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.


**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 ———> 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 $\alpha : FA \xrightarrow{\cong} A$

$$
F : \text{Set} \to \text{Set} \\
F(X) = 1 + \mathbb{Z} \times X
$$

▷ 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: $\mathcal{F}_{Fin} : \mathbf{Set}^U \rightarrow \mathbf{Set}^U$

  \begin{align*}
    \mathcal{F}_{Fin}(X)(S\ a) &= 1 + X(a) \\
    \mathcal{F}_{Fin}(X)(a) &= \emptyset \quad \text{otherwise}
  \end{align*}

  $[\text{Fin}] = \text{the initial } \mathcal{F}_{Fin}\text{-algebra } \text{Fin} \in \mathbf{Set}^U$

▷ How to derive? What are “polynomials”?

▷ Dependent polynomials
The Universe of Discourse

**ADTs**

\[
\text{Set} \quad \rightsquigarrow \quad \text{Set}^U
\]

the category of sets

polynomial ft.

**GADTs**

the category of \(U\)-indexed sets

dependent polynomial ft.
Category of Indexed Sets

> **Category** $\text{Set}^U$ for an arbitrary set $U$

- **Objects:** $A : U \to \text{Set}$
  
i.e. $U$-indexed sets $\{ A(i) \mid i \in U \}$

- **Arrows:** $U$-indexed functions $f : A \to B$,
  
i.e. a family of functions $(f(i) : A(i) \to B(i) \mid i \in U)$

> **Important functors:** given a function $h : I \to J$,

\[
\begin{align*}
\Sigma_h & : \text{Set}^I \to \text{Set}^J \\
\Pi_h & : \text{Set}^I \to \text{Set}^J
\end{align*}
\]

\[
\begin{align*}
\Sigma_h(A)(j) &= \sum_{i \in I} A(i) \\
h^*(A)(j) &= A(h(j)) \\
\Pi_h(A)(j) &= \prod_{i \in I} A(i)
\end{align*}
\]

Lawvere’s quantifiers by adjointness [1969]
**Def.** A (dependent) polynomial $P$ is a triple $P = (d, p, c)$ of functions between sets

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

**NB.** the original version uses a lccc and slices
**Def.** The dependent polynomial functor $F_P$ associated to a dependent polynomial $P = (d, p, c)$ is defined by
\[
F_P : \text{Set}^I \rightarrow \text{Set}^J
\]

\[
F_P(X) \overset{\text{def}}{=} \sum_c (\prod_p (d^*(X))).
\]

**i.e.**
\[
F_P(X)(j) = \sum_{b \in B} \prod_{\substack{e \in E \ j \equiv c(b) \ b \equiv p(e)}} X(d(e))
\]
Modelling \text{Fin}

data \text{Fin} :: * \rightarrow * \text{ where }
\begin{align*}
\text{Zero} & :: \text{Fin} (\text{S} \ a) \\
\text{Succ} & :: \text{Fin} \ a \rightarrow \text{Fin} (\text{S} \ a)
\end{align*}

Model constructors as polynomials

\begin{align*}
\text{Zero} &= \quad \begin{tikzpicture}
\node (A) at (0,0) {$U$};
\node (B) at (-1,0) {$\emptyset$};
\node (C) at (1,0) {$U$};
\node (D) at (2,0) {$U$};
\draw[->] (A) -- (B) node[midway, above] {$!$};
\draw[->] (B) -- (C) node[midway, above] {$!$};
\draw[->] (C) -- (D) node[midway, above] {$\text{S}$};
\end{tikzpicture} \\
\text{Succ} &= \quad \begin{tikzpicture}
\node (A) at (0,0) {$U$};
\node (B) at (-1,0) {$U$};
\node (C) at (1,0) {$U$};
\draw[->] (A) -- (B) node[midway, above] {$\text{id}$};
\draw[->] (B) -- (C) node[midway, above] {$\text{id}$};
\draw[->] (C) -- (A) node[midway, above] {$\text{S}$};
\end{tikzpicture}
\end{align*}
Modelling Fin

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

Sum of polynomials is again a polynomial

\[ \text{Fin} \overset{\text{def}}{=} \text{Zero} + \text{Succ} \]

Dependent polynomial functor \( F_{\text{Fin}} : \text{Set}^U \rightarrow \text{Set}^U \)

\[
F_{\text{Fin}}(X)(n) = F_{\text{Zero}+\text{Succ}}(X)(n) \\
= F_{\text{Zero}}(X)(n) + F_{\text{Succ}}(X)(n) \\
= \Sigma_s \Pi !*(X)(n) + \Sigma_s \Pi \text{id}!*(X)(n) \\
= \sum_{a \in U} (1 + X(a)))
\]
Modelling Fin

- Dependent polynomial functor

\[
F_{\text{Fin}}(X)(n) = \sum_{\substack{a \in U \\ n \equiv S \ a}} (1 + X(a)))
\]

is equivalent to the definition by pattern-matching

\[
F_{\text{Fin}} : \text{Set}^U \rightarrow \text{Set}^U
\]

\[
F_{\text{Fin}}(X)(S \ a) = 1 + X(a)
\]

\[
F_{\text{Fin}}(X)(a) = \emptyset \quad \text{otherwise}
\]

- Initial algebra is constructed by repeated applications of \(F_{\text{Fin}}\)

**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

    Zero = U ⦿ ⦿ 0 ⦿ 0 U → U → U.

    Succ = U ☐ U ☐ U → U → U.

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

    $F_{Fin}(X)(n) = \sum_{a \in U \atop n \equiv S a} (1 + X(a)))$
General Case: Simple GADT

```latex
\begin{align*}
\text{data } D : \ast^n & \to \ast \text{ where } \\
K : \forall \alpha^l, \varepsilon^m. D(d_1[\alpha, \varepsilon]) & \to \cdots \to D(d_k[\alpha, \varepsilon]) \to D(c[\alpha])
\end{align*}
```

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

- “Co-diagonal” $\nabla_k = [\text{id}_U, \ldots, \text{id}_U] : kU \to U$

- **Dependent polynomial functor** $F_D : \text{Set}^{U^n} \to \text{Set}^{U^n}$

\[
F_D X(m) = \sum_{j \in U} X(d_1(j)) \times \cdots \times X(d_k(j))
\]
II. Application: Zippers
Zippers


- A data structure for navigating a tree freely

- A zipper = current focus & lists of depth-one contexts

- Generic way to give the type of depth-one contexts

- McBride’s finding
  - Binary trees $F(X) = 1 + X \times X$
  - Depth-one contexts $F'(X) = X + X$ – differentiation

- Only for ADTs and polynomial functors

- Extension to GADTs/IFs and dependent polynomial functors
Differentiation

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

\[
F(X)(j) = \sum_{b \in B} \prod_{j \equiv c(b)} X(d(e))
\]

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

\[
\partial_i F : \text{Set}^I \to \text{Set}^J
\]

\[
\partial_i F(X)(j) = \sum_{e \in E} \sum_{\ell \in E_b} \prod_{j \equiv c(b)} X(d(e))
\]

Derived from differentiation of generalised species [Fiore FOSSACS’05, etc.]
For dependent polynomial functor $F$ for an GADT/IF,

\[
\text{Zipper}(m) \overset{\text{def}}{=} \mu F(m) \times \text{Ctx}(m)
\]

\[
\text{Ctx}(m) \cong 1 + \sum_{n \in I} \partial_m F(\mu F)(n) \times \text{Ctx}(n)
\]

Navigation operations are defined accordingly.
Summary

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

Reference

Companion *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
     \[
     \text{Lan}_h \dashv (\neg \circ h) \dashv \text{Ran}_h
     \]
   - Ours: Dependent polynomial functors
     - Use **all constructs**, i.e. more structured
     \[
     \Sigma_h \dashv h^* \dashv \Pi_h
     \]

2. Indexed containers. Altenkirch and Morris [LICS’09]
   - Type theoretic characterisations
   - Mathematically equivalent

3. Indexed functors. Löh, Magalhães [WGP’11]
Relationships

(1) $\text{Cont}(I, J) \xrightarrow{\sim} Poly(I, J) \quad (2)$

(3) $[\text{Set}^I, \text{Set}^J]$ 

(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