Razonando acerca de programas funcionales

Elisabet Lobo Vesga
Asesor: Andrés Sicard Ramírez

23 de noviembre de 2012

Proyecto

Objetivo

Indagar sobre programas funcionales enfocando esfuerzos en la posibilidad de razonar ecuacionalmente sobre funciones monádicas.

Justificación

En computación, no ha sido posible crear un enfoque aceptado para usar el razonamiento ecuacional sobre programas con funciones monádicas debido a su estructura imperativa, sin embargo el lograrlo supondría un gran avance en la verificación y depuración de este tipo de programas.

Antecedentes

- Moggi[1] es el primero en usar las mónadas para estructurar programas.
- Wadler[2] presenta la adaptación a los lenguajes funcionales.
- Hutton y Fulger[3] aplican razonamiento ecuacional sobre funciones transformadoras de estado.
- Gibbons y Hinze[4] plantean una aproximación axiomática para razonar ecuacionalmente sobre programas con componentes monádicas.

[1] *Notions of computation and monads (1991)*
[2] *Monads for functional programming (1992)*
[3] *Reasoning about effects: Seeing the wood through the trees (2008)*
[4] *Just do it: Simple Monadic Equational Reasoning (2011)*

Conceptos

Mónada

En Haskell, una mónada es representada como un objeto de tipo $M\ a$ resultante de encapsular un cómputo de tipo a mediante un constructor genérico M. Las mónadas son entonces constructores de tipos que, dada una entrada, la convierten en un nuevo tipo.

class *Monad m* **where**

$return :: a \to m\ a$

$(\gg=) :: m\ a \to (a \to m\ b) \to m\ b$

$(\gg) :: m\ a \to m\ b \to m\ b$

$fail :: String \to m\ a$

-- Mínimo a completar: *return*, *bind*

$mx \gg my = mx \gg= \lambda\_ \to my$

$fail\ s = error\ s$

Conceptos

Razonamiento sobre programas

El razonamiento ecuacional se basa en la sustitución de términos equivalentes tal cómo se plantea en álgebra básica, es decir, dadas unas propiedades, demostrar determinada igualdad.

$$change :: (a, b) \to (b, a)$$
$$change\ (x, y) = (y, x)$$

Demostrar:

$$change\ (change\ (m, n)) = (m, n)$$

Prueba:

$$\begin{aligned} & change\ (change\ (m, n)) \\ & = change\ (n, m) && \text{-- [[aplicando } change \text{]]} \\ & = (m, n) && \text{-- [[aplicando } change \text{]]} \end{aligned}$$

Razonando sobre programas monádicos

class *Monad m* $\Rightarrow$ *MonadCount m* **where**
  tick :: *m* ()

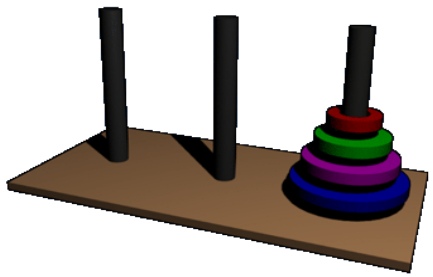

Figura 1: Towers of Hanoi

$$
\begin{aligned}
&\mathit{hanoi} :: \mathit{MonadCount}\ m \Rightarrow \mathit{Int} \to m\ () \\
&\mathit{hanoi}\ 0 \qquad\quad = \mathit{skip} \\
&\mathit{hanoi}\ (n+1) = \mathit{hanoi}\ n \gg \mathit{tick} \gg \mathit{hanoi}\ n
\end{aligned}
$$

Razonando sobre programas monádicos

$$
\begin{aligned}
&\mathit{rep} :: \mathit{Monad}\ m \Rightarrow \mathit{Int} \to m\ () \to m\ ()\\
&\mathit{rep}\ 0 \qquad\quad\ \ \mathit{mx} = \mathit{skip}\\
&\mathit{rep}\ (n+1)\ \mathit{mx} = \mathit{mx} \gg \mathit{rep}\ n\ \mathit{mx}
\end{aligned}
$$

Propiedades:

$$
\begin{aligned}
&\mathit{rep}\ 1 \qquad\quad\ \ \mathit{mx} = \mathit{mx}\\
&\mathit{rep}\ (m+n)\ \mathit{mx} = \mathit{rep}\ m\ \mathit{mx} \gg \mathit{rep}\ n\ \mathit{mx}
\end{aligned}
$$

Razonando sobre programas monádicos

Demostrar:

$$hanoi\ n = rep\ (2^n - 1)\ tick$$

Prueba:

$hanoi\ (n+1)$

-- [[definición de hanoi]]

$= hanoi\ n \gg tick \gg hanoi\ n$

-- [[hipótesis inductiva]]

$= rep\ (2^n - 1)\ tick \gg tick \gg rep\ (2^n - 1)\ tick$

-- [[propiedad de rep]]

$= rep\ ((2^n - 1) + 1 + (2^n - 1))\ tick$

-- [[aritmética]]

$= rep\ (2^{n+1} - 1)\ tick$

Mónada Probabilística

$$\textbf{class } \mathit{Monad}\ m \Rightarrow \mathit{MonadProb}\ m \textbf{ where}$$
$$\quad \mathit{choice} :: \mathit{Prob} \rightarrow m\ a \rightarrow m\ a \rightarrow m\ a$$

Empleando esta mónada, se pueden definir funciones que generen distribuciones, por ejemplo:

$$\mathit{uniform} :: \mathit{MonadProb}\ m \Rightarrow [a] \rightarrow m\ a$$
$$\mathit{uniform}\ [x] \quad = \mathit{return}\ x$$
$$\mathit{uniform}\ (x : xs) =$$
$$\quad \mathit{choice}\ (1\ /\ \mathit{length}\ (x : xs))\ (\mathit{return}\ x)\ (\mathit{uniform}\ xs)$$

Monty Hall

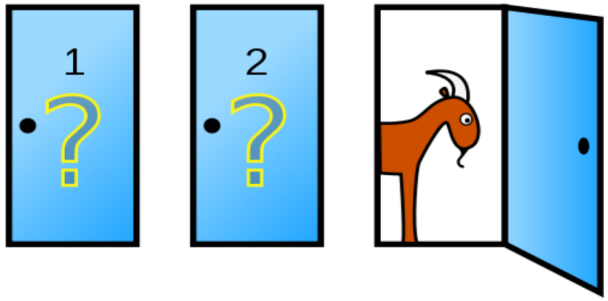

Figura 2: The Monty Hall Problem

$$
\begin{aligned}
&\mathit{play} :: \mathit{MonadProb}\ m \Rightarrow (\mathit{Door} \to \mathit{Door} \to m\ \mathit{Door}) \to m\ \mathit{Bool} \\
&\mathit{play}\ \mathit{strategy} = \mathbf{do} \\
&\quad h \leftarrow \mathit{hide} \\
&\quad p \leftarrow \mathit{pick} \\
&\quad t \leftarrow \mathit{tease}\ h\ p \\
&\quad s \leftarrow \mathit{strategy}\ p\ t \\
&\quad \mathit{return}\ (s \equiv h)
\end{aligned}
$$

Resultados

Razonando ecuacionalmente sobre este juego se obtiene:

$$\begin{aligned} play\ switch &= uniform\ [True, True, False] \\ play\ stick &= uniform\ [False, False, True] \end{aligned}$$

Ahora bien, bajo simulaciones los resultados son:

Bool	Switch	Stick
True	65 %	30 %
False	35 %	70 %

Conclusión

Existe la posibilidad de efectuar el razonamiento ecuacional sobre programas con funciones monádicas, suponiendo así, que es posible realizar de una manera más simple la verificación de programas con estos componentes.