

# Construcción Modular de Módulos con Operaciones

Mauro Jaskelioff

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

JCC 2010

- 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  $m :: * \rightarrow *$

```
class Monad m where
```

```
  return :: a  $\rightarrow$  m a
```

```
  (>>=) :: m a  $\rightarrow$  (a  $\rightarrow$  m b)  $\rightarrow$  m b
```

- Algunas mónadas son:

- Computaciones puras
- Excepciones
- Estado
- Continuaciones

```
type Id a = a
```

```
type Ex a = Either E a
```

```
type St a = S  $\rightarrow$  (a, S)
```

```
type Cn a = (a  $\rightarrow$  R)  $\rightarrow$  R
```



# 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):

$$get :: () \rightarrow M S$$
$$put :: S \rightarrow M ()$$

- Cuando  $M = Ex$  (excepciones):

$$throw :: E \rightarrow M a$$
$$handle :: M a \rightarrow (E \rightarrow M a) \rightarrow M a$$

- Cuando  $M = Cn$  (continuaciones):

$$callcc :: ((M a \rightarrow r) \rightarrow M a) \rightarrow M a$$
$$abort :: R \rightarrow M a$$

- Las que hacen el trabajo interesante son las operaciones!

```
eval (Div t u) = eval t >>= \x →  
  eval u >>= \y →  
    if y ≡ 0 then throw "Division by 0"  
    else return (x 'div' y)
```

- 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

```
class Monad m  $\Rightarrow$  MonadconThrow m where  
  throw :: E  $\rightarrow$  m a
```

- El tipo del evaluador queda:

```
eval :: MonadconThrow m  $\Rightarrow$  Lang  $\rightarrow$  m Int
```

- Diferentes mónadas pueden ser instancia de *MonadconThrow*.

```
instance MonadconThrow Ex where  
  throw e = 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.
- $t :: (* \rightarrow *) \rightarrow (* \rightarrow *)$
- Posee una operación *lift* que levanta una computación de la mónada base a la mónada transformada.

```
class (Monad m, Monad (t m))  $\Rightarrow$  MonadT t where  
  lift :: m a  $\rightarrow$  t m a
```

- Ejemplos:
  - **type** S m a = S  $\rightarrow$  m (a, S)
  - **type** X m a = m (Either E a)
  - **type** C m a = (a  $\rightarrow$  m R)  $\rightarrow$  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.  
 $S \rightarrow \text{Either } E (a, S)$     vs.     $S \rightarrow (\text{Either } E a, s)$
- ¿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  $S\ Ex$  implemente *get*, *put*, y también *throw* y *handle*.
- En este ejemplo, *get* y *put* están definidas para  $S\ m$  (y en particular para  $S\ Ex$ ).
- Pero  $throw :: Ex\ a$  y  $handle :: Ex\ a \rightarrow (E \rightarrow Ex\ a) \rightarrow Ex\ a$  están definidas para  $Ex\ a$ .

# ¿Qué es levantar una operación?

- Levantar una operación  $\sigma$  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  $\sigma$  por  $T m$ .
- El criterio básico de corrección 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  $\sigma_1$  y  $\sigma_2$  se levantan a través de un mismo 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

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

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

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

- 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  $M$ )

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

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  $\Sigma$  cualquiera

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

Ejemplos de  $\Sigma$ -operaciones:

*get* ::  $(S \rightarrow St a) \rightarrow St a$

*set* ::  $(S \times St a) \rightarrow St a$

*throw* ::  $() \rightarrow Ex a$

*handle* ::  $Ex a \rightarrow (E \rightarrow Ex a) \rightarrow Ex a$

*callcc* ::  $((Co a \rightarrow R) \rightarrow Co a) \rightarrow Co a$

*abort* ::  $R \rightarrow Co a$

$\Sigma x = (S \rightarrow x)$

$\Sigma x = S \times x$

$\Sigma x = ()$

$\Sigma x = x \times (E \rightarrow X)$

$\Sigma x = (x \rightarrow R) \rightarrow x$

$\Sigma x = R$

# Operaciones con buen comportamiento

Definición ( $\Sigma$ -operaciones algebraicas para una mónada  $M$ )

Es una  $\Sigma$ -operación  $op :: \Sigma (M x) \rightarrow M x$  tal que

$$\begin{array}{ccc} \Sigma (M (M x)) & \xrightarrow{\text{fmap join}} & \Sigma (M x) \\ \text{op}_{(M x)} \downarrow & & \downarrow \text{op}_x \\ M (M x) & \xrightarrow{\text{join}} & M x \end{array}$$

- Tienen una propiedad análoga a las algebraicas:

$$\Sigma (M x) \rightarrow_{alg} M x \cong \Sigma x \rightarrow M x$$

- Esto significa que podemos levantar estas operaciones fácilmente!

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

# Ejemplos de $\Sigma$ -operaciones algebraicas

- Todas las operaciones algebraicas son  $\Sigma$ -operaciones algebraicas.

Ejemplos:

- *get* y *put*
- *throw*
- *abort*
- Sorprendentemente *callcc* es una  $\Sigma$ -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:

$$\begin{aligned} \text{callcc}_{MTL} &:: ((a \rightarrow Cn\ b) \rightarrow Cn\ a) \rightarrow Cn\ a \\ \text{callcc}_{MTL}\ f &= \text{callcc}\ (\lambda k \rightarrow f\ (\lambda x \rightarrow \text{abort}\ (k\ (\text{return}\ x)))) \end{aligned}$$

- La  $\Sigma$ -operación *handle* no es algebraica.

# Transformadores Functoriales

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

```
class MonadT t  $\Rightarrow$  FunctorialT t where  
  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

## 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  $\Sigma$ -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 funtores monoidales, a través de los cuales se pueden levantar operaciones más generales.

- Levantamiento de operaciones en forma uniforme:

$\Sigma$ -operación algebraica	Cualquier morfismo de mónadas
$\Sigma$ -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

$$\begin{array}{ccc} \Sigma (T M a) & \xrightarrow{op_a^T} & N a \\ \text{fmap lift} \uparrow & & \uparrow \text{lift} \\ \Sigma (M a) & \xrightarrow{op_a} & M a \end{array}$$

# Transformador de Codensidad

- $\mathcal{K} M x = \forall y. (x \rightarrow M y) \rightarrow M y$  es un transformador.
  - $lift^{\mathcal{K}} :: M a \rightarrow \mathcal{K} M a$
- Se puede definir en sistemas impredicativos como  $F\omega$  (y en Haskell).
- Propiedades de  $\mathcal{K}$ :
  - Toda  $\Sigma$ -operación de  $M$  da lugar a una  $\Sigma$ -operación algebraica de  $\mathcal{K} M$

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

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

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