Graphical algebraic foundations for monad stacks

Ohad Kammar

Higher-Order Programming with Effects
31 August, 2014
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.
http://www.cl.cam.ac.uk/~ok259/ graphtool
http://www.cl.cam.ac.uk/~ok259/grahttool

Small print
Full details later, but the tool:
  ▶ can’t handle all monad transformers; and
  ▶ might fail to find valid monad stacks.
1. Algebraic effects.
2. Cographs (aka series-parallel graphs).
3. Tool.
4. Conclusion.
Semantics for exceptions

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

$$\text{raise} :: e \rightarrow m \emptyset$$

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

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

$$h(\text{raise exc}) = \text{raise'} \text{exc}$$
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 $\langle m, \text{get}, \text{put} \rangle$ is a global $s$-state monad if $\text{get}$ and $\text{put}$ are Kleisli arrows:

$$\text{get} :: () \rightarrow m s \quad \text{put} :: s \rightarrow m ()$$

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

$$x \leftarrow \text{get} () = \text{return} () \quad \text{put} x ; \quad \text{put} x ; \quad \text{put} x ; \quad \text{put} y$$

$$\text{put} x \quad \text{get} () = \text{return} x \quad \text{put} y$$

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

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

Ohad Kammar
Graphical algebraic foundations for monad stacks
Algebraic semantics (part 1)

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

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

(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:

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

for all $op \in \pi$. 
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:

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

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: \( \text{raise} :: e \rightarrow m \emptyset \), no equations
▶ Global state: \( \text{get} :: () \rightarrow m s \), \( \text{put} :: s \rightarrow m () \), as before

Additional examples

▶ Environment monad: \( \text{get} :: () \rightarrow m s \), equations:

\[
\begin{align*}
  x & \leftarrow \text{get} (); \\
  \text{return} () &= \text{return} () \\
  y & \leftarrow \text{get} (); \\
  \text{return}(x, y)
\end{align*}
\]

▶ Writer monad for a monoid \( \langle \text{mon}, \cdot, 1 \rangle \): \( \text{act} :: \text{mon} \rightarrow m () \):

\[
\begin{align*}
  \text{act} m_1; \\
  \text{act} m_2
\end{align*}
\]

▶ Free monad for a functor \( F \): no eqns (more details offline)
Additional examples (ctd)

- List monad: \( \text{fail} :: () \rightarrow m \emptyset \), \( \text{choose} :: () \rightarrow m \text{bool} \)

  equations:

  \[
  x \leftarrow \text{choose}(); \\
  \text{if } x \text{ then fail } = \text{return }() = \text{if } x \text{ then return }() \\
  \text{else return }() \\
  \]

  \[
  x \leftarrow \text{choose}(); \\
  y \leftarrow \text{choose}(); \\
  \text{case}(x, y)\text{of} \\
  (\text{True}, \text{True}) \rightarrow \text{return }1 \\
  (\text{True}, \text{False}) \rightarrow \text{return }2 \\
  (\text{False}, \text{False}) \rightarrow \text{return }3 \\
  \]

  \[
  x \leftarrow \text{choose}(); \\
  y \leftarrow \text{choose}(); \\
  \text{case}(x, y)\text{of} \\
  (\text{True}, \_ \text{False}) \rightarrow \text{return }1 \\
  (\text{False}, \text{True}) \rightarrow \text{return }2 \\
  (\text{False}, \text{False}) \rightarrow \text{return }3 \\
  \]
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 \( P_{\text{exc}} \) be the presentation for e-exceptions. For every presentation \( P \):

\[
m_{P_{\text{exc}}+P} \cong \text{ErrorT e} \ m_P
\]

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 \( P \): \( m_{P_F+P} \cong \text{FreeT F} \ m_P \).
Combining effects (Hyland, Plotkin, and Power 2006)

Tensor
By adding the following equations

\[ x_1 \leftarrow \text{op}_1 \ p_1 \quad x_2 \leftarrow \text{op}_2 \ p_2 \]
\[ x_2 \leftarrow \text{op}_2 \ p_2 \quad = \quad x_1 \leftarrow \text{op}_1 \ p_1 \]
\[ \text{return} \ (x_1, x_2) \quad \text{return} \ (x_1, x_2) \]

for all \( \text{op}_1 \in \pi_1, \text{op}_2 \in \pi_2 \) 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 \( P \):

\[ m_{P_{st} \otimes P} \cong \text{StateT} \ s \ m_P \]
\[ m_{P_{env} \otimes P} \cong \text{ReaderT} \ s \ m_P \]
\[ m_{P_{mon} \otimes P} \cong \text{WriterT} \ \text{mon} \ m_P \]
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

\[ \text{ListT} \ m \ a = m \ (\text{Either} \ () \ (a, \text{ListT} \ m \ a)) \]

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

\[
\begin{align*}
b &\leftarrow \text{choose}(); \\
\text{if } b \text{ then } y &\leftarrow \text{op } p; \\
\text{return } \text{Just } y \\
\text{else } &\text{return None}
\end{align*}
\]

\[
\begin{align*}
y &\leftarrow \text{op } p; \\
b &\leftarrow \text{choose}(); \\
\text{if } b \text{ then } &\text{return Just } y \\
\text{else } &\text{return None}
\end{align*}
\]
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.

\[ State \bullet \bullet Exceptions \quad vs \quad State \bullet \leftrightarrow \bullet Exceptions \]

Analysis
Do these commutative equations:

- arise through sum and tensor of basic theories?

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

- result from a monad stack of the given transformers?
Every term denotes a graph:

\[
[x] = x \bullet
\]

\[
[t_1 + t_2] = \begin{array}{c}
\begin{array}{c}
[t_1] \\
\hline
\end{array}
\end{array} + \begin{array}{c}
\begin{array}{c}
[t_2] \\
\hline
\end{array}
\end{array}
\]

\[
[t_1 \otimes t_2] = \begin{array}{c}
\begin{array}{c}
[t_1] \\
\hline
\end{array}
\end{array} \otimes \begin{array}{c}
\begin{array}{c}
[t_2] \\
\hline
\end{array}
\end{array}
\]

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.
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.
http://www.cl.cam.ac.uk/~ok259/grahtool
Small print

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

\[
\text{non-determinism} + \text{exceptions} = \\
\text{non-determinism} \otimes \text{exceptions}
\]

\[
\bullet \quad \bullet \leftrightarrow \bullet \leftrightarrow \bullet \\
\text{vs}
\]

\[
\bullet \leftrightarrow \bullet \leftrightarrow \bullet \leftrightarrow \bullet
\]
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.