way how a μ-detour is removed from a derivation depends on a part of the structure of the predicate transformer F that μ is applied to. This part of the structure can be nicely condensed into a pseudo-'predicate transformer' that we call the shape of F (notation $|F|$). Given any predicate transformer $(X)P$, its shape is obtained from $(X)P$ by replacing all subsentences of P where X does not occur with a special pseudo-sentence _ ("blank").

The contraction rule for μ is best formulated by making reference to an auxiliary inference rule $\mathcal{M}^+$. Both the inference rules for μ and the inference rule $\mathcal{M}^+$ are parametric in a shape; it is forbidden to use a non-positive shape as a value of the parameter. The auxiliary inference rule $\mathcal{M}^+$ states that any predicate transformer with a given (necessarily positive) shape is monotone. It encapsulates a shape-dependent derivation without applications of auxiliary inference rules.

$$\mathcal{M}^+_{|F|} : \quad \frac{F(Q)(s) \qquad (z)\,\dfrac{Q(z)}{R(z)}}{F(R)(s)}$$

$$\mathsf{map}_{|F|} : \mathsf{Term}^{F(Q)} \otimes (\mathsf{Term}^{Q} \twoheadrightarrow \mathsf{Term}^{R}) \twoheadrightarrow \mathsf{Term}^{F(R)}$$

As an example of a calculus on the left, we present the calculus at the left-hand bottom front node, $\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ (the "basic" calculus). The inference rule $\mu\mathcal{I}$ states that, for any predicate transformer F with a given (necessarily positive) shape, $\mu(F)$ is a pre-fixpoint of F. The inference rule $\mu\mathcal{E}$ (Park's fixpoint induction principle) states that, for any predicate transformer F with a given (necessarily positive) shape, $\mu(F)$ is subsumed by any pre-fixpoint of F. Note the use of the inference rule $\mathcal{M}^+$ in the contraction rule for μ.

$$\mu\mathcal{I}_{|F|} : \ \frac{F(\mu(F))(s)}{\mu(F)(s)} \qquad\qquad \mu\mathcal{E}_{|F|} : \ \frac{\mu(F)(s) \qquad (y)\,\dfrac{F(R)(y)}{R(y)}}{R(s)}$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\,\Gamma\\ F(\mu(F))(s)\end{matrix}}{\mu(F)(s)}\,\mu\mathcal{I}_{|F|} \qquad \begin{matrix}F(R)(y)\\ (y)\,\vdots\,\mathrm{E}\\ R(y)\end{matrix}}{R(s)}\,\mu\mathcal{E}_{|F|} \quad \rhd^{\beta} \quad \begin{matrix}\dfrac{\begin{matrix}\vdots\,\Gamma\\ F(\mu(F))(s)\end{matrix} \qquad \begin{matrix}\dfrac{\overline{\mu(F)(z)}\,\zeta \qquad \begin{matrix}F(R)(y)\\ (y)\,\vdots\,\mathrm{E}\\ R(y)\end{matrix}}{R(z)}\,\mu\mathcal{E}_{|F|}\\ {\scriptstyle [\zeta]}\\ {\scriptstyle [z]}\end{matrix}}{F(R)(s)}\,\mathcal{M}^{+}_{|F|}\\ \vdots\,\mathrm{E}(s)\\ R(s)\end{matrix}$$

$$\begin{gathered}\mathsf{wrap}_{|F|} : \mathsf{Term}^{F(\mu(F))} \twoheadrightarrow \mathsf{Term}^{\mu(F)}\\ \mathsf{cata}_{|F|} : \mathsf{Term}^{\mu(F)} \otimes (\mathsf{Term}^{F(R)} \twoheadrightarrow \mathsf{Term}^{R}) \twoheadrightarrow \mathsf{Term}^{R}\\ \mathsf{cata}_{|F|}(\mathsf{wrap}_{|F|}(c), e) \rhd^{\beta} e(\mathsf{map}_{|F|}(c, (\zeta)\mathsf{cata}_{|F|}(\zeta, e)))\end{gathered}$$

The calculi on the right

In these calculi, the inference rules for μ do not have the shape parameter, and there is no analog of the inference rule $\mathcal{M}^+$. The reason is that μ finds the least fixpoint of the majorant F^{e} of a given predicate transformer F, and $|F^{\mathsf{e}}|$ is the same for any F (for the definition of F^{e}, see the next section).

As a first example of a calculus on the right, let us present the calculus at the right-hand bottom front node, $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$. The inference rule $\mu\mathcal{I}$ states that, for any predicate transformer F, $\mu(F)$ is a pre-fixpoint of F^{e}. The inference rule $\mu\mathcal{E}$ states that, for any predicate transformer F, $\mu(F)$ is subsumed by any pre-fixpoint of F^{e}.

$$\mu\mathcal{I} : \ \frac{F(Q)(s) \qquad (u)\,\dfrac{Q(u)}{\mu(F)(u)}}{\mu(F)(s)} \qquad\qquad \mu\mathcal{E} : \ \frac{\mu(F)(s) \qquad (Y,y)\,\dfrac{F(Y)(y) \qquad (u)\,\dfrac{Y(u)}{R(u)}}{R(y)}}{R(s)}$$

$$\dfrac{\dfrac{\begin{matrix}\vdots\,\Gamma\\ F(Q)(s)\end{matrix} \quad \begin{matrix}Q(u)\\ (u)\,\vdots\,\Delta\\ \mu(F)(u)\end{matrix}}{\mu(F)(s)}\,\mu\mathcal{I} \qquad \begin{matrix}F(Y)(y) \quad (u)\,\dfrac{Y(u)}{R(u)}\\ (Y,y)\,\vdots\,\mathrm{E}\\ R(y)\end{matrix}}{R(s)}\,\mu\mathcal{E} \quad \rhd^{\beta} \quad \begin{matrix}\begin{matrix}\vdots\,\Gamma\\ F(Q)(s)\end{matrix} \qquad \begin{matrix}\dfrac{\begin{matrix}\overline{Q(u)}\,v\\ \vdots\,\Delta(u)\\ \mu(F)(u)\end{matrix} \qquad \begin{matrix}F(Y)(y) \quad (u)\,\dfrac{Y(u)}{R(u)}\\ (Y,y)\,\vdots\,\mathrm{E}\\ R(y)\end{matrix}}{R(u)}\,\mu\mathcal{E}\\ {\scriptstyle [v]}\\ {\scriptstyle [u]}\end{matrix}\\ \vdots\,\mathrm{E}(Q,s)\\ R(s)\end{matrix}$$

$$\begin{gathered}\mathsf{mapwrap} : \mathsf{Term}^{F(Q)} \otimes (\mathsf{Term}^{Q} \twoheadrightarrow \mathsf{Term}^{\mu(F)}) \twoheadrightarrow \mathsf{Term}^{\mu(F)}\\ \mathsf{iter} : \mathsf{Term}^{\mu(F)} \otimes \mathbf{\Pi}((Y)\mathsf{Term}^{F(Y)} \otimes (\mathsf{Term}^{Y} \twoheadrightarrow \mathsf{Term}^{R}) \twoheadrightarrow \mathsf{Term}^{R}) \twoheadrightarrow \mathsf{Term}^{R}\\ \mathsf{iter}(\mathsf{mapwrap}(c, d), e) \rhd^{\beta} e(c, (v)\mathsf{iter}(d(v), e))\end{gathered}$$

In the two calculi we have presented so far (the calculi at the bottom front nodes), the inference rule $\mu\mathcal{E}$ behaves as an iteration operator. In the calculi at the top front nodes, the inference rule

$\mu\mathcal{E}$ behaves as a (primitive) recursion operator. As an example, we present the calculus at the right-hand top front node, $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\mathbf{r}}$.

$$\mu\mathcal{I}: \quad \frac{F(Q)(s) \qquad (u)\,\dfrac{Q(u)}{\mu(F)(u)}}{\mu(F)(s)} \qquad\qquad \mu\mathcal{E}: \quad \frac{\mu(F)(s) \qquad (Y,y)\,\dfrac{F(Y)(y) \qquad (u)\,\dfrac{Y(u)}{R(u)} \qquad (u)\,\dfrac{Y(u)}{\mu(F)(u)}}{R(y)}}{R(s)}$$

$$\cfrac{\cfrac{\begin{array}{c}\vdots\,\Gamma\\ F(Q)(s)\end{array} \quad \begin{array}{c}Q(u)\\ (u)\,\vdots\,\Delta\\ \mu(F)(u)\end{array}}{\mu(F)(s)}\,\mu\mathcal{I} \qquad \begin{array}{c}F(Y)(y) \quad (u)\,\dfrac{Y(u)}{R(u)} \quad (u)\,\dfrac{Y(u)}{\mu(F)(u)}\\ (Y,y)\,\vdots\,\mathrm{E}\\ R(y)\end{array}}{R(s)}\,\mu\mathcal{E} \quad \rhd^{\beta} \quad \begin{array}{c}\vdots\,\Gamma\\ F(Q)(s)\end{array} \quad \begin{array}{c}\cfrac{\begin{array}{c}\overline{Q(u)}\;v\\ \vdots\,\Delta(u)\\ \mu(F)(u)\end{array} \qquad \begin{array}{c}F(Y)(y) \quad (u)\,\dfrac{Y(u)}{R(u)} \quad (u)\,\dfrac{Y(u)}{\mu(F)(u)}\\ (Y,y)\,\vdots\,\mathrm{E}\\ R(y)\end{array}}{R(u)}\,\mu\mathcal{E}\\ {}^{[v]}_{[u]}\\ \vdots\,\mathrm{E}(Q,s)\\ R(s)\end{array} \quad \begin{array}{c}Q(u)\\ (u)\,\vdots\,\Delta\\ \mu(F)(u)\end{array}$$

$$\mathsf{mapwrap} : \mathsf{Term}^{F(Q)} \otimes (\mathsf{Term}^{Q} \twoheadrightarrow \mathsf{Term}^{\mu(F)}) \twoheadrightarrow \mathsf{Term}^{\mu(F)}$$
$$\mathsf{rec} : \mathsf{Term}^{\mu(F)} \otimes \mathbf{\Pi}((Y)\mathsf{Term}^{F(Y)} \otimes (\mathsf{Term}^{Y} \twoheadrightarrow \mathsf{Term}^{R}) \otimes (\mathsf{Term}^{Y} \twoheadrightarrow \mathsf{Term}^{\mu(F)}) \twoheadrightarrow \mathsf{Term}^{R}) \twoheadrightarrow \mathsf{Term}^{R}$$
$$\mathsf{rec}(\mathsf{mapwrap}(c,d),e) \rhd^{\beta} e(c,(v)\mathsf{rec}(d(v),e),d)$$

In the calculi at the bottom back nodes, the inference rule $\mu\mathcal{E}$ behaves as a courses-of-values iteration operator. As an example, we present the calculus at the right-hand bottom back node, $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\star}$.

$$\mu\mathcal{I}: \quad \frac{F(Q)(s) \qquad (u)\,\dfrac{Q(u)}{\mu(F)(u)} \qquad (u)\,\dfrac{Q(u)}{F(Q)(u)}}{\mu(F)(s)} \qquad\qquad \mu\mathcal{E}: \quad \frac{\mu(F)(s) \qquad (Y,y)\,\dfrac{F(Y)(y) \qquad (u)\,\dfrac{Y(u)}{R(u)} \qquad (u)\,\dfrac{Y(u)}{F(Y)(u)}}{R(y)}}{R(s)}$$

$$\cfrac{\cfrac{\begin{array}{c}\vdots\,\Gamma\\ F(Q)(s)\end{array} \quad \begin{array}{c}Q(u)\\ (u)\,\vdots\,\Delta\\ \mu(F)(u)\end{array} \quad \begin{array}{c}Q(u)\\ (u)\,\vdots\,\mathrm{K}\\ F(Q)(u)\end{array}}{\mu(F)(s)}\,\mu\mathcal{I} \qquad \begin{array}{c}F(Y)(y) \quad (u)\,\dfrac{Y(u)}{R(u)} \quad (u)\,\dfrac{Y(u)}{F(Y)(u)}\\ (Y,y)\,\vdots\,\mathrm{E}\\ R(y)\end{array}}{R(s)}\,\mu\mathcal{E}$$

$$\rhd^{\beta} \quad \begin{array}{c}\vdots\,\Gamma\\ F(Q)(s)\end{array} \quad \begin{array}{c}\cfrac{\begin{array}{c}\overline{Q(u)}\;v\\ \vdots\,\Delta(u)\\ \mu(F)(u)\end{array} \qquad \begin{array}{c}F(Y)(y) \quad (u)\,\dfrac{Y(u)}{R(u)} \quad (u)\,\dfrac{Y(u)}{F(Y)(u)}\\ (Y,y)\,\vdots\,\mathrm{E}\\ R(y)\end{array}}{R(u)}\,\mu\mathcal{E}\\ {}^{[v]}_{[u]}\\ \vdots\,\mathrm{E}(Q,s)\\ R(s)\end{array} \quad \begin{array}{c}Q(u)\\ (u)\,\vdots\,\mathrm{K}\\ F(Q)(u)\end{array}$$

$$\mathsf{cv\text{-}mapwrap} : \mathsf{Term}^{F(Q)} \otimes (\mathsf{Term}^{Q} \twoheadrightarrow \mathsf{Term}^{\mu(F)}) \otimes (\mathsf{Term}^{Q} \twoheadrightarrow \mathsf{Term}^{F(Q)}) \twoheadrightarrow \mathsf{Term}^{\mu(F)}$$
$$\mathsf{cv\text{-}iter} : \mathsf{Term}^{\mu(F)} \otimes \mathbf{\Pi}((Y)\mathsf{Term}^{F(Y)} \otimes (\mathsf{Term}^{Y} \twoheadrightarrow \mathsf{Term}^{R}) \otimes (\mathsf{Term}^{Y} \twoheadrightarrow \mathsf{Term}^{F(Y)}) \twoheadrightarrow \mathsf{Term}^{R}) \twoheadrightarrow \mathsf{Term}^{R}$$
$$\mathsf{cv\text{-}iter}(\mathsf{cv\text{-}mapwrap}(c,d,k),e) \rhd^{\beta} e(c,(v)\mathsf{cv\text{-}iter}(d(v),e),k)$$

$\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ / $\boldsymbol{\lambda}^{\mu,\nu}$ and $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ / $\mathrm{M}\boldsymbol{\lambda}^{\mu,\nu}$ were first formulated by Leivant [Lei90]; he acknowledged the work of Mendler [Men87], which is the first publication of a variant of $\mathrm{M}\boldsymbol{\lambda}^{\mu,\nu}_{\mathbf{r}}$. $\boldsymbol{\lambda}^{\mu,\nu}_{\mathbf{r}}$ appeared first in Geuvers [Geu92]. We are not aware of any accounts of the calculi at the back nodes of the cube in the literature.

Hagino's category-theoretically motivated λ-calculus in [Hag87] is a relative of $\boldsymbol{\lambda}^{\mu,\nu}$. A λ-calculus with two destructors for μ (namely, cata and an inverse of wrap) appears in [Gre92] and [How92, chapter 3]. Martin-Löf published a paper about a N.D. calculus supporting iterated inductive definitions of predicates as early as in 1971 [ML71].

3.2 The embeddings

There exist embeddings (i.e. *injective* homomorphisms) between any two N.D. calculi at adjacent nodes in the cube in both directions. In addition, there exists a homomorphism from the basic calculus $\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ to $\mathbf{NI}_{\mathrm{P}}$. The only logical constants and inference rules not mapped identically by these are μ and ν and the associating inference rules. We characterize each mapping here by the image of μ only.

The embeddings in the forward directions

- left-to-right
$$\mu^{\sharp}(F) \equiv \mu(F)$$
- bottom-to-top
$$\mu^{\sharp}(F) \equiv \mu(F)$$
- front-to-back
$$\mu^{\sharp}(F) \equiv \mu(F^{\mathbf{i}}) \text{ where } F^{\mathbf{i}} \equiv (X)(x)F(X)(x) \vee X(x)$$
For any F, $F^{\mathbf{i}}$ is inflationary.

The embeddings in the backward directions

- right-to-left
$$\mu^{\sharp}(F) \equiv \mu(F^{\mathbf{e}}) \text{ where } F^{\mathbf{e}} \equiv (X)(x)\exists^2((Y)\forall^1((u)Y(u) \supset X(u)) \wedge F(Y)(x))$$
For any F, $F^{\mathbf{e}}$ is a positive (and hence monotone) majorant of F. (Note the use of 2nd-order quantifiers here.)
- top-to-bottom, on the left, and on the right
$$\mu^{\sharp}(F) \equiv \sigma((Z)\mu((X)F((z)X(z) \wedge Z(z))))$$
$$\mu^{\sharp}(F) \equiv \sigma((Z)\mu((X)(x)\forall^1((u)X(u) \supset Z(u)) \wedge F(X)(x)))$$
(Note the use of σ here.)
- back-to-front, on the left, and on the right
$$\mu^{\sharp}(F) \equiv \mu((X)F((z)\exists^2((Z)\forall^1((u)Z(u) \supset X(u) \wedge F(Z)(u)) \wedge Z(z))))$$
$$\mu^{\sharp}(F) \equiv \mu((X)(x)\forall^1((u)X(u) \supset F(X)(u)) \wedge F(X)(x))$$

The embeddings in the forward directions are almost straightforward. The top-to-bottom embedding from $\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\mathbf{r}}$ to $\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ is implicit in [Par90]. For the corresponding λ-calculi, it is explicitly given in [Geu92].

The homomorphism from $\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ to $\mathbf{NI}_{\mathrm{P}}$

$$\mu^{\sharp} \equiv (x)\forall^2((X)\forall^1((y)F(X)(y) \supset X(y)) \supset X(x))$$

This definition of the μ and ν and the associating inference rules of $\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ via the 2nd-order quantifiers and the associating inference rules is well-known; it is due to Leivant [Lei83] and Böhm and Berarducci [BB85].

$\mathbf{NI}_{\mathrm{P}}$ / $\boldsymbol{\lambda}$ enjoy the property of strong normalizability. Therefore, the N.D. / λ-calculi of the cube also enjoy this property, since for each of them, there exists a homomorphism to $\mathbf{NI}_{\mathrm{P}}$ / $\boldsymbol{\lambda}$.

4 Discussion

The calculi of the cube are equally powerful in terms of expressible *functions* (i.e. denotationally), but differ in terms of expressible *algorithms* (i.e. operationally). Recursion and courses-of-values iteration allow more efficient programming than plain iteration. The iterative predecessor algorithm for naturals which is expressible in $\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ and $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$ requires linear time. In $\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\mathbf{r}}$ and $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\mathbf{r}}$, in contrast, one can express a better recursive algorithm that computes in constant time. Below we give two examples of program scheme construction in the N.D. calculi of the cube.

Example: The "factorial" scheme

Define

$$\begin{array}{rcl} \mathsf{N}(R)(s) & \equiv & \mathsf{Zero}(s) \vee (\mathsf{Succ}(s) \wedge R(\mathsf{p}(s))) \\ \mathsf{Nat} & \equiv & \mu(\mathsf{N}) \\ \mathsf{Fact}(s) & \equiv & \mathsf{Nat}(\mathsf{fact}(s)) \end{array}$$

The scheme of specifying a one-place function f on naturals in terms of a number g_0 and a two-place function g_1 by the equations $f(0) = g_0$ and $f(n) = g_1(f(n-1), n-1)$ $(n \geq 1)$ is expressible as follows:

$$\begin{array}{c} \forall^1((x)\mathsf{Zero}(x) \supset \mathsf{Fact}(x)) \qquad \begin{array}{c}\forall^1((x)\mathsf{Succ}(x) \\ \wedge(\mathsf{Fact}(\mathsf{p}(x)) \wedge \mathsf{Nat}(\mathsf{p}(x))) \supset \mathsf{Fact}(x))\end{array} \\ \vdots \\ \forall^1((x)\mathsf{Nat}(x) \supset \mathsf{Fact}(x)) \end{array}$$

The following iterative program scheme in $\mathrm{M}\boldsymbol{\lambda}^{\mu,\nu}$, which, given programs for g_0, g_1, computes $f(n)$ as the first component of the pair $f'(n) = (f(n), n)$, corresponds to a program scheme constructible in $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$:

$$(g_0, g_1)\lambda((\beta)\mathsf{fst}(\mathsf{iter}(\beta, (\gamma, \delta)\mathsf{case}\left(\gamma, \begin{array}{l} (\xi)\langle \begin{array}{l}\mathsf{app}(g_0, \xi) \\ \mathsf{mapwrap}(\mathsf{inl}(\xi), (\zeta)\zeta)\end{array} \rangle \\ (\xi)\langle \begin{array}{l}\mathsf{app}(g_1, \langle \mathsf{fst}(\xi), \delta(\mathsf{snd}(\xi))\rangle) \\ \mathsf{mapwrap}(\mathsf{inr}(\langle \mathsf{fst}(\xi), \mathsf{snd}(\delta(\mathsf{snd}(\xi)))\rangle), (\zeta)\zeta)\end{array} \rangle \end{array}\right))))$$

The following recursive program scheme in $\mathrm{M}\boldsymbol{\lambda}^{\mu,\nu}_{\mathbf{r}}$ corresponds to a program scheme constructible in $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}_{\mathbf{r}}$:

$$(g_0, g_1)\lambda((\beta)\mathsf{rec}(\beta, (\gamma, \delta, \iota)\mathsf{case}\left(\gamma, \begin{array}{l} (\xi)\mathsf{app}(g_0, \xi) \\ (\xi)\mathsf{app}(g_1, \langle \mathsf{fst}(\xi), \langle \delta(\mathsf{snd}(\xi)), \iota(\mathsf{snd}(\xi))\rangle\rangle) \end{array}\right)))$$

Example: The "Fibonacci" scheme

Define

$$\begin{array}{rcl} \mathsf{N}(R)(s) & \equiv & \mathsf{Zero}(s) \vee (\mathsf{Succ}(s) \wedge R(\mathsf{p}(s))) \\ \mathsf{Nat} & \equiv & \mu(\mathsf{N}) \\ \mathsf{Fibo}(s) & \equiv & \mathsf{Nat}(\mathsf{fibo}(s)) \end{array}$$

The scheme of specifying a one-place function f on naturals in terms of numbers g_0, g_1 and a two-place function g_2 by the equations $f(0) = g_0$, $f(1) = g_1$, and $f(n) = g_2(f(n-1), f(n-2))$ $(n \geq 2)$ is expressible as follows:

$$\begin{array}{c} \begin{array}{c}\forall^1((x)\mathsf{Zero}(x) \\ \supset \mathsf{Fibo}(x))\end{array} \qquad \begin{array}{c}\forall^1((x)\mathsf{Succ}(x) \wedge \mathsf{Zero}(\mathsf{p}(x)) \\ \supset \mathsf{Fibo}(x))\end{array} \qquad \begin{array}{c}\forall^1((x)\mathsf{Succ}(x) \wedge (\mathsf{Succ}(\mathsf{p}(x)) \\ \wedge(\mathsf{Fibo}(\mathsf{p}(x)) \wedge \mathsf{Fibo}(\mathsf{p}(\mathsf{p}(x))))) \\ \supset \mathsf{Fibo}(x))\end{array} \\ \vdots \\ \forall^1((x)\mathsf{Nat}(x) \supset \mathsf{Fibo}(x)) \end{array}$$

The following iterative program scheme in $\mathrm{M}\boldsymbol{\lambda}^{\mu,\nu}$, which, given programs for g_0, g_1, and g_2, computes $f(n)$ as the first component of the pair $f'(n) = (f(n), f(n-1))$, corresponds to a program scheme constructible in $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}^{\mu,\nu}$:

$$(g_0,g_1,g_2)\lambda((\beta)\mathsf{fst}(\mathsf{iter}(\beta,(\gamma,\delta)\mathsf{case}\left(\begin{array}{ll}\gamma, & \begin{array}{l}(\xi)\langle\begin{array}{l}\mathsf{app}(g_0,\xi)\\ \mathsf{inl}(\xi)\end{array}\rangle\\ (\xi)\langle\begin{array}{l}\mathsf{case}\left(\begin{array}{ll}\mathsf{snd}(\delta(\mathsf{snd}(\xi))), & \begin{array}{l}(\xi')\mathsf{app}(g_1,\langle\mathsf{fst}(\xi),\xi'\rangle)\\ (\xi')\mathsf{app}(g_2,\langle\mathsf{fst}(\xi),\langle\mathsf{fst}(\xi'),\\ \quad\langle\mathsf{fst}(\delta(\mathsf{snd}(\xi))),\mathsf{snd}(\xi')\rangle\rangle\rangle)\end{array}\end{array}\right)\\ \mathsf{inr}(\langle\mathsf{fst}(\xi),\mathsf{fst}(\delta(\mathsf{snd}(\xi)))\rangle)\end{array}\rangle\end{array}\end{array}\right))))$$

The following 'courses-of-values'-iterative program scheme in $\mathrm{M}\boldsymbol{\lambda}_{\star}^{\mu,\nu}$ corresponds to a program scheme constructible in $\mathrm{M}\mathbf{NI}_{\mathrm{P}}{}_{\star}^{\mu,\nu}$:

$$(g_0,g_1,g_2)\lambda((\beta)\mathsf{cv\text{-}iter}(\beta,(\gamma,\delta,\kappa)\mathsf{case}\left(\begin{array}{ll}\gamma, & \begin{array}{l}(\xi)\mathsf{app}(g_0,\xi)\\ (\xi)\mathsf{case}\left(\begin{array}{ll}\kappa(\mathsf{snd}(\xi)), & \begin{array}{l}(\xi')\mathsf{app}(g_1,\langle\mathsf{fst}(\xi),\xi'\rangle)\\ (\xi')\mathsf{app}(g_2,\langle\mathsf{fst}(\xi),\langle\mathsf{fst}(\xi'),\\ \quad\langle\delta(\mathsf{snd}(\xi)),\delta(\mathsf{snd}(\xi'))\rangle\rangle\rangle)\end{array}\end{array}\right)\end{array}\end{array}\right)))$$

5 Conclusion

In this paper, we presented eight N.D. calculi arranged into a cube, and showed how these function as media for construction of functional programs. We were pleased to discover that each provides support for recursive-corecursive programming with a *meaningful* recursion-corecursion operator. We intend to continue this research with a closer inspection of the perspectives of practical applicability of these calculi in program construction; this concerns both specification methodology and computer assistance in program construction. We also intend to look into the connections with the category-theoretic technology of program construction advocated by the mathematics of program construction (calculation of programs) community.

Acknowledgement

Varmo Vene is grateful to the Stockholm-Ladugårdslandet Club of District 2350 of Rotary International for financing his two two-month visits to the Royal Institute of Technology in Stockholm during 1996.

References

[BB85] Corrado Böhm and Alessandro Berarducci. Automatic synthesis of typed λ-programs on term algebras. *Theoretical Computer Science*, 39(2–3):135–154, 1985.

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory.* Perspectives in Mathematical Logic. Springer-Verlag, Berlin, 1995.

[Geu92] Herman Geuvers. Inductive and coinductive types with iteration and recursion. In B. Nordström, K. Pettersson, and G. Plotkin, editors, *Preliminary Proceedings Workshop on Types for Proofs and Programs, Båstad, Sweden, 8–12 June 1992*, pages 193–217. Dept. of Computing Science, Chalmers Univ. of Technology and Göteborg Univ., 1992. (ftp://ftp.cs.chalmers.se/pub/cs-reports/baastad.92/proc.dvi.Z.)

[Gir71] Jean-Yves Girard. Une extension de l'interprétation de Gödel à l'analyse, et son application à l'élimination des coupures dans l'analyse et la théorie des types. In J. E. Fenstad, editor, *Proceedings 2nd Scandinavian Logic Symp., Oslo, Norway, 18–20 June 1970*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92. North-Holland, Amsterdam, 1971.

[Gre92] John Greiner. Programming with inductive and co-inductive types. Technical Report CMU-CS-92-109, School of Computer Science, Carnegie-Mellon Univ., Pittsburgh, PA, USA, January 1992.

[Hag87] Tatsuya Hagino. A typed lambda calculus with categorical type constructors. In D. H. Pitt, A. Poigné, and D. E. Rydeheard, editors, *Proceedings 2nd Int'l Conf. on Category Theory and Computer Science, CTCS'87, Edinburgh, UK, 7–9 Sept 1987*, volume 283 of *Lecture Notes in Computer Science*, pages 140–157. Springer-Verlag, Berlin, 1987.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, London, 1980. Reprint of a manuscript written 1969.

[How92] Brian T. Howard. *Fixed Points and Extensionality in Typed Functional Programming Languages*. PhD thesis (Technical Report STAN-CS-92-1455), Computer Science Dept., Stanford Univ., CA, October 1992.

[Lei83] Daniel 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, 7–9 Nov 1983*, pages 460–469. IEEE Computer Society Press, Los Alamitos, CA, 1983.

[Lei90] Daniel Leivant. Contracting proofs to programs. In P. Odifreddi, editor, *Logic and Computer Science*, volume 31 of *APIC Studies in Data Processing*, pages 279–327. Academic Press, London, 1990.

[Men87] Nax Paul 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, 22–25 June 1987*, pages 30–36. IEEE Computer Society Press, Washington, DC, 1987.

[ML71] Per Martin-Löf. Hauptsatz for the intuitionistic theory of iterated inductive definitions. In J. E. Fenstad, editor, *Proceedings 2nd Scandinavian Logic Symp., Oslo, Norway, 18–20 June 1970*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 179–216. North-Holland, Amsterdam, 1971.

[NPS90] Bengt Nordström, Kent Petersson, and Jan M. Smith. *Programming in Martin-Löf's Type Theory*, volume 7 of *The Int'l Series of Monographs on Computer Science*. Clarendon Press, Oxford, 1990.

[Par90] Michel Parigot. On the representation of data in lambda-calculus. In E. Börger, H. Kleine Büning, and M. M. Richter, editors, *Proceedings 3rd Workshop on Computer Science Logic, CSL'89, Kaiserslautern, FRG, 2–6 Oct 1989*, volume 440 of *Lecture Notes in Computer Science*, pages 309–321. Springer-Verlag, Berlin, 1990.

[Rey74] John C. Reynolds. Towards a theory of type structure. In B. Robinet, editor, *Proceedings Programming Symp. (Colloque sur la Programmation), Paris, France, 9–11 Apr 1974*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer-Verlag, Berlin, 1974.

[SH84] Peter Schroeder-Heister. A natural extension of natural deduction. *Journal of Symbolic Logic*, 49(4):1284–1300, 1984.