Dependent Polynomial Functors for Inductive Families

Makoto Hamana
Gunma University, Japan

Joint work with
Marcelo Fiore, University of Cambridge

AIM-DTP joint day
14 September, 2011.

This Work

- ▷ Dependent polynomial functor representation of Inductive Families

Background

N. Gambino and M. Hyland,
Wellfounded Trees and Dependent Polynomial Functors, TYPES'03.

J. Kock, *Notes on Polynomial functors*,
manuscript, 412 pages, Version 2009-08-05.

This Work

▷ Dependent polynomial functor representation of Inductive Families

Background

N. Gambino and M. Hyland,
Wellfounded Trees and Dependent Polynomial Functors, TYPES'03.

J. Kock, *Notes on Polynomial functors*,
manuscript, 412 pages, Version 2009-08-05.

Problem ▶ Not clear what are dependent polynomials for IFs in these papers

.

Aim ▶ Recipes for giving dependent polynomials for IFs

ADTs and Programming Techniques

ADTs have a solid foundation: ordinary polynomial functors

It is the basis for various programming techniques:

- Fold and fusion techniques
  [Meijer et al.'91][Launchbury,Sheard'95][Takano,Meijer'95]
  [Hu et al.'96][Katsumata,Nisimura'08][Ghani et al.'05][Hinze'10]
- Polytypic programming [Jansson,Jeuring'97]
- Generic Haskell [Hinze,Jeuring'03]
- Program reasoning [Danielsson et al.'06]
- Generic zippers [McBride'01][Morihata et al.'09]

- Polynomial functor representation is useful

This Talk

To extend this story to IFs

[I] Polynomial representation of Inductive Families
that generates dependent polynomial functors

[II] Zippers for Inductive Families

ADTs ⟶ Inductive Families

polynomial ft.s dependent polynomials & ft.s

zippers, etc.

Review: Meaning of Algebraic Datatypes

```
data List = Nil
          | Cons Int List
```

- Semantics = the initial $\boldsymbol{F}$-algebra $\boldsymbol{\alpha} : \boldsymbol{FA} \xrightarrow{\cong} \boldsymbol{A}$

$$\boldsymbol{F} : \mathbf{Set} \to \mathbf{Set}$$

$$\boldsymbol{F}(\boldsymbol{X}) = \mathbf{1} + \mathbb{Z} \times \boldsymbol{X}$$

- **Point:** polynomial functor $\boldsymbol{F}$ characterises `List`
- How can we extend this to IFs?

I. How to Model Inductive Families

Inductive Families

- Bounded natural numbers

```
data Nat : Set where
  Z : Nat
  S : Nat -> Nat


data Fin : Nat -> Set where
  Zero : {n : Nat} -> Fin (S n)
  Succ : {n : Nat} -> Fin n -> Fin (S n)
```

- `Fin` n types natural numbers below n
- `Nat` is characterised as the initial $\mathbf{1} + (-)$-algebra $\mathbb{N} \in \mathbf{Set}$

Modelling Fin

```
data Fin : Nat -> Set where
  Zero : {n : Nat} -> Fin (S n)
  Succ : {n : Nat} -> Fin n -> Fin (S n)
```

▷ What is the polynomial functor for `Fin`?

Modelling Fin

```
data Fin : Nat -> Set where
  Zero : {n : Nat} -> Fin (S n)
  Succ : {n : Nat} -> Fin n -> Fin (S n)
```

- What is the polynomial functor for `Fin`?
- Answer:

$$\boldsymbol{F_{Fin}} : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}^{\mathbb{N}}$$

$$\boldsymbol{F_{Fin}}(\boldsymbol{X})(\mathtt{S}\ \boldsymbol{a}) = \mathbf{1} + \boldsymbol{X}(\boldsymbol{a})$$

$$\boldsymbol{F_{Fin}}(\boldsymbol{X})(\mathtt{Z}) \quad = \varnothing$$

$[\![\mathtt{Fin}]\!] =$ the initial $\boldsymbol{F_{Fin}}$-algebra Fin $\in \mathbf{Set}^{\mathbb{N}}$

- How to derive? What are "polynomials"?
- Dependent polynomials

The Universe of Discourse

ADTs		Inductive Famines
		Fam(Set)
Set	$\rightsquigarrow$	$\mathbf{Set}^I$ for every $I \in \mathbf{Set}$
the category of sets		the categories of indexed sets
polynomial ft.		dependent polynomial ft.

Category of Indexed Sets

▷ Category $\mathbf{Set}^{I}$ for an arbitrary set I

- Objects: $A : I \longrightarrow \mathbf{Set}$
  i.e. I-indexed sets $\{A(i) \mid i \in I\}$
- Arrows: I-indexed functions $f : A \longrightarrow B$,
  i.e. a family of functions $(f(i) : A(i) \longrightarrow B(i) \mid i \in I)$

Category of Indexed Sets

- Category $\mathbf{Set}^{I}$ for an arbitrary set I
  - Objects: $A : I \longrightarrow \mathbf{Set}$
    i.e. I-indexed sets $\{A(i) \mid i \in I\}$
  - Arrows: I-indexed functions $f : A \longrightarrow B$,
    i.e. a family of functions $(f(i) : A(i) \longrightarrow B(i) \mid i \in I)$

- Important functors: given a function $h : I \longrightarrow J$,

$$\mathbf{Set}^{I} \underset{\Pi_h}{\overset{\Sigma_h}{\leftarrow}} \mathbf{Set}^{J} \qquad \text{with } h^{*} \text{ between}$$

$$\Sigma_h(A)(j) = \sum_{\substack{i \in I \\ j \equiv h(i)}} A(i) \qquad h^{*}(A)(j) = A(h(j))$$

$$\Pi_h(A)(j) = \prod_{\substack{i \in I \\ j \equiv h(i)}} A(i)$$

Lawvere's quantifiers by adjointness [1969]

Dependent Polynomials [Gambino-Hyland'03]

▷ **Def.** A (dependent) polynomial P is
a triple $P = (d, p, c)$ of functions between sets

$$I \xleftarrow{d} E \xrightarrow{p} B \xrightarrow{c} J\ .$$

Dependent Polynomials [Gambino-Hyland'03]

▷ **Def.** A (dependent) polynomial P is
a triple $P = (d, p, c)$ of functions between sets

$$I \xleftarrow{d} E \xrightarrow{p} B \xrightarrow{c} J\,.$$

▷ Then, one obtains a functor

$$\mathbf{Set}^I \xrightarrow{d^*} \mathbf{Set}^E \xrightarrow{\Pi_p} \mathbf{Set}^B \xrightarrow{\Sigma_c} \mathbf{Set}^J$$

Dependent Polynomials [Gambino-Hyland'03]

- **Def.** A (dependent) polynomial P is
  a triple $P = (d, p, c)$ of functions between sets

$$I \xleftarrow{d} E \xrightarrow{p} B \xrightarrow{c} J\,.$$

- Then, one obtains a functor

$$\mathbf{Set}^I \xrightarrow{d^*} \mathbf{Set}^E \xrightarrow{\Pi_p} \mathbf{Set}^B \xrightarrow{\Sigma_c} \mathbf{Set}^J$$

- NB. the original version uses a lccc and slices

Dependent Polynomials [Gambino-Hyland'03]

▷ **Def.** The dependent polynomial functor F_P associated to a dependent polynomial $P = (d, p, c)$ is defined by

$$F_P : \mathbf{Set}^I \longrightarrow \mathbf{Set}^J$$

$$F_P(X) \stackrel{def}{=} \Sigma_c(\Pi_p(d^*(X))).$$

▷ i.e.

$$F_P(X)(j) = \sum_{\substack{b \in B \\ j \equiv c(b)}} \prod_{\substack{e \in E \\ b \equiv p(e)}} X(d(e))$$

Modelling Fin

```
data Fin : Nat -> Set where
  Zero : {n : Nat} -> Fin (S n)
  Succ : {n : Nat} -> Fin n -> Fin (S n)
```

▷ Model constructors as polynomials

$$\boldsymbol{Zero} = \quad \mathbb{N} \xleftarrow{!} \varnothing \xrightarrow{!} \mathbb{N} \xrightarrow{\mathtt{S}} \mathbb{N}.$$

$$\boldsymbol{Succ} = \quad \mathbb{N} \xleftarrow{\mathbf{id}} \mathbb{N} \xrightarrow{\mathbf{id}} \mathbb{N} \xrightarrow{\mathtt{S}} \mathbb{N}.$$

Modelling Fin

```
data Fin : Nat -> Set where
  Zero : {n : Nat} -> Fin (S n)
  Succ : {n : Nat} -> Fin n -> Fin (S n)
```

- Model constructors as polynomials

$$\boldsymbol{Zero} = \quad \mathbb{N} \xleftarrow{!} \varnothing \xrightarrow{!} \mathbb{N} \xrightarrow{\mathrm{S}} \mathbb{N}.$$

$$\boldsymbol{Succ} = \quad \mathbb{N} \xleftarrow{\mathrm{id}} \mathbb{N} \xrightarrow{\mathrm{id}} \mathbb{N} \xrightarrow{\mathrm{S}} \mathbb{N}.$$

- The sum of polynomials is again a polynomial

$$\boldsymbol{Fin} \stackrel{def}{=} \boldsymbol{Zero} + \boldsymbol{Succ}$$

Modelling Fin

```
data Fin : Nat -> Set where
  Zero : {n : Nat} -> Fin (S n)
  Succ : {n : Nat} -> Fin n -> Fin (S n)
```

- Sum of polynomials is again a polynomial

$$\boldsymbol{Fin} \stackrel{def}{=} \boldsymbol{Zero} + \boldsymbol{Succ}$$

- Dependent polynomial functor $\boldsymbol{F_{Fin}} : \mathbf{Set}^{\mathbb{N}} \longrightarrow \mathbf{Set}^{\mathbb{N}}$

$$\begin{aligned}
\boldsymbol{F_{Fin}}(\boldsymbol{X})(\boldsymbol{n}) &= \boldsymbol{F_{Zero+Succ}}(\boldsymbol{X})(\boldsymbol{n}) \\
&= \boldsymbol{F_{Zero}}(\boldsymbol{X})(\boldsymbol{n}) + \boldsymbol{F_{Succ}}(\boldsymbol{X})(\boldsymbol{n}) \\
&= \boldsymbol{\Sigma_{\mathtt{S}}\Pi_{!}!^{*}}(\boldsymbol{X})(\boldsymbol{n}) + \boldsymbol{\Sigma_{\mathtt{S}}\Pi_{\mathtt{id}}\mathbf{id}^{*}}(\boldsymbol{X})(\boldsymbol{n}) \\
&= \sum_{\substack{\boldsymbol{a}\in\mathbb{N} \\ \boldsymbol{n}\equiv_{\mathtt{S}}\boldsymbol{a}}} (\mathbf{1} + \boldsymbol{X}(\boldsymbol{a})))
\end{aligned}$$

Modelling Fin

- Dependent polynomial functor

$$\boldsymbol{F_{Fin}(X)(n)} = \sum_{\substack{\boldsymbol{a}\in\mathbb{N} \\ \boldsymbol{n}\equiv_{\mathbb{S}}\boldsymbol{a}}} (1 + \boldsymbol{X(a)}))$$

  is equivalent to the definition by pattern-matching

$$\boldsymbol{F_{Fin}} : \mathbf{Set}^{\mathbb{N}} \to \mathbf{Set}^{\mathbb{N}}$$
$$\boldsymbol{F_{Fin}(X)}(\mathbb{S}\ \boldsymbol{a}) = 1 + \boldsymbol{X(a)}$$
$$\boldsymbol{F_{Fin}(X)}(\mathbb{Z}) = \varnothing$$

- Initial algebra is constructed by repeated applications of $\boldsymbol{F_{Fin}}$

Thm. [Gambino-Hyland'03]
Every dependent polynomial functor has an initial algebra.

Example: Fin

(1)
```
data Fin : Nat -> Set where
     Zero : {n : Nat} -> Fin (S n)
     Succ : {n : Nat} -> Fin n -> Fin (S n)
```

(2) Polynomial

$$\boldsymbol{Zero} = \quad \mathbb{N} \xleftarrow{!} \varnothing \xrightarrow{!} \mathbb{N} \xrightarrow{\mathtt{S}} \mathbb{N}.$$

$$\boldsymbol{Succ} = \quad \mathbb{N} \xleftarrow{\mathbf{id}} \mathbb{N} \xrightarrow{\mathbf{id}} \mathbb{N} \xrightarrow{\mathtt{S}} \mathbb{N}.$$

(3) Dependent polynomial functor $\boldsymbol{F_{Fin}} : \mathbf{Set}^{\mathbb{N}} \longrightarrow \mathbf{Set}^{\mathbb{N}}$

$$\boldsymbol{F_{Fin}(X)(n)} = \sum_{\substack{a \in \mathbb{N} \\ n \equiv \mathtt{S}\ a}} (1 + \boldsymbol{X(a)}))$$

General Case: Simple Inductive Family

$$\texttt{data}\ D : I \longrightarrow \texttt{Set}\ \texttt{where}$$
$$K : (j : J) \longrightarrow D(d_1[j]) \longrightarrow \cdots \longrightarrow D(d_k[j]) \longrightarrow D(c[j])$$

▷ Polynomial (functions $d_i : J \longrightarrow I, \quad c : J \longrightarrow I$)

$$I \xleftarrow{[d_1,\ldots,d_k]} kJ \xrightarrow{\nabla_k} J \xrightarrow{c} I$$

- "Co-diagonal" function $\nabla_k = [\mathrm{id}_J, \ldots, \mathrm{id}_J] : kJ \longrightarrow J$

▷ Dependent polynomial functor $F_D : \mathbf{Set}^I \longrightarrow \mathbf{Set}^I$

$$F_D X(m) = \sum_{\substack{j \in J \\ m \equiv c(j)}} X(d_1(j)) \times \cdots \times X(d_k(j))$$

More General: Basic Inductive Family

(1)

$$\begin{array}{l}\texttt{data } D : I \longrightarrow \texttt{Set where} \\ \quad K : (j : J) \longrightarrow \\ \qquad (e : E[j]) \longrightarrow ((s : S_1[j,e]) \longrightarrow D(d_1[j,e,s])) \longrightarrow \cdots \\ \qquad\qquad \longrightarrow ((s : S_k[j,e]) \longrightarrow D(d_k[j,e,s])) \longrightarrow D(c[j])\end{array}$$

(2) Polynomial $(d = [d_1, \ldots, d_k])$

$$I \xleftarrow{d} \{S_1 + \cdots + S_k\} \xrightarrow{\pi} \{E\} \xrightarrow{c\pi} I$$

(3) Dependent polynomial functor

$$F_D X(u) \cong \sum_{\substack{j \in J,\, e \in E(j) \\ u \equiv c(j)}} \prod_{s \in S_1(j,e)} X(d_1(j,e,s)) \times \cdots \times \prod_{s \in S_k(j,e)} X(d_k(j,e,s))$$

II. Application: Zippers

Zippers

- G. Huet, *Functional Pearl: The Zipper*,
  Journal of Functional Programming, 1997.
- A data structure for navigating a tree freely
  (not only decomposition)
- A zipper = current forcus & lists of depth-one contexts
- Generic way to give the type of depth-one contexts

Zippers

- G. Huet, *Functional Pearl: The Zipper*,
  Journal of Functional Programming, 1997.
- A data structure for navigating a tree freely
  (not only decomposition)
- A zipper = current forcus & lists of depth-one contexts
- Generic way to give the type of depth-one contexts
- McBride's finding
  - Binary trees $\boldsymbol{F}(\boldsymbol{X}) = \mathbf{1} + \boldsymbol{X} \times \boldsymbol{X}$
  - Depth-one contexts $\boldsymbol{F}'(\boldsymbol{X}) = \boldsymbol{X} + \boldsymbol{X}$ – differentiation
- Only for ADTs and polynomial functors
- Extension to IFs and dependent polynomial functors

Differentiation

- Dependent polynomial functor $F : \mathbf{Set}^I \longrightarrow \mathbf{Set}^J$

$$F(X)(j) = \sum_{\substack{b \in B \\ j \equiv c(b)}} \prod_{e \in E_b} X(d(e))$$

Differentiation

- Dependent polynomial functor $F : \mathbf{Set}^I \longrightarrow \mathbf{Set}^J$

$$F(X)(j) = \sum_{\substack{b \in B \\ j \equiv c(b)}} \prod_{e \in E_b} X(d(e))$$

- Partial derivative of F with respect to $i \in I$

$$\partial_i F : \mathbf{Set}^I \longrightarrow \mathbf{Set}^J$$

$$\partial_i F(X)(j) = \sum_{\substack{e \in E \\ j \equiv c(b)}} \sum_{\substack{\ell \in E_b \\ i \equiv d(\ell)}} \prod_{e \in E_b \setminus \{\ell\}} X(d(e))$$

  - Derived from differentiation of generalised species [Fiore FOSSACS'05, etc.]

Differentiation

- Dependent polynomial functor $F : \mathbf{Set}^I \to \mathbf{Set}^J$

$$F(X)(j) = \sum_{\substack{b \in B \\ j \equiv c(b)}} \prod_{e \in E_b} X(d(e))$$

- Partial derivative of F with respect to $i \in I$

$$\partial_i F : \mathbf{Set}^I \to \mathbf{Set}^J$$

$$\partial_i F(X)(j) = \sum_{\substack{e \in E \\ j \equiv c(b)}} \sum_{\substack{\ell \in E_b \\ i \equiv d(\ell)}} \prod_{e \in E_b \setminus \{\ell\}} X(d(e))$$

cf.

- $(x^n)' = nx^{n-1}$
- $(f(x) + g(x))' = f'(x) + g'(x)$

Zipper Datatype

- For dependent polynomial functor $\boldsymbol{F}$ for an IF,

$$\boldsymbol{Zipper}(\boldsymbol{m}) \stackrel{def}{=} \boldsymbol{\mu}\, \boldsymbol{F}(\boldsymbol{m}) \times \boldsymbol{Ctx}(\boldsymbol{m})$$

$$\boldsymbol{Ctx}(\boldsymbol{m}) \cong \mathbf{1} + \sum_{\boldsymbol{n} \in \boldsymbol{I}} \boldsymbol{\partial}_{\boldsymbol{m}}\, \boldsymbol{F}(\boldsymbol{\mu}\, \boldsymbol{F})(\boldsymbol{n}) \times \boldsymbol{Ctx}(\boldsymbol{n})$$

- Navigation operations are defined accordingly

Summary

- ▷ Polynomial representation of Inductive Families
- ▷ that automatically generates dependent polynomial functors
- ▷ Zippers for Inductive Families by differentiation

Reference

Hamana, Fiore. *A Foundation for GADTs and Inductive Families: Dependent Polynomial Functor Approach*, WGP'11, 2011-9-18.

Related Work

- Induction-recursion. Dybjer and A. Setzer
  [Ann. Pure Appl. Logic'03][J. Log. Algebr. Program.'06]
  - What are functors for inductive recursive definitions?
  - Next step
- Indexed containers. Morris and Altenkirch [LICS'09]
  - Type theoretic characterisations
  - Mathematically equivalent

Containers & Dependent Polynomials

Indexed containers [Morris and Altenkirch'09]

- $S \in \mathbf{Set}^J$
- $P \in I \times \sum_{j \in J} Sj \to \mathbf{Set}$

$$(S \lhd P) \in Cont\ I\ J$$

Containers & Dependent Polynomials

Indexed containers [Morris and Altenkirch'09]

- $S \in \mathbf{Set}^J$
- $P \in I \times \sum_{j \in J} Sj \to \mathbf{Set}$

$$(S \lhd P) \in Cont\ I\ J$$

$Cont \to Poly$

$$I \xleftarrow{\pi_1 \pi} \sum_{i,a \in I \times \sum_{j \in J} Sj} P(i,a) \xrightarrow{\pi_2 \pi} \sum_{j \in J} Sj \xrightarrow{\pi} J$$

Containers & Dependent Polynomials

Indexed containers [Morris and Altenkirch'09]

- $S \in \mathbf{Set}^J$
- $P \in I \times \sum_{j \in J} Sj \to \mathbf{Set}$

$$(S \lhd P) \in Cont\ I\ J$$

$Poly \to Cont$

$$I \xleftarrow{d} E \xrightarrow{p} B \xrightarrow{c} J$$

- $c^{-1} \in \mathbf{Set}^J$
- $\langle d, p \rangle : E \to I \times B \qquad \langle d, p \rangle^{-1} : \mathbf{Set}^{I \times B}$

$$(c^{-1} \lhd \langle d, p \rangle^{-1}) \in Cont\ I\ J$$

Containers & Dependent Polynomials

Both generate the same dependent polynomial functor

$$\mathbf{\Sigma}_c(\mathbf{\Pi}_p(d^*(X))) \quad \text{“dependent polynomial functor”}$$

$$\cong \sum_{b \in B_j} \prod_{e \in E_b} X(de)$$

$$\cong \sum_{b \in B_j} \prod_{e \in \sum_{i \in I} E_{i,b}} X(\pi e)$$

$$\cong \sum_{b \in B_j} \prod_{i \in I} \prod_{e \in E_{i,b}} X(i)$$

$$\cong \sum_{b \in B_j} \prod_{i \in I} E_{i,b} \Rightarrow X(i)$$

$$\cong [\![(c^{-1} \triangleleft \langle d, p \rangle^{-1})]\!](X) \quad \text{“extension of container”}$$

Containers & Dependent Polynomials

- Indexed vs. fibrational

$$p^{-1} : I \to \mathbf{Set} \quad \Leftrightarrow \quad p : \sum_{i \in I} p^{-1}(i) \to I$$

- $Cont \to Poly$ "fibrational"

$$I \xleftarrow{\pi_1 \pi} \sum_{i,a \in I \times \sum_{j \in J} Sj} P(i,a) \xrightarrow{\pi_2 \pi} \sum_{j \in J} Sj \xrightarrow{\pi} J$$

- $Poly \to Cont$

$$I \xleftarrow{d} E \xrightarrow{p} B \xrightarrow{c} J$$

$$(c^{-1} \triangleleft \langle d, p \rangle^{-1}) \in Cont\ I\ J \quad \text{"indexed"}$$