Construcción Modular de Mónadas con Operaciones

Mauro Jaskelioff

CIFASIS /
Depto de Cs. de la Computación
FCEIA - Universidad Nacional de Rosario

JCC 2010

En esta charla

- Estructuración de programas usando mónadas
- Transformadores para la construcción modular de mónadas.
- El problema de *levantar* operaciones (lifting of operations).
- Un nuevo enfoque para la resolución de este problema

Mónadas en Cs de la Computación

- Las mónadas son una estructura algebraica que aparece en la teoría de categorías
- Moggi se da cuenta que pueden modelar una gran cantidad de *efectos computacionales*.
  - Estado, excepciones, continuaciones, no determinismo, entrada/salida, etc.
  - Las usa para estructurar la semántica de lenguajes de programación CBV con efectos computacionales.
- Wadler las utiliza para estructurar programas.
  - Las mónadas son una abstracción muy útil!

Mónadas en Haskell

- Especificadas por una clase de tipo, para $m :: * \to *$

$$
\begin{array}{l}
\textbf{class}\ \mathit{Monad}\ m\ \textbf{where} \\
\quad \mathit{return} :: a \to m\ a \\
\quad (\gg=)\ :: m\ a \to (a \to m\ b) \to m\ b
\end{array}
$$

- Algunas mónadas son:
  - Computaciones puras $\quad \textbf{type}\ \mathit{Id}\ a = a$
  - Excepciones $\quad \textbf{type}\ \mathit{Ex}\ a = \mathit{Either}\ E\ a$
  - Estado $\quad \textbf{type}\ \mathit{St}\ a = S \to (a, S)$
  - Continuaciones $\quad \textbf{type}\ \mathit{Cn}\ a = (a \to R) \to R$

Intérpretes Monádicos

$$\textbf{data}\ Lang = Num\ Int \mid Add\ Lang\ Lang$$

$$\begin{aligned}
&eval && :: Monad\ m \Rightarrow Lang \rightarrow m\ Int \\
&eval\ (Num\ n) && = return\ n \\
&eval\ (Add\ t\ u) && = eval\ t \gg\!= \lambda x \rightarrow \\
& && \quad eval\ u \gg\!= \lambda y \rightarrow \\
& && \quad return\ (t + u)
\end{aligned}$$

- *eval* está definida para *cualquier* mónada:
- La mónada abstrae la composición de efectos computacionales y la noción de valor puro (efecto trivial).

Operaciones manipuladoras de efectos

- Cada mónada M viene equipada con operaciones que manipulan los efectos que la mónada modela.

- Por ejemplo:
  - Cuando $M = St$ (estado):

$$\begin{array}{l} get :: () \to M\ S \\ put :: S \to M\ () \end{array}$$

  - Cuando $M = Ex$ (excepciones):

$$\begin{array}{l} throw\ \ :: E \to M\ a \\ handle :: M\ a \to (E \to M\ a) \to M\ a \end{array}$$

  - Cuando $M = Cn$ (continuaciones):

$$\begin{array}{l} callcc :: ((M\ a \to r) \to M\ a) \to M\ a \\ abort :: R \to M\ a \end{array}$$

Operaciones, Operaciones y Operaciones

- Las que hacen el trabajo interesante son las operaciones!

$$\begin{aligned} eval\ (Div\ t\ u) = {} & eval\ t \gg\!= \lambda x \rightarrow \\ & eval\ u \gg\!= \lambda y \rightarrow \\ & \mathbf{if}\ y \equiv 0\ \mathbf{then}\ throw\ \texttt{"Division by 0"} \\ & \qquad\quad \mathbf{else}\ return\ (x\ `div`\ y) \end{aligned}$$

- *eval* está definido para *cualquier* mónada que implemente *throw*.
- En general, los programas monádicos están definidos para cualquier mónada *que implementa las operaciones requeridas*.

Mónadas con operaciones

- Podemos expresar esto en Haskell con una clase de tipo

$$\textbf{class } \mathit{Monad}\ m \Rightarrow \mathit{MonadconThrow}\ m\ \textbf{where}$$
$$\quad \mathit{throw} :: E \rightarrow m\ a$$

- El tipo del evaluador queda:

$$\mathit{eval} :: \mathit{MonadconThrow}\ m \Rightarrow \mathit{Lang} \rightarrow m\ \mathit{Int}$$

- Diferentes mónadas pueden ser instancia de *MonadconThrow*.

$$\textbf{instance } \mathit{MonadconThrow}\ \mathit{Ex}\ \textbf{where}$$
$$\quad \mathit{throw}\ e = \mathit{Left}\ e$$

¿Cómo combinar efectos?

- Supongamos que necesitamos una mónada M que implemente las operaciones de estado y excepción.
- Entonces nos podemos preguntar:
  - ¿Podremos estructurar la mónada M como una combinación de efectos mas simples?
  - ¿Hay una sola forma de combinar estado y excepciones?
  - ¿Podemos combinar cualquier mónada sistemáticamente?

Transformadores de Mónadas

- Un transformador de mónada toma una mónada y le agrega un efecto computacional.
- Permite agregar efectos incrementalmente.
- $t :: (* \to *) \to (* \to *)$
- Posee una operación *lift* que levanta una computación de la mónada base a la mónada transformada.

  $\textbf{class}\ (Monad\ m, Monad\ (t\ m)) \Rightarrow MonadT\ t\ \textbf{where}$
  $\quad lift :: m\ a \to t\ m\ a$

- Ejemplos:
  - $\textbf{type}\ \mathcal{S}\ m\ a = S \to m\ (a, S)$
  - $\textbf{type}\ \mathcal{X}\ m\ a = m\ (Either\ E\ a)$
  - $\textbf{type}\ \mathcal{C}\ m\ a = (a \to m\ R) \to m\ R$

Transformadores de Mónadas: algunas respuestas

- ¿Podremos estructurar la mónada M como una combinación de efectos más simples?
  - Podemos empezar con cualquier mónada y agregar *capas* de efectos a gusto.

- ¿Hay una sola forma de combinar estado y excepciones?
  - No, transformar la mónada de excepciones con el transformador de estado no es lo mismo que transformar la mónada de estado con el transformador de excepciones.
    $S \to Either\ E\ (a, S) \qquad vs. \qquad S \to (Either\ E\ a, s)$

- ¿Podemos combinar cualquier mónada sistemáticamente?
  - Punto flojo de los transformadores. El tensor y la suma de teorías algebraicas (Hyland, Plotkin & Power) explican algunos de ellos.

Operaciones de los Transformadores

- Los transformadores agregan efectos, por lo que también deben agregar operaciones.
- La mónada que se obtiene de usar el transformador de estado debe implementar *get* y *put*.
- Uno esperaría que la mónada $\mathcal{S}$ *Ex* implemente *get*, *put*, y también *throw* y *handle*.
- En este ejemplo, *get* y *put* están definidas para $\mathcal{S}\ m$ (y en particular para $\mathcal{S}$ *Ex*).
- Pero $\mathit{throw} :: \mathit{Ex}\ a$ y $\mathit{handle} :: \mathit{Ex}\ a \to (E \to \mathit{Ex}\ a) \to \mathit{Ex}\ a$ están definidas para *Ex a*.

¿Qué es levantar una operación?

- Levantar una operación σ de una mónada m a través de un transformador T es una operación $\hat{\sigma}$ cuyo tipo puede ser derivado substituyendo todas las ocurrencias de m en el tipo de σ por $T\ m$.
- El criterio básico de correción es que un programa que no usa los efectos agregados por el transformador se debe comportar de la misma manera luego de la aplicación del transformador.

Levantando operaciones

- ¿Cómo levantar una operación de la mónada subyacente a la mónada transformada?
- Liang, Hudak y Jones (1995) propusieron

  > *“Proveer una operación levantada para cada par operación/transformador”*

- Problemas:
  - Demasiadas definiciones! $\mathcal{O}(|t| \times |op|)$
  - La forma de levantar una operación a través de un transformador T_1 no está relacionada con la forma de levantar esa operación a través de T_2.
  - La forma en que dos operaciones σ_1 y σ_2 se levantan a través de un mnismo transformador no tiene por qué ser coherente.

Nuestro enfoque

Identificar clases de operaciones y clases de transformadores para los cuales podemos levantar operaciones uniformemente

Operaciones Algebraicas

- Cada transformador T provee una función polimórfica

$$\mathit{lift} :: \mathit{Monad}\ m \Rightarrow m\ a \rightarrow T\ m\ a$$

- Si la operación σ de una mónada M es de la forma $A \rightarrow M\ B$, se puede levantar fácilmente.

$$A \xrightarrow{\sigma} M\ B \xrightarrow{\mathit{lift}} T\ M\ B$$

- Estas operaciones se denominan *algebraicas* ya que existe un iso entre ellas y conjuntos de A operaciones de aridad B (que preservan la multiplicación de M)

$$A \rightarrow M\ B \quad \cong \quad A \times (M\ x)^B \rightarrow M\ x$$

Algebraicas:

- *get*, *set*, *throw*, y *abort*

No algebraicas:

- *handle* y *callcc*

Generalizando las operaciones algebraicas

- Generalizamos las operaciones algebraicas, generalizando la aridad de las operaciones a un functor Σ cualquiera

$$A \times (M\ x)^B \to M\ x \qquad \to \qquad \Sigma\ (M\ x) \to M\ x$$

Ejemplos de Σ-operaciones:

$get :: (S \to St\ a) \to St\ a$	$\Sigma\ x = (S \to x)$
$set :: (S \times St\ a) \to St\ a$	$\Sigma\ x = S \times x$
$throw :: () \to Ex\ a$	$\Sigma\ x = ()$
$handle :: Ex\ a \to (E \to Ex\ a) \to Ex\ a$	$\Sigma\ x = x \times (E \to X)$
$callcc :: ((Co\ a \to R) \to Co\ a) \to Co\ a$	$\Sigma\ x = (x \to R) \to x$
$abort :: R \to Co\ a$	$\Sigma\ x = R$

Operaciones con buen comportamiento

Definición (Σ-operaciones algebraicas para una mónada M)

Es una Σ-operación $op :: \Sigma\ (M\ x) \to M\ x$ *tal que*

$$
\begin{tikzcd}
\Sigma\ (M\ (M\ x)) \arrow[r, "fmap\ join"] \arrow[d, "op_{(M\ x)}"'] & \Sigma\ (M\ x) \arrow[d, "op_x"] \\
M\ (M\ x) \arrow[r, "join"'] & M\ x
\end{tikzcd}
$$

- Tienen una propiedad análoga a las algebraicas:

$$\Sigma\ (M\ x) \to_{alg} M\ x \quad \cong \quad \Sigma\ x \to M\ x$$

- Esto significa que podemos levantar estas operaciones fácilmente!

$$\Sigma\ x \xrightarrow{op} M\ x \xrightarrow{lift} T\ M\ x$$

Ejemplos de Σ-operaciones algebraicas

- Todas las operaciones algebraicas son Σ-operaciones algebraicas.
  Ejemplos:
  - *get* y *put*
  - *throw*
  - *abort*
- Sorprendentemente *callcc* es una Σ-operación algebraica (y por lo tanto es fácil de levantar!)
- La versión de *callcc* que aparece en la MTL se puede *derivar* de nuestra versión:

$$\begin{aligned} &callccMTL \quad :: ((a \to Cn\ b) \to Cn\ a) \to Cn\ a \\ &callccMTL\ f = callcc\ (\lambda k \to f\ (\lambda x \to abort\ (k\ (return\ x)))) \end{aligned}$$

- La Σ-operación *handle* no es algebraica.

Transformadores Functoriales

- Para poder levantar operaciones como *handle*, necesitamos mas información acerca del tranformador.
- Los *transformadores functoriales* son una clase de transformadores de mónadas más estructurados.

$$\mathbf{class}\ \mathit{MonadT}\ t \Rightarrow \mathit{FunctorialT}\ t\ \mathbf{where}$$
$$\quad \mathit{tmap} :: (\forall a. m\ a \rightarrow n\ a) \rightarrow t\ m\ a \rightarrow t\ n\ a$$

Functoriales:

- Transformador de Estado
- Transformador de Excepciones

No Functoriales:

- Transformador de Continuaciones

Levantando Σ-operaciones

Teorema

Dados

- $op :: \Sigma\ (M\ a) \rightarrow M\ a$
- *Transformador Functorial* T.

Existe una operación $op^T :: \Sigma\ (T\ M\ a) \rightarrow T\ M\ a$ *que levanta a* op.

- La construcción de op^T se puede consultar en
  - "Modular Monad Transformers"
    M. Jaskelioff, ESOP 2009.
  - "Monad Transformers as Monoid Transformers"
    M. Jaskelioff y E. Moggi, TCS 2010.
- Si op es una Σ-operación algebraica, las dos maneras vistas de levantar la operación coinciden.

Abstrayendo un poco

- Toda la teoría funciona a un nivel más general (monoides en una categoría monoidal)
  - Las mónadas son una instancia de estos.
  - Para el último teorema visto la categoría debe ser cerrada a derecha.
- Trabajo futuro: Aplicar la teoría a otras estructuras.
  - Por ejemplo, Arrows
- La abstracción hizo evidente otra clase de transformadores: Los functores monoidales, a traves de los cuales se pueden levantar operaciones más generales.

Resumen

- Levantamiento de operaciones en forma uniforme:

Σ-operaqción algebraica	Cualquier morfismo de mónadas
Σ-operación	Transformador Functorial

- **Monatron**: Biblioteca de Transformadores de Mónadas.

- Trabajo Futuro
  - Encontrar otras formas de levantar operaciones.
  - Encontrar un lifting general para teorías predicativas.
  - Extender los resultados a Arrows.

Especificación de operación levantada

Definition

$$
\begin{tikzcd}
\Sigma\ (T\ M\ a) \arrow[r, "op_a^T"] & N\ a \\
\Sigma\ (M\ a) \arrow[u, "fmap\ lift"] \arrow[r, "op_a"'] & M\ a \arrow[u, "lift"']
\end{tikzcd}
$$

Transformador de Codensidad

- $\mathcal{K}\ M\ x = \forall y.(x \to M\ y) \to M\ y$ es un transformador.
  - $\mathit{lift}^{\mathcal{K}} :: M\ a \to \mathcal{K}\ M\ a$

- Se puede definir en sistemas impredicativos como $F\omega$ (y en Haskell).

- Propiedades de $\mathcal{K}$:
  - Toda Σ-operación de M da lugar a una Σ-operación algebraica de $\mathcal{K}\ M$

$$\frac{\Sigma\ (M\ a) \xrightarrow{op} M\ a}{\Sigma\ (\mathcal{K}\ M\ a) \xrightarrow{op^{\mathcal{K}}}_{alg} \mathcal{K}\ M\ a}$$

  - $\mathit{from} :: \mathcal{K}\ M\ a \to M\ a$, tal que $\mathit{from} \circ \mathit{lift}^{\mathcal{K}} = \mathit{id}$.

  - $op = \Sigma\ (M\ a) \xrightarrow{\Sigma(\mathit{lift}^{\mathcal{K}})} \Sigma\ (\mathcal{K}\ M\ a) \xrightarrow{op^{\mathcal{K}}} \mathcal{K}\ M\ a \xrightarrow{\mathit{from}} M\ a$.