Equational Theories and Monads from Polynomial Cayley Representations

Maciej Piróg

(joint work with Piotr Polesiuk and Filip Sieczkowski)

`pl-uwr.bitbucket.io/caymon/`

Recipe for monads (a recap from Gordon's talk)

- Take any (finitary) equational theory (Σ, E) you can imagine,
- Take the equivalence $\sim$ induced by the equations,
- Your monad is given by $\Sigma^E A = [\Sigma^* A]_\sim$,
- The monadic structure is induced by freeness.

If you're a set-theorist
or maybe a HoTT person

If you're a Haskell
programmer

The puzzle for this beautiful morning is…

Which monads can I implement in, say, Haskell using

$$+, \times, \rightarrow, \forall, \exists?$$

It is a **serious question**, about the very nature of the connection of different equational theories and computation.

Sadly, it is a horribly difficult question!

What do other people with undecidable problems?

METHOD 1: Ignore altogether

Examples: `UndecidableInstances`, C++ templates

METHOD 2: Investigate specific cases. E.g., satisfiability:

FOL is undecidable ✗
FOL with 2 variables is decidable ✓
FOL with 2 variables and 2 transitive relations is not ✗
FOL with 2 variables and 1 transitive relation is... **???**

What are the specific cases that we can examine?

- ❌ Equational theories in general
- ❌ Possible implementations of monads
- ✔️ Types that are always equipped with canonical monadic structure

In particular...

$$\mathbf{M\ a = (a \to X) \to X}$$

is a monad for all **types** **X**

In particular...

$$\mathbf{M\ a = \forall x.\ (a \to F\ x) \to F\ x}$$

is a monad for all **functors F**

In particular...

$$\mathbf{M\ a = \forall x.\ (a \longrightarrow F\ x\ x) \longrightarrow F\ x\ x}$$

is a monad for all **mixed-variance bifunctors F**

$$\mathbf{M\ a = \forall x.\ (a \to F\ x\ x) \to F\ x\ x}$$

as in

$$\mathbf{List\ a = \forall x.\ (a \to (x \to x)) \to (x \to x)}$$

or

$$\mathbf{State\ s\ a = \forall x.\ (a \to (s \to x)) \to (s \to x)}$$

Did I just say State?...

$$\mathbf{State\ s\ a} = \forall x.\ (a \to s \to x) \to s \to x$$

$$\text{(flip)} \cong \forall x.\ s \to (a \to s \to x) \to x$$

$$(\to \text{ and } \forall) \cong s \to \forall x.\ (a \to s \to x) \to x$$

$$\text{(Church)} \cong s \to (a, s)$$

The overall idea

(inspired by Ralf Hinze's "Kan extensions for program optimization")

I prove the following (vaguely stated) theorem:

> If an equational theory $\mathcal{T}$ has a
>
> **well-behaved Cayley representation F**,
>
> then the monad
>
> $$\mathbf{M\,a = \forall x.\,(a \rightarrow F\,x\,x) \rightarrow F\,x\,x}$$
>
> is the free monad of $\mathcal{T}$.

...reducing(?) the problem of finding implementations of free models of theories to finding implementations of Cayley representations of theories.

Making the statement of the theorem more precise (1)

Our domain is the category **SET** of sets and functions.

We model **our particular** polymorphic functions as what **Philip Mulry** calls *strong dinatural transformations*, while **Michael Barr** calls *Barr-dinatural transformations*

Making the statement of the theorem more precise (2)

A *well-behaved Cayley representation* of $\mathscr{T}$ with respect to $F \dashv U$ consists of the following components: • A bifunctor $R : \mathbf{Set}^{\mathrm{op}} \times \mathbf{Set} \to \mathbf{Set}$, • For each set X, an object $\mathbb{R}X$ in $\mathscr{T}$, such that $U\mathbb{R}X = RXX$, • For all sets A, X, Y and functions $f_1 : A \to RXX$, $f_2 : A \to RYY$, $g : X \to Y$, it is the case that

if
$$\begin{array}{ccc} & RXX & \\ {}^{f_1}\nearrow & & \searrow^{RXg} \\ A & & RXY \\ {}_{f_2}\searrow & & \nearrow_{RgY} \\ & RYY & \end{array}$$
commutes, then
$$\begin{array}{ccc} & RXX & \\ {}^{\widehat{f_1}}\nearrow & & \searrow^{RXg} \\ UFA & & RXY \\ {}_{\widehat{f_2}}\searrow & & \nearrow_{RgY} \\ & RYY & \end{array}$$
commutes. • For each object M in $\mathscr{T}$, a morphism $\sigma_M : M \to \mathbb{R}(UM)$ in $\mathscr{T}$, such that $U\sigma_M : UM \to R(UM)(UM)$ is Barr-dinatural in M, • A Barr-dinatural transformation $\rho_M : R(UM)(UM) \to UM$, such that $\rho_M \cdot U\sigma_M = \mathrm{id}$, • For each set X, a set of indices I_X and a family of functions $\mathrm{run}_{X,i} : RXX \to X$, where $i \in I_X$, such that $R(RXX)\mathrm{run}_X$ is a jointly monic family, and the following diagram commutes for all X and $i \in I_X$:

$$\begin{array}{ccc} RXX & \xrightarrow{U\sigma_{\mathbb{R}X}} & R(RXX)(RXX) \\ & {}_{R\mathrm{run}_{X,i}X}\searrow & \downarrow{\scriptstyle R(RXX)\mathrm{run}_{X,i}} \\ & & R(RXX)X \end{array}$$

So what can I offer you today, exactly?

I can offer you **(many-sorted)** equational theories Cayley-represented by the type

$$\mathbf{F}\,\mathbf{x}\,\mathbf{y} = \mathbf{P}\,\mathbf{x} \to \mathbf{y}$$

where **P** is a polynomial functor with natural coefficients (= finite sets).

Polynomial:

$$PX = \sum_{i=1}^{d} c_i \times X^{e_i}$$

Sorts:

Ω (main sort),

K_i, for all $i \leq d$

Operations:

cons : $\prod_{i=1}^{d} K_i^{c_i} \to \Omega$

$\pi_i^j : \Omega \to K_i$, for $i \leq d$ and $j \leq c_i$

$\varepsilon_i^j : K_i$, for $i \leq d$ and $j \leq e_i$

$\gamma_i^j : K_j \times K_i^{e_j} \to K_i$, for $i, j \leq d$

Equations:

$$\pi_i^j(\mathrm{cons}([[x_i^j]_{j \leq c_i}]_{i \leq d})) = x_i^j \qquad \text{(beta-}\pi\text{)}$$

$$\mathrm{cons}([[\pi_i^j(x)]_{j \leq c_i}]_{i \leq d}) = x \qquad \text{(eta-}\pi\text{)}$$

$$\gamma_i^j(\varepsilon_j^k, [x_t]_{t \leq e_j}) = x_k \qquad \text{(beta-}\varepsilon\text{)}$$

$$\gamma_i^i(x, [\varepsilon_i^j]_{j \leq e_i}) = x \qquad \text{(eta-}\varepsilon\text{)}$$

$$\gamma_i^j(\gamma_j^k(x, [y_t]_{t \leq e_k}), [z_s]_{s \leq e_j}) = \gamma_i^k(x, [\gamma_i^j(y_t, [z_s]_{s \leq e_j})]_{t \leq e_k}) \qquad \text{(assoc-}\gamma\text{)}$$

Example: Px = n

Sorts:

Ω, K

Operations:

$\pi^t : \Omega \to K \quad (t \leq n)$

$\text{cons} : K^n \to \Omega$

Equations:

$\pi^t(\text{cons}([x_i]_{i \leq n})) = x_t$

$\text{cons}([\pi^i(x)]_{i \leq n}) = x$

Macro-operations:

$\text{put}^t : \Omega \to \Omega$

$\text{get} : \Omega^n \to \Omega$

$\text{put}^t(x) = \text{cons}([\pi^t(x)]_n)$

$\text{get}([x_i]_{i \leq n}) = \text{cons}([\pi^i(x_i)]_{i \leq n})$

Example: **Px = n**

$\text{put}^j(\text{put}^k(x))$

$= \text{(definition of put)}$

$\text{cons}([\boldsymbol{\pi}^j(\text{cons}([\boldsymbol{\pi}^k(x)]_n))]_n)$

$= (\text{beta-}\boldsymbol{\pi})$

$\text{cons}([\boldsymbol{\pi}^k(x)]_n)$

$= \text{(definition of put)}$

$\text{put}^k(x)$

$\text{put}^j(\text{get}([x_i]_{i \leq n}))$

$= \text{(definition of get)}$

$\text{put}^j(\text{cons}([\boldsymbol{\pi}^i(x_i)]_{i \leq n}))$

$= \text{(definition of put)}$

$\text{cons}([\boldsymbol{\pi}^j(\text{cons}([\boldsymbol{\pi}^i(x_i)]_{i \leq n}))]_n)$

$= (\text{beta-}\boldsymbol{\pi})$

$\text{cons}([\boldsymbol{\pi}^j(x_j)]_n)$

$= \text{(definition of put)}$

$\text{put}^j(x_j)$

Effects

(Ohad, please put on red glasses. Jeremy, please put on blue glasses)

Px	**Effect**
x	Nondeterminism
n	State
nx	Nondeterminism with local/provisional state
x^n	Nondeterminism with global/persistent state
nx^p	Nondeterminism with both local/provisional state and global/persistent and state
$nx^p + mx^q$	Nondeterminism with global/persistent state dependent on the local/provisional state

Lessons…

I was surprised to see state

…yet alone the appropriate combinations of state and nondeterminism

The formula produced a novel (at least to me) presentation of state in terms of 2-sorted theory of tupling and projections

The formula produced a novel (at least to me) presentation of local/provisional state – one I probably wouldn’t write from the top of my head

Note to self:

You were supposed to show the **Caymon** tool, but I guess you’re out of time by now!

`pl-uwr.bitbucket.io/caymon/`