



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 30 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, 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:

```
newtype State a = State {runState :: Env → (a, Env)}  
instance Monad State where  
  return x = State (λs → (x, s))  
  m >>= f = State (λs → let (v, s') = runState m s  
                        in runState (f v) s')
```

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.

```
class Monad m => MonadState m where  
  lookfor :: Variable → m Int  
  update :: Variable → Int → m ()
```

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 trabajar en el archivo `src/Eval2.hs`. Como en el trabajo práctico anterior, la segunda versión del evaluador podrá controlar los errores de división por cero. Esta vez 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 ⇒ 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 cuenta de operaciones

Este evaluador se deberá trabajar en el archivo `src/Eval3.hs`. En esta versión del evaluador, se deberán contar la cantidad de operaciones `+`, `-`, `*` y `/` llevadas a cabo durante la computación. Para esto, deberá proponer una modificación de la mónada `StateError` que lleve un entero donde se cuenten las operaciones aritméticas.

Ejercicio 3. Completar el evaluador con cuenta de operaciones:

- a) Proponga una nueva mónada que lleve la cantidad de operaciones (además de manejar errores y estado). Llámela `StateErrorTick`.
- b) Dar una clase que provea las operaciones necesarias para llevar la cantidad de cuentas realizadas, llame a esta clase `MonadTick`.
- c) Dar una instancia de `MonadTick` para `StateErrorTick`.
- d) Dar una instancia de `MonadError` para `StateErrorTick`.
- e) Dar una instancia de `MonadState` para `StateErrorTick`.
- f) Implementar el evaluador utilizando la mónada `StateErrorTick`.