



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 hasta dos personas y la fecha límite de entrega es el 1 de Diciembre, en donde se debe entregar: un archivo comprimido del directorio `src`, usando el sitio de la materia del campus virtual de la UNR (<http://comunidades.campusvirtualunr.edu.ar>).

Como complemento a esta consigna, el trabajo práctico incluye un `README` con aclaraciones técnicas sobre la implementación propuesta. Se recomienda leer este documento con atención y consultar a algún docente cualquier duda.

2 Intérpretes monádicos

En la carpeta `TP4` 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 `src/Eval1.hs`, `src/Eval2.hs`, y `src/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 una asociación 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 → Pair 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`, definida en `src/Monads.hs`, 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 `src/Eval1.hs` se encuentra una instancia de estas operaciones para la mónada `State`.

Ejercicio 1. Completar el evaluador simple:

- Demostrar que `State` es efectivamente una mónada.
- Implementar en `src/Eval1.hs` 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 y variables no definidas. 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 → Either Error (a, Env)}
```

En este, el tipo `Error` corresponde a los posibles errores de un programa, y se encuentra definido en `src/AST.hs` como:

```
data Error = DivByZero | UndefVar deriving (Eq, Show)
```

Con la definición de `StateError` podremos marcar que ocurre un error e devolviendo `Left e`. Agregamos además una clase `MonadError` (en `src/Monads.hs`) para representar las operaciones de aquellas mónadas que pueden producir errores.

```
class Monad m ⇒ MonadError m where
  throw :: Error → m a
```

Ejercicio 2. Completar el evaluador con manipulación de errores:

- Dar en `src/Eval2.hs` una instancia de `Monad` para `StateError`.
- Dar en `src/Eval2.hs` una instancia de `MonadError` para `StateError`.
- Dar en `src/Eval2.hs` una instancia de `MonadState` para `StateError`.
- Implementar en `src/Eval2.hs` el evaluador utilizando la mónada `StateError`.

2.3 Evaluador con traza de ejecución

Este evaluador se deberá trabajar en el archivo `src/Eval3.hs`. En esta versión del evaluador, se deberá llevar el registro de la traza de ejecución.

Ejercicio 3. Completar el evaluador con traza de ejecución::

- Proponga en `src/Eval3.hs` una nueva mónada que lleve una traza de ejecución (además de manejar errores y estado) y de su instancia de mónada. Llámela `StateErrorTrace`.
- Dar en `src/Monads.hs` una clase que provea las operaciones necesarias para llevar una traza, llame a esta clase `MonadTrace`.
- Dar en `src/Eval3.hs` una instancia de `MonadTrace` para `StateErrorTrace`.
- Dar en `src/Eval3.hs` una instancia de `MonadError` para `StateErrorTrace`.
- Dar en `src/Eval3.hs` una instancia de `MonadState` para `StateErrorTrace`.
- Implementar en `src/Eval3.hs` el evaluador utilizando la mónada `StateErrorTrace`.