Discrete Generalised Polynomial Functors

(Extended Abstract)

Marcelo Fiore*

Computer Laboratory, University of Cambridge
Marcelo.Fiore@cl.cam.ac.uk

Abstract. We study generalised polynomial functors between presheaf categories, developing their mathematical theory together with computational applications. The main theoretical contribution is the introduction of discrete generalised polynomial functors, a class that lies in between the classes of cocontinuous and finitary functors, and is closed under composition, sums, finite products, and differentiation. A variety of applications are given: to the theory of nominal algebraic effects; to the algebraic modelling of languages, and equational theories there of, with variable binding and polymorphism; and to the synthesis of dependent zippers.

Keywords: presheaf categories, Kan extensions, generalised logic, polynomial functors, equational systems, algebraic effects, abstract syntax, dependent programming

1 Introduction

The recurrent appearance of a structure in mathematical practice, as substantiated by interesting examples and applications, is a strong indicator of a worthwhile theory lurking in the background that merits development. This work is a direct outgrowth of this viewpoint as it arises from the interaction of two ubiquitous mathematical structures —presheaf categories and polynomial functors— in the context of computational applications. The paper thus contributes to the continued search for the foundational mathematical structures that underlie computer science.

Presheaf categories are functor categories of the form $\boldsymbol{\mathcal{S}et}^{\mathbb{C}}$ for $\mathbb{C}$ a small category and $\boldsymbol{\mathcal{S}et}$ the category of sets and functions. They enrich the universe of constant sets to one of variable sets [28]. The crucial import of this being that the mode of variation, as given by the parameter small category, translates to new, often surprising, internal structure in the presheaf category. As such, the use of presheaf categories in computer science applications has been prominent: *e.g.*, in programming language theory [33, 31, 15], lambda calculus [34, 25], domain theory [14, 16], concurrency theory [13, 35, 5], and type theory [23, 7].

Polynomial functors, *i.e.* polynomial constructions in or between categories, have also featured extensively; especially in the semantic theory of ADTs (see *e.g.* [37]). In modern theories of data structure there has been a need for the

* Partially supported by ERC ECSYM.

naive notion of polynomial functor as a sum-of-products construction to evolve to more sophisticated ones. In connection with type-theoretic investigations, this arose in the work of Gambino and Hyland [17] on the categorical study of W-types in Martin Löf Type Theory, and in the work on (indexed) containers of Abbott, Altenkirch, Ghani, and Morris [1, 4] on data structure in dependent programming. Type theoretically, the extension can be roughly understood as generalising from sums and products to Σ and Π types.

It is with the above level of generality that we are concerned here, in the particular context of presheaf categories. Our motivation stems from a variety of applications that require these new generalised polynomial functors. A case in point, treated in some detail in the paper, is the use of a class of discrete generalised polynomial functors as formal semantic counterparts of the informal vernacular rules that one encounters in presentations of syntactic structure. A main contribution of the paper shows the flexibility and expressiveness of our theory to the extent of being able to encompass languages, and equational theories thereof, with variable binding and polymorphism.

The work is presented in three parts, with Secs. 2 and 3 providing the necessary background, Secs. 4 and 5 developing the mathematical theory, and Secs. 6 and 7 dwelling on applications.

2 Generalised logic

This section recalls the basics of Lawvere's generalised logic [27]. Emphasis is placed on the categorical view of quantifiers as adjoints [26] that plays a central role in our development.

The basic categorical modelling of quantifiers as adjoints arises from the consideration of the contravariant powerset construction on sets, for which we will write $\wp$. Indeed, for all functions $f : X \to Y$, the monotone function $\wp f : \wp Y \to \wp X$ (where, for $T \in \wp Y$ and $x \in X$, $x \in \wp f(T) \Leftrightarrow fx \in T$) has both a left and a right adjoint, that roughly correspond to existential and universal quantification. More precisely, there are Galois connections

$$\exists_f \dashv \wp f \dashv \forall_f : \wp X \to \wp Y$$

given, for $S \in \wp X$ and $y \in Y$, by

$$y \in \exists_f(S) \iff \exists x \in X.\ fx = y \wedge x \in S \ , \tag{1}$$

$$y \in \forall_f(S) \iff \forall x \in X.\ y = fx \Rightarrow x \in S \ . \tag{2}$$

The categorical viewpoint of quantifiers generalises from sets to categories by considering the contravariant presheaf construction on small categories. For a small category $\mathbb{C}$, let $\mathcal{P}\mathbb{C}$ be the functor category $\boldsymbol{\mathcal{S}et}^{\mathbb{C}}$ of covariant presheaves and, for a functor $f : \mathbb{X} \to \mathbb{Y}$ between small categories, let $f^* : \mathcal{P}\mathbb{Y} \to \mathcal{P}\mathbb{X}$ be given by $P \mapsto P\,f$. A fundamental result states that there are adjunctions

$$f_! \dashv f^* \dashv f_* : \mathcal{P}\mathbb{X} \to \mathcal{P}\mathbb{Y} \ . \tag{3}$$

For $P : \mathbb{X} \to \boldsymbol{\mathcal{S}et}$, $f_!(P)$ is a left Kan extension of P along f; whilst f_*P is a right Kan extension of P along f. Importantly for our development, these can

be expressed by the following coend and end formulas

$$f_! P(y) \;=\; \int^{x \in \mathbb{X}} \; \mathbb{Y}(fx, y) \times Px \;\;, \tag{4}$$

$$f_* P(y) \;=\; \int_{x \in \mathbb{X}} \, [\, \mathbb{Y}(y, fx) \Rightarrow Px \,] \tag{5}$$

for $P \in \mathcal{P}\mathbb{X}$ and $y \in \mathbb{Y}$. (See *e.g.* [29].)

Example 2.1. Whenever necessary, we will identify a set (resp. a function) with its induced discrete category (resp. functor). For a function $f : X \to Y$, the formulas (4) and (5) for $f_!, f_* : \mathcal{P}X \to \mathcal{P}Y$ simplify to give, for $P \in \mathcal{P}X$ and $y \in Y$,

$$f_! P(y) \;\cong\; \coprod_{\{x \in X \,|\, fx = y\}} P(x) \;\;, \tag{6}$$

$$f_* P(y) \;\cong\; \prod_{\{x \in X \,|\, y = fx\}} P(x) \;\;. \tag{7}$$

Coends are quotients of sums under a compatibility equivalence relation and, as such, correspond to a generalised form of existential quantification; ends are restrictions of products under a parametricity condition and, dually, correspond to a generalised form of universal quantification (see *e.g.* [27]). In this respect, the formulas (6) and (7) are intensional generalisations of the formulas (1) and (2). Further in this vein, the reader is henceforth encouraged to use the following translation table

sum	disjunction
coend	existential quantification
product	conjunction
end	universal quantification
exponential	implication
hom	equality
presheaf application	predicate membership

to provide an intuitive logical reading of generalised categorical structures. In particular, note for instance that the categorical formulas (4) and (5) translate into the logical formulas (1) and (2).

3 Polynomial functors

The main objects of study and application in the paper are given by a general notion of polynomial functor between presheaf categories. This will be introduced in the next section. Here, as preliminary motivating background, we briefly recall an analogous notion of polynomial functor between slices of locally cartesian closed categories as it arose in the work of Gambino and Hyland [17] and of Abbott, Altenkirch, Ghani, and Morris [1, 4]. See [18] for further details.

In a type-theoretic setting, starting with the informal idea that a polynomial is a sum-of-products construction, one can consider them as constructions

$$X \;\mapsto\; \textstyle\sum_{j \in J} \prod_{i \in I(j)} X \tag{8}$$

arising from dependent pairs $(J : \mathrm{Set}\,,\, I : J \to \mathrm{Set})$, where each $j \in J$ can be naturally understood as a constructor with arities in $I(j)$. The categorical formalisation of this idea is founded on the view of Σ and Π types as adjoints.

Recall that these arise as follows

$$\Sigma_f \dashv \mathrm{R}_f \dashv \Pi_f : \mathscr{C}/A \to \mathscr{C}/B \tag{9}$$

where, for an object C of a category $\mathscr{C}$, we write $\mathscr{C}/C$ for the slice category of $\mathscr{C}$ over C and, for $f : A \to B$ in $\mathscr{C}$, we set $\Sigma_f : \mathscr{C}/A \to \mathscr{C}/B : (a : C \to A) \mapsto (f\,a : C \to B)$. Thus, (8) corresponds to

$$X \;\mapsto\; \Sigma_{J \to 1}\, \Pi_{I \to J}\, X \tag{10}$$

for a specification of constructors J and arities $I \to J$. More generally, we have the following definitions extending the construction (10) to slices.

Definition 3.1. *1. A* polynomial *in a category is a diagram* $A \leftarrow I \to J \to B$.

2. The polynomial functor *induced by a polynomial* $P = (A \xleftarrow{s} I \xrightarrow{f} J \xrightarrow{t} B)$ *in* $\mathscr{C}$ *is the composite*

$$\mathrm{F}_P = \Sigma_t\, \Pi_f\, \mathrm{R}_s : \mathscr{C}/A \to \mathscr{C}/B \ . \tag{11}$$

Example 3.1. To grasp the definition, it might be convenient to instantiate it in the category of sets. There, one sees that modulo the equivalence $\boldsymbol{\mathcal{S}et}/S \simeq \boldsymbol{\mathcal{S}et}^S$,

$$\mathrm{F}_{(A \xleftarrow{s} I \xrightarrow{f} J \xrightarrow{t} B)}\, \langle X_a \rangle_{a \in A} \;=\; \Big\langle \coprod_{j \in J_b} \prod_{i \in I_j} X_{si} \Big\rangle_{b \in B} \tag{12}$$

for $J_b = \{j \in J \mid tj = b\}$ and $I_j = \{i \in I \mid fi = j\}$. These are the normal functors of Girard [19].

4 Generalised polynomial functors

The notion, though not the terminology, of polynomial in a category appeared in the work of Tambara [36] as an abstract setting for inducing polynomial constructions in the presence of additive and multiplicative transfer structure. The transfer structure provided by Σ and Π types (9) gives rise to polynomial functors between slice categories (11). Our interest here is in the transfer structure provided by existential and universal quantification in generalised logic (3) as giving rise to generalised polynomial functors between presheaf categories (13).

Definition 4.1. *The* generalised polynomial functor *induced by a polynomial* $P = (\mathbb{A} \xleftarrow{s} \mathbb{I} \xrightarrow{f} \mathbb{J} \xrightarrow{t} \mathbb{B})$ *in the category of small categories and functors is the composite*

$$\mathrm{F}_P = t_!\, f_*\, s^* : \mathcal{P}\mathbb{A} \to \mathcal{P}\mathbb{B} \ . \tag{13}$$

That is,

$$\mathrm{F}_P\, X\, b = \textstyle\int^{j \in \mathbb{J}} \mathbb{B}(tj, b) \times \int_{i \in \mathbb{I}} \big[\, \mathbb{J}(j, fi) \Rightarrow X(si) \,\big] \ .$$

The closure under natural isomorphism of these functors yields the class of generalised polynomial functors.

A polynomial is said to represent a functor between presheaf categories whenever the latter is naturally isomorphic to the generalised polynomial functor induced by the former.

Example 4.1. 1. Write $\oint P$ for the category of elements of a presheaf $P \in \mathcal{P}\mathbb{C}$ and let π be the projection functor $\oint P \to \mathbb{C} : (c \in \mathbb{C}, p \in Pc) \mapsto c$.
Modulo the equivalence $\mathcal{P}(\mathbb{C})/P \simeq \mathcal{P}(\oint P)$, polynomial functors $\mathcal{P}(\mathbb{C})/P \to \mathcal{P}(\mathbb{C})/Q$ are subsumed by generalised polynomial functors $\mathcal{P}(\oint P) \to \mathcal{P}(\oint Q)$. The two notions coinciding for $\mathbb{C} = \mathbf{1}$.
It follows, for instance, that for every presheaf P, (i) the product endofunctor $(-) \times P$ and (ii) the exponential endofunctor $(-)^P$ are generalised polynomial.
2. Constant functors between presheaf categories are generalised polynomial.
3. Every cocontinuous functor between presheaf categories is generalised polynomial.

The definition of generalised polynomial functor extends to the multi-ary case as follows.

Definition 4.2. *For indexed families of small categories $\{\mathbb{A}_i\}_{i\in I}$ and $\{\mathbb{B}_j\}_{j\in J}$, a functor $\prod_{i\in I} \mathcal{P}\mathbb{A}_i \to \prod_{j\in J} \mathcal{P}\mathbb{B}_j$ is generalised polynomial iff so is the composite functor $\mathcal{P}\big(\coprod_{i\in I} \mathbb{A}_i\big) \cong \prod_{i\in I} \mathcal{P}\mathbb{A}_i \to \prod_{j\in J} \mathcal{P}\mathbb{B}_j \cong \mathcal{P}\big(\coprod_{j\in J} \mathbb{B}_j\big)$.*

The examples below play an important role in applications.

Example 4.2. For a small monoidal category $\mathbb{C}$, Day's monoidal-convolution tensor product [8] on $\mathcal{P}\mathbb{C}$ is a generalised polynomial functor $\mathcal{P}(\mathbb{C})^2 \to \mathcal{P}(\mathbb{C})$. Furthermore, for every $c \in \mathbb{C}$, monoidal-convolution exponentiation to the representable $\boldsymbol{y}c \in \mathcal{P}\mathbb{C}$ is a generalised polynomial endofunctor on $\mathcal{P}\mathbb{C}$.

Proposition 4.1. *The class of generalised polynomial functors is closed under sums and finite products.*

5 Discrete generalised polynomial functors

We introduce a simple subclass of generalised polynomial functors: the discrete ones. The results of this section show that they have a rich theory; those of Section 7 provide a range of sample applications.

Notation. For a set L and a category $\mathscr{C}$, let $L \cdot \mathscr{C} = \coprod_{\ell\in L} \mathscr{C}$ with ∇_L the codiagonal functor $[\mathrm{Id}]_{\ell\in L} : L \cdot \mathscr{C} \to \mathscr{C}$.

Definition 5.1. *The class of* discrete generalised polynomial functors *is induced by* discrete polynomials*, defined to be those of the form*

$$P = \Big(\mathbb{A} \xleftarrow{s} \coprod_{k\in K} L_k \cdot \mathbb{J}_k \xrightarrow{\coprod_{k\in K} \nabla_{L_k}} \coprod_{k\in K} \mathbb{J}_k \xrightarrow{t} \mathbb{B} \Big)$$

where L_k is finite for all $k \in K$.

One then has that: $\mathrm{F}_P X\, b \cong \coprod_{k\in K} \int^{j\in \mathbb{J}_k} \mathbb{B}(t\iota_k j, b) \times \prod_{\ell\in L_k} X(s\iota_k\iota_\ell j)$.

Remark. The class of discrete generalised polynomial functors is the closure under sums of the class of functors induced by *uninomials*, defined as diagrams

$$\mathbb{A} \longleftarrow L \cdot \mathbb{J} \xrightarrow{\nabla_L} \mathbb{J} \longrightarrow \mathbb{B}$$

with L finite.

Example 5.1. 1. For sets A and B, the discrete generalised polynomial functors $\mathcal{P}A \to \mathcal{P}B$ are as in (12) with I_j finite for all $j \in J$.
2. The generalised polynomial functors of Examples 4.1 ($1(i)$), (2), (3) and 4.2 are discrete.

The theory of discrete generalised polynomial functors will be developed elsewhere. An outline of main results and constructions on them follows.

Definition 5.2. *A functor is said to be inductive whenever it is finitary and preserves epimorphisms.*

Proposition 5.1. *Discrete generalised polynomial functors are inductive. Thus, they admit inductively constructed free algebras.*

Theorem 5.1. *The class of discrete generalised polynomial functors is closed under sums, finite products, and composition.*

Proposition 5.2. *The* 2*-category of small categories, discrete generalised polynomial functors, and natural transformations is cartesian. As such, and up to biequivalence, it subsumes the cartesian bicategory of small categories, profunctors, and natural transformations.*

Definition 5.3. *The* differential *of a uninomial* $M = (\mathbb{A} \xleftarrow{s} L \cdot \mathbb{J} \xrightarrow{\nabla_L} \mathbb{J} \xrightarrow{t} \mathbb{B})$ *is the discrete polynomial given by*

$$\partial M = \big(\mathbb{A} \xleftarrow{s'} \coprod_{(L_0,\ell_0)\in L'} L_0 \cdot \widetilde{\mathbb{J}} \xrightarrow{\coprod_{(L_0,\ell_0)\in L'} \nabla_{L_0}} L' \cdot \widetilde{\mathbb{J}} \xrightarrow{t'} \mathbb{A}^\circ \times \mathbb{B} \big)$$

for $L' = \big\{(L_0, \ell_0) \in \wp(L) \times L \mid L_0 \cap \{\ell_0\} = \emptyset, L_0 \cup \{\ell_0\} = L\big\}$*;* $\widetilde{\mathbb{J}} = \oint \hom_{\mathbb{J}}$ *(the twisted arrow category of* $\mathbb{J}$*);* $s' = [s\,(\imath_0 \cdot \pi_2)]_{(L_0,\ell_0)\in L'}$ *for* $\imath_0 : L_0 \hookrightarrow L$*; and* $t' = \big[\langle (s\,\iota_{\ell_0})^\circ\,\pi_1, t\,\pi_2\rangle\big]_{(L_0,\ell_o)\in L'}$*.*

For a discrete polynomial P *expressed as the sum of uninomials* $\coprod_{i\in I} M_i$*, we set* $\partial P = \coprod_{i\in I} \partial M_i$*.*

We have that:

$$\mathrm{F}_{\partial M}\, X\,(a,b) \;\cong\; \textstyle\int^{j\in\mathbb{J}} \mathbb{B}(tj,b) \times \coprod_{(L_0,\ell_0)\in L'} \mathbb{A}(a, s\iota_{\ell_0} j) \times \prod_{\ell\in L_0} X(s\iota_\ell j) \;\; .$$

6 Equational systems

We reformulate and extend the abstract theory of equational systems of Fiore and Hur [10] to more directly provide a framework for the specification of equational presentations. This we apply to generalised polynomial functors in the next section.

Notation. Writing Σ-Alg for the category of Σ-algebras of an endofunctor Σ on a category $\mathscr{C}$, let U denote the forgetful functor $\Sigma\text{-Alg} \to \mathscr{C} : (C, \Sigma C \xrightarrow{\gamma} C) \mapsto C$ and σ the natural transformation $\Sigma\,\mathrm{U} \Rightarrow \mathrm{U}$ with $\sigma_{(C,\gamma)} = \gamma$.

Definition 6.1. *1. An* inductive equational system *is a structure* $(\mathscr{C} : \Sigma \triangleright \Gamma \vdash \lambda \equiv \rho)$ *with* $\mathscr{C}$ *a cocomplete category,* Σ, Γ *inductive endofunctors on* $\mathscr{C}$*, and* λ, ρ *natural transformations* $\Gamma\,\mathrm{U} \Rightarrow \mathrm{U} : \Sigma\text{-Alg} \to \mathscr{C}$*.*

2. *For an inductive equational system* $S = (\mathscr{C} : \Sigma \triangleright \Gamma \vdash \lambda \equiv \rho)$*, the* category of S-algebras*,* S-Alg*, is the full subcategory of* Σ-Alg *determined by the* Σ*-algebras that equalise* λ *and* ρ*.*

Categories of algebras for inductive equational systems have the expected properties.

Theorem 6.1 ([10]). *For an inductive equational system* $S = (\mathscr{C} : \Sigma \triangleright \Gamma \vdash \lambda \equiv \rho)$*,* S-Alg *is reflective in* Σ-Alg *(with inductively constructed free* S*-algebras over* Σ*-algebras), the forgetful functor* S-Alg $\to \mathscr{C}$ *is monadic (with inductively constructed free algebras), and* S-Alg *is complete and cocomplete.*

It is important in applications to have a systematic framework for specifying inductive equational systems. To this end, we introduce rules of a somewhat syntactic character for deriving *natural terms*. These are judgements $F \vdash_n \varphi$ with $n \in \mathbb{N}$, F an inductive functor $\mathscr{C}^n \to \mathscr{C}$, and φ a natural transformation $F\,\mathrm{U}_n \Rightarrow \mathrm{U} : \Sigma\text{-Alg} \to \mathscr{C}$ for $\mathrm{U}_n = \langle \mathrm{U}, \ldots, \mathrm{U} \rangle : \Sigma\text{-Alg} \to \mathscr{C}^n$.

$$\frac{}{\Sigma \vdash_1 \sigma} \qquad \frac{}{\Pi_i \vdash_n \mathrm{id}}\ (1 \le i \le n)$$

$$\frac{F \vdash_n \varphi \quad F_i \vdash_m \varphi_i\ (1 \le i \le n)}{F \circ (F_1, \ldots, F_n) \vdash_m \varphi \circ F(\varphi_1, \ldots, \varphi_n)} \qquad \frac{\gamma : G \Rightarrow F \qquad F \vdash_n \varphi}{G \vdash_n \varphi \circ \gamma_{\mathrm{U}_n}}\ (G \text{ inductive})$$

Within this new framework, the specification of an equational presentation $\Gamma \vdash \lambda \equiv \rho$ is done by setting $\Gamma = \coprod_{i \in I} \Gamma_i$ and $\lambda = [\lambda_i]_{i \in I}$, $\rho = [\rho_i]_{i \in I}$ for natural terms $\Gamma_i \vdash_1 \lambda_i$ and $\Gamma_i \vdash_1 \rho_i$ for $i \in I$.

7 Applications

We give applications of generalised polynomial functors to nominal effects, abstract syntax, and dependent programming.

7.1 Nominal effects

The algebraic approach to computational effects of Plotkin and Power [32] regards the view of notions of computation as monads of Moggi [31] as derived from algebraic structure. This section illustrates the use of the theories of generalised polynomial functors and of equational systems in this context. We focus on nominal effects, explaining that the algebraic theory of the π-calculus of Stark [35] is an inductive equational system.

The algebraic theory of the π-calculus is built from the combination of three sub-theories: for non-determinism, communication, and name creation. Whilst each of the sub-theories is subject to algebraic laws of their own, the overall theory is obtained by further laws of interaction between them; see [35, Sec. 3] for details. The signature of π-algebras gives rise to an inductive generalised polynomial endofunctor on $\mathcal{P}\boldsymbol{I}$, for $\boldsymbol{I}$ a skeleton of the category of finite sets and injections. All the laws, that in [35] are either informally presented by syntactic equations or formally presented by commuting diagrams, can be seen to arise as

equational presentations of natural terms. The monadicity of π-calculus algebras is thus a consequence of the abstract theory of inductive equational systems.

7.2 Abstract syntax

An algebraic theory for polymorphic languages is developed within the framework of discrete generalised polynomial functors. This is done in three successive steps by considering first variable binding and capture-avoiding substitution, and then polymorphism.

Variable binding. We start by recasting the algebraic approach to variable binding of Fiore, Plotkin and Turi [15] in the setting of discrete polynomial functors. This we exemplify by means of the syntax of types in the polymorphic lambda calculus. Recall that this is given by the following informal vernacular rules (see *e.g.* [6]):

$$\text{(Var)}\ \frac{}{\Delta, X : * \vdash X : *} \quad (\Rightarrow)\ \frac{\Delta \vdash A : * \quad \Delta \vdash B : *}{\Delta \vdash A \Rightarrow B : *} \quad (\forall)\ \frac{\Delta, X : * \vdash A : *}{\Delta \vdash \forall X{:}*.\, A : *} \tag{14}$$

Our first task here will be to explain how these, and thereby the syntax they induce, are formalised as discrete generalised polynomial functors.

We need first to describe the mathematical structure of variable contexts. To this end, define a *scalar* in a category $\mathscr{C}$ to be an object $C \in \mathscr{C}$ equipped with a coproduct structure $\imath : Z \to (Z \bullet C) \leftarrow C : \jmath$ for all objects $Z \in \mathscr{C}$. In the vein of [9], for a set of sorts S, define the category of S-sorted variable contexts $\boldsymbol{C}[S]$ to be the free category with scalars $\langle s \rangle \in \boldsymbol{C}[S]$ for all sorts $s \in S$. The mathematical development does not depend on explicit descriptions of $\boldsymbol{C}[S]$, rather these provide different implementations. To fix ideas, however, the reader may take $\boldsymbol{C}[S]$ to be a skeleton of the comma category $\boldsymbol{\mathcal{F}in\mathcal{S}et} \downarrow S$ of S-indexed finite sets.

Writing $\boldsymbol{C}$ for the category of mono-sorted contexts $\boldsymbol{C}[\,\{*\}\,]$, which is a skeleton of $\boldsymbol{\mathcal{F}in\mathcal{S}et}$, each of the rules (14) directly translates as a uninomial as follows:

$$\begin{aligned}
&\text{(Var)} && \boldsymbol{C} \longleftarrow 0 \cdot \boldsymbol{C} \xrightarrow{\nabla_0} \boldsymbol{C} \xrightarrow{-\bullet\langle * \rangle} \boldsymbol{C} \\
&(\Rightarrow) && \boldsymbol{C} \xleftarrow{[\mathrm{id},\mathrm{id}]} 2 \cdot \boldsymbol{C} \xrightarrow{\nabla_2} \boldsymbol{C} \xrightarrow{\mathrm{id}} \boldsymbol{C} \\
&(\forall) && \boldsymbol{C} \xleftarrow{-\bullet\langle * \rangle} 1 \cdot \boldsymbol{C} \xrightarrow{\nabla_1} \boldsymbol{C} \xrightarrow{\mathrm{id}} \boldsymbol{C}
\end{aligned} \tag{15}$$

Note that the maps $\nabla_n : n \cdot \boldsymbol{C} \to \boldsymbol{C}$ correspond to the number of premises in the rule and that the associated maps $\boldsymbol{C} \leftarrow n \cdot \boldsymbol{C}$ describe the type of context needed for each of the premises. On the other hand, the maps $\boldsymbol{C} \to \boldsymbol{C}$ describe the type of context needed in the conclusion. Indeed, note that an algebra structure for the functor induced by a uninomial $\boldsymbol{C} \xleftarrow{s} n \cdot \boldsymbol{C} \xrightarrow{\nabla_n} \boldsymbol{C} \xrightarrow{t} \boldsymbol{C}$ on a presheaf $P \in \mathcal{P}\boldsymbol{C}$ corresponds to a family of functions $\prod_{i=1}^{n} P\big(s_i(\Delta)\big) \to P\big(t(\Delta)\big)$ natural for $\Delta \in \boldsymbol{C}$.

The sum of the uninomials (15) yields the discrete generalised polynomial

endofunctor Σ on $\mathcal{P}\boldsymbol{C}$ given by

$$\Sigma(P)(\Delta) = \boldsymbol{C}(\langle * \rangle, \Delta) + P(\Delta) \times P(\Delta) + P(\Delta \bullet \langle * \rangle) \ ,$$

whose initial algebra universally describes the syntax of polymorphic types in context up to α-equivalence.

Capture-avoiding substitution. Within this framework, one can algebraically account for the operation of capture-avoiding substitution. This is achieved by introducing a suitably axiomatised operator for substitution (see [15, Secs. 3 & 4]), that specifies its basic properties and renders it a derived operation. An outline, omitting details for brevity, follows.

The uninomial for the substitution operator is on the left below

$$(\varsigma) \quad \boldsymbol{C} \xleftarrow{[-\bullet\langle * \rangle, \mathrm{id}]} 2 \cdot \boldsymbol{C} \xrightarrow{\nabla_2} \boldsymbol{C} \xrightarrow{\mathrm{id}} \boldsymbol{C} \qquad \frac{\Delta, X : * \vdash A : * \quad \Delta \vdash B : *}{\Delta \vdash \varsigma(X :*. A \,, B) : *}$$

and, as such, arises from the informal vernacular rule on the right above. Consequently, one extends (Var), ($\Rightarrow$), ($\forall$) with (ς) to obtain the discrete generalised polynomial endofunctor Σ' on $\mathcal{P}\boldsymbol{C}$ given by

$$\Sigma'(P)(\Delta) = \Sigma(P)(\Delta) + P(\Delta \bullet \langle * \rangle) \times P(\Delta) \ ,$$

and then equips it with an equational presentation axiomatising the Σ-substitution algebras of [15, Sec. 4]. This axiomatisation can be equationally presented by means of natural terms as explained at the end of Sec. 6. Thereby, it determines an inductive generalised equational system by construction. Its initial algebra universally describes the syntax of polymorphic types in context up to α-equivalence equipped with the operation of single-variable capture-avoiding substitution. (This development applies to general second-order algebraic presentations, for which see [11, 12].)

Polymorphism. As we proceed to show, our framework is expressive enough to also allow for the modelling of polymorphic languages. This we exemplify with the term syntax of the polymorphic lambda calculus, for which the informal vernacular rules follow (see *e.g.* [6]):

$$(\mathrm{var}) \ \frac{\Delta \vdash A : *}{\Delta; \Gamma, x : A \vdash x : A}$$

$$(\mathrm{app}) \ \frac{\Delta; \Gamma \vdash t : A \Rightarrow B \quad \Delta; \Gamma \vdash u : A}{\Delta; \Gamma \vdash t(u) : B} \qquad (\mathrm{abs}) \ \frac{\Delta; \Gamma, x : A \vdash t : B}{\Delta; \Gamma \vdash \lambda x{:}A.\, t : \ A \Rightarrow B} \tag{16}$$

$$(\mathrm{App}) \ \frac{\Delta; \Gamma \vdash t : \ \forall X :*. A \quad \Delta \vdash B}{\Delta; \Gamma \vdash t[B] : \ A[B/X]} \qquad (\mathrm{Abs}) \ \frac{\Delta, X : * \,; \Gamma \vdash t : A}{\Delta; \Gamma \vdash \Lambda X :*. t : \ \forall X :*. A}$$

(In the rule (Abs) the type variable X is not free in any type of the context Γ.)

Following Hamana [21], we set up the algebraic framework on presheaves over type and term variable contexts obtained from indexed categories by means of the Grothendieck construction [20]. Recall that from $\mathcal{K} : \mathscr{K} \to \boldsymbol{CAT}$ this yields the category, which we will denote $\int^{K \in \mathscr{K}} \mathcal{K}(K)$, with objects (K, k) for $K \in \mathscr{K}$

and $k \in \mathcal{K}(K)$, and morphisms $(f, g) : (K, k) \to (K', k')$ for $f : K \to K'$ in $\mathscr{K}$ and $g : \mathcal{K}(f)(k) \to k'$ in $\mathcal{K}(K')$.

Let $[\nu, \Rightarrow, \forall, \varsigma] : \Sigma'(T) \to T$ in $\mathcal{P}\boldsymbol{C}$ be an initial algebra for the equational system of polymorphic types with substitution. For $\boldsymbol{G} = \oint^{\Delta \in \boldsymbol{C}} \boldsymbol{C}[T\Delta] \times T(\Delta)$, the rules (16) directly translate into uninomials as follows:

$$\text{(var)} \quad \boldsymbol{G} \xleftarrow{} 0 \cdot \boldsymbol{G} \xrightarrow{\nabla_0} \boldsymbol{G} \xrightarrow{\oint \langle \bullet, \mathrm{id} \rangle} \boldsymbol{G}$$
$$\text{where } \bullet_S : \boldsymbol{C}[S] \times S \to \boldsymbol{C}[S] : \Gamma, s \mapsto \Gamma \bullet \langle s \rangle$$

$$\text{(app)} \quad \boldsymbol{G} \xleftarrow{[\oint(\mathrm{id} \times \Rightarrow), \oint(\mathrm{id} \times \pi_1)]} 2 \cdot \boldsymbol{G}_{\mathrm{a}} \xrightarrow{\nabla_2} \boldsymbol{G}_{\mathrm{a}} \xrightarrow{\oint(\mathrm{id} \times \pi_2)} \boldsymbol{G}$$
$$\text{where } \boldsymbol{G}_{\mathrm{a}} = \oint^{\Delta \in \boldsymbol{C}} \boldsymbol{C}[T\Delta] \times T(\Delta) \times T(\Delta)$$

$$\text{(abs)} \quad \boldsymbol{G} \xleftarrow{\oint \langle \bullet, \mathrm{id} \rangle} 1 \cdot \boldsymbol{G}_{\mathrm{a}} \xrightarrow{\nabla_1} \boldsymbol{G}_{\mathrm{a}} \xrightarrow{\oint(\mathrm{id} \times \Rightarrow)} \boldsymbol{G}$$

$$\text{(App)} \quad \boldsymbol{G} \xleftarrow{[\oint(\mathrm{id} \times (\forall\, \pi_1)), \oint(\mathrm{id} \times \pi_2)]} 2 \cdot \boldsymbol{G}_{\mathrm{App}} \xrightarrow{\nabla_2} \boldsymbol{G}_{\mathrm{App}} \xrightarrow{\oint(\mathrm{id} \times \varsigma)} \boldsymbol{G}$$
$$\text{where } \boldsymbol{G}_{\mathrm{App}} = \oint^{\Delta \in \boldsymbol{C}} \boldsymbol{C}[T\Delta] \times T(\Delta \bullet \langle * \rangle) \times T(\Delta)$$

$$\text{(Abs)} \quad \boldsymbol{G} \xleftarrow{\imath^{\#}} 1 \cdot \boldsymbol{G}_{\mathrm{Abs}} \xrightarrow{\nabla_1} \boldsymbol{G}_{\mathrm{Abs}} \xrightarrow{\oint(\mathrm{id} \times \forall)} \boldsymbol{G}$$
$$\text{where } \boldsymbol{G}_{\mathrm{Abs}} = \oint^{\Delta \in \boldsymbol{C}} \boldsymbol{C}[T\Delta] \times T(\Delta \bullet \langle * \rangle)$$
$$\text{and } \imath^{\#}(\Delta; \Gamma, A) = \big(\Delta \bullet \langle * \rangle; \boldsymbol{C}[T\imath](\Gamma), A\big)$$

Note again that the maps $\nabla_n : n \cdot \mathbb{G} \to \mathbb{G}$ correspond to the number of premises in the rules and that the associated maps $\boldsymbol{G} \leftarrow n \cdot \mathbb{G}$ describe the type of context and types needed for each premise; whilst the maps $\mathbb{G} \to \boldsymbol{G}$ describe the context and type needed in the conclusion. One can therefore see rules as syntactic specifications of uninomials.

The sum of the above uninomials yields a discrete generalised polynomial endofunctor on $\mathcal{P}\boldsymbol{G}$ whose initial algebra universally describes the syntax of typed polymorphic terms in context up to α-equivalence. One can even go further and equationally axiomatise the operations of type-in-term and term-in-term substitution leading to a purely algebraic notion of model for polymorphic simple type theories. Details will appear elsewhere.

7.3 Dependent programming

McBride [30] observed that the formal derivative of the type constructor of an ADT yields the type constructor for the one-hole contexts of the ADT, and use this as the basis for a generic framework for developing *zippers* for data-structure navigation as originally conceived by Huet [24]. In dependent programming, these ideas were revisited by Abbott, Altenkirch, Ghani, and McBride [2, 3] for containers [1]. Following this line, Hamana and Fiore [22, Sec. 5] considered partial derivation for set-theoretic indexed containers (*i.e.* polynomial functors between slice categories of $\boldsymbol{\mathcal{S}et}$) and applied the construction to synthesise dependent zippers for GADTs. This section shows that the more general notion of differential for discrete generalised polynomial functors introduced in this pa-

per (Definition 5.3) leads to more refined zippers, with staging information. For brevity, we only consider an example.

Dependent zippers. The differentials of the uninomials (15) are as follows

$$(\mathrm{Var}') \quad \boldsymbol{C} \leftarrow \mathbf{0} \rightarrow \mathbf{0} \rightarrow \boldsymbol{C}^\circ \times \boldsymbol{C}$$

$$(\Rightarrow') \quad \boldsymbol{C} \xleftarrow{[\pi_2,\pi_2]} 2 \cdot \widetilde{\boldsymbol{C}} \xrightarrow{\mathrm{id}} 2 \cdot \widetilde{\boldsymbol{C}} \xrightarrow{[\pi,\pi]} \boldsymbol{C}^\circ \times \boldsymbol{C}$$

$$(\forall') \quad \boldsymbol{C} \leftarrow \mathbf{0} \rightarrow \widetilde{\boldsymbol{C}} \xrightarrow{\langle (-\bullet\langle * \rangle)\, \pi_1\, ,\, \pi_2 \rangle} \boldsymbol{C}^\circ \times \boldsymbol{C}$$

from which it follows that the induced discrete generalised polynomial functor $\partial\Sigma : \mathcal{P}\boldsymbol{C} \to \mathcal{P}(\boldsymbol{C}^\circ \times \boldsymbol{C})$ is given by

$$\partial\Sigma(P)(\Delta', \Delta) = 2 \times \boldsymbol{C}(\Delta', \Delta) \times P(\Delta) + \boldsymbol{C}\big(\Delta', \Delta \bullet \langle * \rangle\big) \ .$$

For an algebra $[\nu, \Rightarrow, \forall] : \Sigma S \to S$, the construction $\partial\Sigma(S)(\Delta', \Delta)$ is that of the one-hole contexts at stage Δ for elements of S at stage Δ'. Indeed, we have a map for plugging components

$$\mathsf{plug} : S(\Delta') \times \partial\Sigma(S)(\Delta', \Delta) \ \to \ S(\Delta)$$

with $\mathsf{plug}\big(A, (0, f, B)\big) = \ \big(B \Rightarrow Sf(A)\big)$, $\mathsf{plug}\big(A, (1, f, B)\big) = \ (Sf(A) \Rightarrow B)$, and $\mathsf{plug}(A, f) = \forall\big(Sf(A)\big)$; and maps for focussing on sub-components

$$\mathsf{focus}_0, \mathsf{focus}_1 : S(\Delta') \times S(\Delta') \times \boldsymbol{C}(\Delta', \Delta) \longrightarrow S(\Delta') \times \partial\Sigma(S)(\Delta', \Delta)$$

$$\mathsf{focus} : \quad S(\Delta' \bullet \langle * \rangle) \times \boldsymbol{C}(\Delta', \Delta) \quad \longrightarrow S(\Delta' \bullet \langle * \rangle) \times \partial\Sigma(S)(\Delta' \bullet \langle * \rangle, \Delta)$$

with $\mathsf{focus}_0\big((A, B), f\big) = \big(A, \big(1, f, Sf(B)\big)\big)$, $\mathsf{focus}_1\big((A, B), f\big) = \big(B, \big(0, f, Sf(A)\big)\big)$, and $\mathsf{focus}(A, f) = (A, f \bullet \langle * \rangle)$.

References

1. Abbott, M., Altenkirch, T., Ghani, N.: Categories of containers. In: FoSSaCS'03. LNCS, vol. 2620, pp. 23–38 (2003)
2. Abbott, M., Altenkirch, T., Ghani, N., McBride, C.: Derivatives of containers. In: TLCA'03. LNCS, vol. 2701, pp. 16–30 (2003)
3. Abbott, M., Altenkirch, T., Ghani, N., McBride, C.: ∂ is for data - differentiating data structures. Fundamenta Informaticae 65, pp. 1–28 (2005)
4. Altenkirch, T., Morris, P.: Indexed containers. In: LICS'09. pp. 277–285 (2009)
5. Cattani, G.L., Winskel, G.: Profunctors, open maps, and bisimulation. Mathematical Structures in Computer Science 15, pp. 553–614 (2005)
6. Crole, R.: Categories for Types. Cambridge University Press (1994)
7. Danvy, O., Dybjer, P. (eds.): Preliminary Proceedings of the APPSEM Workshop on Normalisation by Evaluation (1998)
8. Day, B.: On closed categories of functors. In: Reports of the Midwest Category Seminar IV, LNM, vol. 137, pp. 1–38. Springer-Verlag (1970)
9. Fiore, M.: Semantic analysis of normalisation by evaluation for typed lambda calculus. In: PPDP'02. pp. 26–37 (2002)
10. Fiore, M., Hur, C.-K.: On the construction of free algebras for equational systems. Theoretical Computer Science 410, pp. 1704–1729 (2008)

11. Fiore, M., Hur, C.-K.: Second-order equational logic. In: CSL'10. LNCS, vol. 6247, pp. 320–335 (2010)
12. Fiore, M., Mahmoud, O.: Second-order algebraic theories. In: MFCS'10. LNCS, vol. 6281, pp. 368–380 (2010)
13. Fiore, M., Moggi, E., Sangiorgi, D.: A fully-abstract model for the pi-calculus. Information and Computation 179, pp. 76–117 (2002)
14. Fiore, M., Plotkin, G., Power, A.J.: Complete cuboidal sets in Axiomatic Domain Theory. In: LICS'97. pp. 268–279 (1997)
15. Fiore, M., Plotkin, G., Turi, D.: Abstract syntax and variable binding. In: LICS'99. pp. 193–202 (1999)
16. Fiore, M., Rosolini, G.: Domains in $\mathcal{H}$. Theoretical Computer Science 264, pp. 171–193 (2001)
17. Gambino, N., Hyland, M.: Wellfounded trees and dependent polynomial functors. In: TYPES 2003. LNCS, vol. 3085, pp. 210–225 (2004)
18. Gambino, N., Kock, J.: Polynomial functors and polynomial monads. ArXiv:0906:4931 (2010)
19. Girard, J.-Y.: Normal functors, power series and λ-calculus. Annals of Pure and Applied Logic 37, 129–177 (1988)
20. Grothendieck, A.: Catégories fibrées et descente. In: SGA1, LNM, vol. 224. Springer-Verlag (1971)
21. Hamana, M.: Polymorphic abstract syntax via Grothendieck construction. In: FoSSaCS'11. LNCS, vol. 6604, pp. 381–395 (2011)
22. Hamana, M., Fiore, M.: A foundation for GADTs and Inductive Families: Dependent polynomial functor approach. In: WGP'11. pp. 59–70 (2011)
23. Hofmann, M.: Syntax and semantics of dependent types. In: Semantics and Logics of Computation, pp. 79–130. Cambridge University Press (1997)
24. Huet, G.: The zipper. Journal of Functional Programming 7, pp. 549–554 (1997)
25. Jung, A., Tiuryn, J.: A new characterization of lambda definability. In: TLCA'93. pp. 245–257 (1993)
26. Lawvere, F.W.: Adjointness in foundations. Dialectica 23 (1969)
27. Lawvere, F.W.: Metric spaces, generalized logic, and closed categories. Rend. del Sem. Mat. e Fis. di Milano 43, pp. 135–166 (1973)
28. Lawvere, F.W.: Continuously variable sets: algebraic geometry = geometric logic. In: Proc. Logic Colloq. '73. pp. 135–156 (1975)
29. Mac Lane, S.: Categories for the working mathematician. Springer-Verlag (1971)
30. McBride, C.: The derivative of a regular type is its type of one-hole contexts. Unpublished (2001)
31. Moggi, E.: Notions of computation and monads. Information and Computation 93, pp. 55–92 (1991)
32. Plotkin, G., Power, A.J.: Notions of computation determine monads. In: FoSSaCS'02. LNCS, vol. 2303, pp. 342–356 (2002)
33. Reynolds, J.: Using functor categories to generate intermediate code. In: POPL'95. pp. 25–36 (1995)
34. Scott, D.: Relating theories of the λ-calculus. In: To H.B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalisms. Academic Press (1980)
35. Stark, I.: Free-algebra models for the π-calculus. In: FoSSaCS'05. LNCS, vol. 3441, pp. 155–169 (2005)
36. Tambara, D.: On multiplicative transfer. Comm. Alg. 21, pp. 1393–1420 (1993)
37. Wagner, E.: Algebraic specifications: some old history and new thoughts. Nordic J. of Computing 9, pp. 373–404 (2002)