A Foundation for GADTs and Inductive Families

Dependent Polynomial Functor Approach

Makoto Hamana

Gunma University, Japan

Joint work with

Marcelo Fiore, University of Cambridge

WGP'11, 18 September, 2011.

This Work

▷ Dependent polynomial functor representation of GADTs and Inductive Families, uniformly

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
. GADTs/IFs in these papers

Aim ▶ Recipes for dependent polynomials for GADTs/IFs
. "mathematical codes"

Related Work

Johann and Ghani,
Foundations for structured programming with GADTs,
POPL'08.

Our dependent polynomial functor approach

- ▷ Refines this
- ▷ Unified framework to deal with GADTs and IFs

ADTs and Programming Techniques

ADTs have a solid foundation: ordinary polynomial functors

It is the basis of 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 GADTs

[I] Polynomial representation of GADTs
that generates dependent polynomial functors

[II] Zippers for GADTs (and IFs)

ADTs $\longrightarrow$ GADTs

polynomial ft.s dependent polynomials & ft.s

zippers, etc.

Review: Meaning of Algebraic Datatypes

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

- **Assumption:** set-theoretic models
- Semantics = the initial F-algebra $\alpha : F\,A \xrightarrow{\cong} A$

$$F : \mathbf{Set} \longrightarrow \mathbf{Set}$$

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

- **Point:** polynomial functor F characterises `List`
- How can we extend this to GADTs?

I. How to Model GADTs

GADTs with Type-level Data

▷ Bounded natural numbers

```
data Z
data S a

data Fin :: * -> * where
  Zero :: Fin (S a)
  Succ :: Fin a -> Fin (S a)
```

Modelling Fin

```
data Fin :: * -> * where
  Zero :: Fin (S a)
  Succ :: Fin a -> Fin (S a)
```

- What is the polynomial functor for `Fin`?
- Answer:
$$F_{Fin} : \mathbf{Set}^U \longrightarrow \mathbf{Set}^U$$
$$F_{Fin}(X)(\mathtt{S}\ a) = 1 + X(a)$$
$$F_{Fin}(X)(a) = \varnothing \qquad \text{otherwise}$$
$$[\![\mathtt{Fin}]\!] = \text{the initial } F_{Fin}\text{-algebra Fin} \in \mathbf{Set}^U$$
- How to derive? What are “polynomials”?
- Dependent polynomials

The Universe of Discourse

ADTs		GADTs
$\mathbf{Set}$	$\leadsto$	$\mathbf{Set}^{U}$
the category of sets		the category of U-indexed sets
polynomial ft.		dependent polynomial ft.

Category of Indexed Sets

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

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

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

$$
\begin{tikzcd}
\mathbf{Set}^I \arrow[r, shift left=4, "\Sigma_h"] \arrow[r, shift right=4, "\Pi_h"'] & \mathbf{Set}^J \arrow[l, "h^*" description]
\end{tikzcd}
\qquad \Sigma_h \dashv h^* \dashv \Pi_h
$$

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

$$h^*(A)(j) = A(h(j))$$

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

Lawvere's quantifiers by adjointness [1969]

Dependent Polynomials [Gambino-Hyland'03]

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

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

- 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 :: * -> * where
  Zero :: Fin (S a)
  Succ :: Fin a -> Fin (S a)
```

- Model constructors as polynomials

$$\boldsymbol{Zero} = \qquad \boldsymbol{U} \xleftarrow{!} \varnothing \xrightarrow{!} \boldsymbol{U} \xrightarrow{\mathrm{S}} \boldsymbol{U}$$

$$\boldsymbol{Succ} = \qquad \boldsymbol{U} \xleftarrow{\mathbf{id}} \boldsymbol{U} \xrightarrow{\mathbf{id}} \boldsymbol{U} \xrightarrow{\mathrm{S}} \boldsymbol{U}$$

Modelling Fin

```
data Fin :: * -> * where
  Zero :: Fin (S n)
  Succ :: Fin n -> Fin (S n)
```

- Sum of polynomials is again a polynomial

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

- Dependent polynomial functor $F_{Fin} : \mathbf{Set}^U \longrightarrow \mathbf{Set}^U$

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

Modelling Fin

▷ Dependent polynomial functor

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

is equivalent to the definition by pattern-matching

$$\begin{aligned} &\boldsymbol{F_{Fin}} : \mathbf{Set}^{U} \longrightarrow \mathbf{Set}^{U} \\ &\boldsymbol{F_{Fin}(X)}(\mathtt{S}\ \boldsymbol{a}) = 1 + \boldsymbol{X(a)} \\ &\boldsymbol{F_{Fin}(X)(a)} = \varnothing \qquad \text{otherwise} \end{aligned}$$

▷ 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 :: * -> * where
  Zero :: Fin (S n)
  Succ :: Fin n -> Fin (S n)
```

(2) Polynomial

$$\boldsymbol{Zero} = \qquad U \xleftarrow{!} \varnothing \xrightarrow{!} U \xrightarrow{\mathtt{S}} U.$$

$$\boldsymbol{Succ} = \qquad U \xleftarrow{\mathrm{id}} U \xrightarrow{\mathrm{id}} U \xrightarrow{\mathtt{S}} U.$$

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

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

General Case: Simple GADT

$$\texttt{data } D : *^n \longrightarrow * \texttt{ where}$$
$$K : \forall \overline{\alpha}^l, \overline{\epsilon}^m.\ D(d_1[\overline{\alpha}, \overline{\epsilon}]) \longrightarrow \cdots \longrightarrow D(d_k[\overline{\alpha}, \overline{\epsilon}]) \longrightarrow D(c[\overline{\alpha}])$$

- Polynomial
  (functions $d_i : U^{l+m} \longrightarrow U^n, \quad c : U^l \longrightarrow U^n$)

$$U^n \xleftarrow{[d_1,\ldots,d_k]} kU^{l+m} \xrightarrow{\nabla_k} U^{l+m} \xrightarrow{c\pi_l} U^n$$

  - "Co-diagonal" $\nabla_k = [\mathrm{id}_U, \ldots, \mathrm{id}_U] : kU \longrightarrow U$

- Dependent polynomial functor $F_D : \mathbf{Set}^{U^n} \longrightarrow \mathbf{Set}^{U^n}$

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

II. Application: Zippers

Zippers

- G. Huet, *Functional Pearl: The Zipper*, Journal of Functional Programming, 1997.
- A data structure for navigating a tree freely
- A zipper = current forcus & lists of depth-one contexts
- Generic way to give the type of depth-one contexts
- McBride's finding
  - Binary trees $\boldsymbol{F(X) = 1 + X \times X}$
  - Depth-one contexts $\boldsymbol{F'(X) = X + X}$ – differentiation
- Only for ADTs and polynomial functors
- Extension to GADTs/IFs and dependent polynomial functors

Differentiation

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

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

- ▷ Partial derivative of $\boldsymbol{F}$ with respect to $\boldsymbol{i} \in \boldsymbol{I}$

$$\boldsymbol{\partial}_i\, \boldsymbol{F} : \mathbf{Set}^{\boldsymbol{I}} \longrightarrow \mathbf{Set}^{\boldsymbol{J}}$$

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

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

Zipper Datatype

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

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

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

- Navigation operations are defined accordingly

Summary

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

Reference

Comanion slides at AIM-DTP'11 Shonan Workshop
are available from my homepage

- ▶ More on inductive families

Related Work

1. Initial algebras for GADTs. Johann and Ghani [POPL'08]
   - ▷ Use Left Kan extension for representing the codomains

$$\mathbf{Lan}_h \dashv (- \circ h) \dashv \mathbf{Ran}_h$$

   Ours: Dependent polynomial functors
   - ▷ Use all constructs, i.e. more structured

$$\boldsymbol{\Sigma}_h \dashv h^* \dashv \boldsymbol{\Pi}_h$$

2. Indexed containers. Altenkirch and Morris [LICS'09]
   - ▷ Type theoretic characterisations
   - ▷ Mathematically equivalent
3. Indexed functors. Löh, Magalhães [WGP'11]

Relationships

$$
\begin{tikzcd}[column sep=huge, row sep=huge]
\textbf{(1)}\quad \boldsymbol{Cont(I,J)} \arrow[rr, shift right, "\simeq"'] \arrow[dr, "\text{interpretation}\ [\![-]\!]^{C}"'] & & \boldsymbol{Poly(I,J)}\quad \textbf{(2)} \arrow[ll, shift right] \arrow[dl, "\text{poly. funct.}\ F_{(\_)}"] \\
& \textbf{(3)}\quad [\mathbf{Set}^{I}, \mathbf{Set}^{J}] &
\end{tikzcd}
$$

(1) Indexed containers, Altenkirch and Morris

(2) Dependent polynomials, Gambino, Hyland; Hamana, Fiore

(3) Indexed functors, Löh, Magalhães

Problem ▶ Indexed functor may not have an initial algebra