Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencia de la Computación
Análisis de Lenguajes de Programación

Trabajo práctico 4 - Programación monádica

1 Introducción

El objetivo de este trabajo práctico es familiarizarse con la escritura de intérpretes mediante el uso de mónadas. La base de este trabajo práctico serán los evaluadores del trabajo práctico 1.
El trabajo se debe realizar en grupos de dos personas y la fecha límite de entrega es el 14 de Noviembre, en donde se debe entregar:

- en papel, un informe con los ejercicios resueltos incluyendo todo el código que haya escrito;
- en forma electrónica (en un archivo comprimido) el código fuente de los programas y el informe, usando el sitio de la materia del campus virtual de la UNR (`http://comunidades.campusvirtualunr.edu.ar`).

2 Intérpretes monádicos

En la carpeta `src/` se encuentran los archivos correspondientes al intérprete del trabajo práctico 1. Al igual que en el primer trabajo, las tres etapas de evaluadores están divididas en los archivos `Eval1.hs`, `Eval2.hs`, y `Eval3.hs`.

2.1 Evaluador simple

El evaluador simple se encuentra parcialmente implementado en `src/Eval1.hs`. El estado del programa se representa mediante el tipo de datos Env, que es simplemente una lista de pares de nombres de variable y sus respectivos valores. Se utiliza una mónada de estado, llamada State, para representar una computación que tiene acceso al estado del programa:

$$
\begin{aligned}
&\textbf{newtype}\ \mathsf{State}\ a = \mathsf{State}\ \{\mathsf{runState} :: \mathsf{Env} \to (a, \mathsf{Env})\}\\
&\textbf{instance}\ \mathsf{Monad}\ \mathsf{State}\ \textbf{where}\\
&\quad \mathsf{return}\ x = \mathsf{State}\ (\lambda s \to (x, s))\\
&\quad m \gg\!= f = \mathsf{State}\ (\lambda s \to \textbf{let}\ (v, s') = \mathsf{runState}\ m\ s\\
&\qquad\qquad\qquad\qquad\ \ \textbf{in}\ \mathsf{runState}\ (f\ v)\ s')
\end{aligned}
$$

Una computación con estado es una función que recibe un estado original, computa algún valor y retorna un nuevo estado. Notar que en este caso no alcanza con la mónada Reader, ya que al ejecutar una asignación necesitamos modificar el entorno.
La clase MonadState tiene las operaciones necesarias a implementar en mónadas con posibilidad de manejar variables con valores enteros.

$$
\begin{aligned}
&\textbf{class}\ \mathsf{Monad}\ m \Rightarrow \mathsf{MonadState}\ m\ \textbf{where}\\
&\quad \mathsf{lookfor} :: \mathsf{Variable} \to m\ \mathsf{Int}\\
&\quad \mathsf{update} :: \mathsf{Variable} \to \mathsf{Int} \to m\ ()
\end{aligned}
$$

En el código se encuentra una instancia de estas operaciones para la mónada State.

Ejercicio 1. Completar el evaluador simple:

a) Demostrar que State es efectivamente una mónada.

b) Implementar el evaluador utilizando la mónada State.

2.2 Evaluador con manipulación de errores

Este evaluador se deberá implementar en el archivo `src/Eval2.hs`. Esta versión del evaluador podrá controlar los errores de división por cero, pero a diferencia del primer trabajo práctico, se utilizará una estructura monádica para manipular el error. La mónada utilizada para representar computaciones con estado de variables y posibilidad de error será:

newtype StateError a = StateError { runStateError :: Env → Maybe $(a$, Env$)$ }

Con esta nueva definición podremos marcar errores devolviendo un Nothing. Agregamos además una clase MonadError para representar las operaciones de aquellas mónadas que pueden producir errores.

class Monad $m \Rightarrow$ MonadError m **where**
  throw :: $m\ a$

Ejercicio 2. Completar el evaluador con manipulación de errores:

a) Dar una instancia de Monad para StateError.

b) Dar una instancia de MonadError para StateError.

c) Dar una instancia de MonadState para StateError.

d) Implementar el evaluador utilizando la mónada StateError.

2.3 Evaluador con análisis de costo

Este evaluador se deberá implementar en el archivo `src/Eval3.hs`. En esta versión del evaluador, se deberá calcular el costo de las operaciones efectuadas, según se indica a continuación:

$$\begin{array}{rcll} \mathrm{W}(e_1\ nop\ e_2) & = & 2 + \mathrm{W}(e_1) + \mathrm{W}(e_2) & \text{donde } nop \in \{\div, \times\} \\ \mathrm{W}(e_1\ bop\ e_2) & = & 1 + \mathrm{W}(e_1) + \mathrm{W}(e_2) & \text{donde } bop \in \{+, -_b, \wedge, \vee, >, <, ==, \neq\} \\ \mathrm{W}(op\ e) & = & 1 + \mathrm{W}(e) & \text{donde } op \in \{-_u, \neg\} \end{array}$$

Para esto, deberá proponer una modificación de la mónada StateError, de manera que también lleve un entero para almacenar el costo de las operaciones.

Ejercicio 3. Completar el evaluador con análisis de costo:

a) Proponer una nueva mónada que lleve el costo de las operaciones efectuadas en la computación, además de manejar errores y estado. Llámela StateErrorCost.

b) Dar una clase que provea las operaciones necesarias para llevar el costo de las operaciones efectuadas. Llámela MonadCost.

c) Dar una instancia de MonadCost para StateErrorCost.

d) Dar una instancia de MonadError para StateErrorCost.

e) Dar una instancia de MonadState para StateErrorCost.

f) Implementar el evaluador utilizando la mónada StateErrorCost.