Graphical algebraic foundations for monad stacks

Ohad Kammar

Higher-Order Programming with Effects
31 August, 2014

UNIVERSITY OF CAMBRIDGE Computer Laboratory

erc European Research Council
EPSRC Engineering and Physical Sciences Research Council

Problem statement

- ? Effects in a pure language.
- ! Use monads.
- ? Monads don't compose.
- ! Use monad transformers (monad stacks).
- ? But which order...

  (StateT s . ErrorT e)
  vs
  (ErrorT e . StateT s)

- !? Current practice relies on:
  Programmer insight and experience[1], and black art[2].
- ! Towards a systematic approach? Tool support?

[1] HOPE reviewer #1.
[2] HOPE reviewer #3.

Demo (part I)

`http://www.cl.cam.ac.uk/~ok259/graphtool`

Demo (part I)

http://www.cl.cam.ac.uk/~ok259/graphtool

Small print

Full details later, but the tool:

- can't handle all monad transformers; and
- might fail to find valid monad stacks.

Talk structure

1. Algebraic effects.
2. Cographs (aka series-parallel graphs).
3. Tool.
4. Conclusion.

Algebraic effects (Plotkin and Power 2002)

Semantics for exceptions

Let m be a (set-theoretic) monad, and e a set of exceptions. We say that $\langle m, raise \rangle$ is an *e-exception* monad if *raise* is a Kleisli arrow:

$$raise :: e \to m\ \emptyset$$

The *initial* e-exception monad is the exception monad $m\ a = Error\ e\ a$ with its standard *raise* operation.

I.e., for every other e-exception monad $\langle m', raise' \rangle$, there exists a unique monad morphism $h :: m \to m'$ satisfying for all $exc :: e$:

$$h(raise\ exc) = raise'\ exc$$

Algebraic effects (Plotkin and Power 2002)

Semantics for global state

Let m be a (set-theoretic) monad, and s a set of states.
We say that $\langle m, get, put \rangle$ is a *global s-state* monad if *get* and *put* are Kleisli arrows:

$$get :: () \to m\ s \qquad put :: s \to m\ ()$$

such that the following three equations hold (Plotkin and Power 2002, and Melliès 2010):

$$\begin{array}{l} x \leftarrow get\ (); \\ put\ x \end{array} = return\ () \qquad \begin{array}{l} put\ x; \\ get\ () \end{array} = \begin{array}{l} put\ x; \\ return\ x \end{array} \qquad \begin{array}{l} put\ x; \\ put\ y \end{array} = put\ y$$

(in $m\ ()$, $m\ s$, and $m\ ()$ respectively).

The *initial* global s-state monad is the global s-state monad $m\ a = State\ s\ a = s \to (s, a)$ with its *get* and *put* operations.

Algebraic effects (Plotkin and Power 2002)

Algebraic semantics (part 1)

A *presentation* is a triple $\langle \pi, ar, E \rangle$ consisting of a set π (of *generic effect symbols*) and a π-indexed collection of pairs of sets ar:

$$\langle p_{op}, a_{op} \rangle \qquad op \in \pi$$

(the pair $\langle \pi, ar \rangle$ is called a *signature*), and E is a set of pairs of terms (called *equations*) involving the monadic *return* and *do* notation, and Kleisli arrows:

$$op :: p_{op} \to m\ a_{op}$$

for all $op \in \pi$.

Algebraic effects (Plotkin and Power n2002)

Algebraic semantics (part 2)

Given a presentation $P = \langle \pi, ar, E \rangle$, a P-monad is a monad m and an assignment of Kleisli arrows:

$$op :: p_{op} \to m\ a_{op}$$

for all $op \in \pi$, satisfying all the equations in E.

The *initial* P-monad m_P always exists.

All these concepts are well established and date back to Lawvere's thesis (1963) and to Linton (1966).

Plotkin and Power's algebraic theory of effects analyses monads used in the semantics of computational effects in terms of their presentations.

(Excludes the continuation monad, more details offline.)

Examples (Plotkin and Power 2002)

Previous examples

- Exceptions: $\mathit{raise} :: e \to m\ \emptyset$, no equations
- Global state: $\mathit{get} :: () \to m\ s$, $\mathit{put} :: s \to m\ ()$, as before

Additional examples

- Environment monad: $\mathit{get} :: () \to m\ s$, equations:

$$\begin{array}{l} x \leftarrow \mathit{get}\ (); \\ \mathit{return}\ () \end{array} = \mathit{return}\ () \qquad \begin{array}{l} x \leftarrow \mathit{get}\ (); \\ y \leftarrow \mathit{get}\ (); \\ \mathit{return}(x, y) \end{array} = \begin{array}{l} z \leftarrow \mathit{get}\ (); \\ \mathit{return}(z, z) \end{array}$$

- Writer monad for a monoid $\langle \mathit{mon}, \cdot, 1 \rangle$: $\mathit{act} :: \mathit{mon} \to m\ ()$:

$$\begin{array}{l} \mathit{act}\ m_1; \\ \mathit{act}\ m_2 \end{array} = \mathit{act}\ (m_1 \cdot m_2) \qquad \mathit{act}\ 1 = \mathit{return}()$$

- Free monad for a functor F: no eqns (more details offline)

Additional examples (ctd)

- List monad: $\mathit{fail} :: () \to m\ \emptyset$, $\mathit{choose} :: () \to m\ \mathit{bool}$ equations:

$$\begin{array}{ll} x \leftarrow & \mathit{choose}\ (); \\ \mathit{if}\ \ x & \mathit{then}\ \mathit{fail} \\ & \mathit{else}\ \ \mathit{return}\ () \end{array} = \mathit{return}\ () = \begin{array}{ll} x \leftarrow & \mathit{choose}\ (); \\ \mathit{if}\ \ x & \mathit{then}\ \mathit{return}\ () \\ & \mathit{else}\ \ \mathit{fail} \end{array}$$

$$\begin{array}{l} x \leftarrow \mathit{choose}(); \\ y \leftarrow \mathit{choose}(); \\ \mathit{case}(x, y)\mathit{of} \\ \quad \begin{array}{ll} (\mathit{True}, \mathit{True}) & \to \mathit{return}\ 1 \\ (\mathit{True}, \mathit{False}) & \to \mathit{return}\ 2 \\ (\_, \mathit{False}) & \to \mathit{return}\ 3 \end{array} \end{array} = \begin{array}{l} x \leftarrow \mathit{choose}(); \\ y \leftarrow \mathit{choose}(); \\ \mathit{case}(x, y)\mathit{of} \\ \quad \begin{array}{ll} (\mathit{True}, \_) & \to \mathit{return}\ 1 \\ (\mathit{False}, \mathit{True}) & \to \mathit{return}\ 2 \\ (\mathit{False}, \mathit{False}) & \to \mathit{return}\ 3 \end{array} \end{array}$$

Combining effects (Hyland, Plotkin, and Power 2006)

Sum

Every two presentations $P_1 = \langle \pi_1, ar_1, E_1 \rangle$, $P_2 = \langle \pi_2, ar_2, E_2 \rangle$ can be combined by the disjoint union of the operations $\pi_1 + \pi_2$, and subsequent relabelling of the equations. Call the resulting presentation their *sum*, denoted by $P_1 + P_2$.

Theorem

Let P_{exc} be the presentation for e-exceptions. For every presentation P:

$$m_{P_{exc}+P} \cong ErrorT\ e\ m_P$$

Therefore, the action of the exception monad transformer arises as the sum with the theory for exception.

Theorem

Let P_F be the presentation for the free monad for a functor F. For every presentation P: $m_{P_F+P} \cong FreeT\ F\ m_P$.

Combining effects (Hyland, Plotkin, and Power 2006)

Tensor

By adding the following equations

$$\begin{array}{lcl} x_1 \leftarrow op_1\ p_1 & & x_2 \leftarrow op_2\ p_2 \\ x_2 \leftarrow op_2\ p_2 & = & x_1 \leftarrow op_1\ p_1 \\ \mathit{return}\ (x_1, x_2) & & \mathit{return}\ (x_1, x_2) \end{array}$$

for all $op_1 \in \pi_1$, $op_2 \in pi_2$ to the sum $P_1 + P_2$, we obtain another way to combine presentations, their tensor $P_1 \otimes P_2$.

Theorem

Let P_{st}, P_{env}, P_{mon} be the presentations for the s-state, s-environment and mon-writer monads. Then for every presentation P:

$$m_{P_{st} \otimes P} \cong \mathit{StateT}\ s\ m_P \quad m_{P_{env} \otimes P} \cong \mathit{ReaderT}\ s\ m_P$$

$$m_{P_{mon} \otimes P} \cong \mathit{WriterT}\ mon\ m_P$$

Combining effects

Applicability

Covered the MTL (sans continuations).
Not all monad transformers arise as either sum or tensor, even when their associated monads arise from presentations.

Jaskelioff's ListT

$$ListT\ \ m\ a = m\ (Either\ ()\ (a, ListT\ \ m\ a))$$

Theorem

Let P_{list} be the presentations for the list monad. For every presentation P: $m_{P_{list} \odot P} \cong ListT\ \ m_P$, where $P_{list} \odot P$ is obtained from $P_{list} + P$ by adding the following equation, for all $op \in \pi$:

$$\begin{array}{l} b \leftarrow choose();\\ \text{if } \ b \text{ then } y \leftarrow op\ \ p;\\ \qquad\qquad\quad return\ \ Just\ \ y\\ \qquad\ \ else\ \ return\ \ None \end{array} = \begin{array}{l} y \leftarrow op\ \ p;\\ b \leftarrow choose();\\ \text{if } \ b \text{ then } return\ \ Just\ \ y\\ \qquad\ \ else\ \ return\ \ None \end{array}$$

Commutativity analysis (Hyland, Plotkin, and Power 2006)

Setting

Restrict attention to monad transformers arising as sum or tensor of theories (e.g., MTL).

Design choice

Choose, for every pair of effects, whether they should commute.

Analysis

Do these commutative equations:

- arise through sum and tensor of basic theories?
- result from a monad stack of the given transformers?

Commutativity analysis (Hyland, Plotkin, and Power 2006)

Setting

Restrict attention to monad transformers arising as sum or tensor of theories (e.g., MTL).

Design choice

Choose, for every pair of effects, whether they should commute.

$$\mathit{State}\bullet \qquad \bullet\mathit{Exceptions} \qquad \mathit{vs} \qquad \mathit{State}\bullet \leftrightarrow \bullet\mathit{Exceptions}$$

Analysis

Do these commutative equations:

- arise through sum and tensor of basic theories?

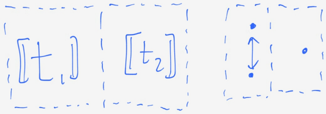
$$t ::= x \mid \sum_{i \in I} t_i \mid \bigotimes_{i \in I} t_i$$

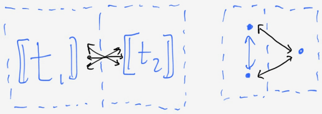
- result from a monad stack of the given transformers?

Graph theory (Hyland, Plotkin, and Power 2006)

Every term denotes a graph:

$$[\![x]\!] = x\bullet$$

$$[\![t_1 + t_2]\!] =$$ 

$$[\![t_1 \otimes t_2]\!] =$$ 

But not all graphs arise in this way, e.g., P_4:

$$\bullet \leftrightarrow \bullet \leftrightarrow \bullet \leftrightarrow \bullet$$

Cographs

Definition

A *cograph* is a graph isomorphic to $[\![t]\!]$ for some t (a.k.a. series-parallel graphs, ambiguously).

Theorem (Corneil et al. 1981)

A graph is a cograph $\iff$ P_4 *does not embed into it*

$$\bullet \leftrightarrow \bullet \leftrightarrow \bullet \leftrightarrow \bullet$$

- Witness for negative result.
- Polynomial time.
- Does not provide a sum and tensor decomposition.

Cographs

Theorem (McConnell and Spinrad 1999)

There is a linear time algorithm for deciding whether a given graph is a cograph, and if so, exhibiting its sum and tensor decomposition.

- Computes the modular decomposition of the graph (more offline).
- Simpler algorithms in polynomial time.

Demo (part II)

http://www.cl.cam.ac.uk/~ok259/graphtool

Demo (part II)

`http://www.cl.cam.ac.uk/~ok259/graphtool`

Small print

- Only applies to algebraic effects (excludes continuations) arising as sum and tensor (excludes ListT).
- Might fail to find valid monad stacks.

$$\text{non-determinism} + \text{exceptions} = \text{non-determinism} \otimes \text{exceptions}$$

$$\bullet \quad \bullet \leftrightarrow \bullet \leftrightarrow \bullet$$

vs

$$\bullet \leftrightarrow \bullet \leftrightarrow \bullet \leftrightarrow \bullet$$

Summary

Conclusion

The algebraic perspective, regardless of the tool, is insightful.

Contributions

- Connecting this problem with cographs (suggested by Atkey).
- Characterising graphs arising from monad stacks (straightforward).
- The algebraic analysis of Jaskelioff's *ListT*.

Further work

- Beyond the MTL (e.g., Jaskelioff's thesis).
- No idea how to deal with continuations.