CI4251 - Programación Funcional Avanzada
Tareas 5

Ernesto Hernández-Novich
86-17791
<emhn@usb.ve>

Julio 2, 2013

1. Expresiones y sus derivadas

Considere el siguiente subconjunto de las expresiones que hemos usado en las últimas tareas:

```
data Exp = Const Int
         | Var String
         | Add Exp Exp
         | Mul Exp Exp
         deriving (Eq,Show)

eval (Const i) = i
eval (Var n)   = fromJust (DM.lookup n env)
eval (Add l r) = (eval l) + (eval r)
eval (Mul l r) = (eval l) * (eval r)

env = DM.fromList [ ("foo",42), ("bar",69),
                    ("baz",27), ("qux",0) ]
```

1.1. Reconocedor con Parsec (3 puntos)

Aproveche la librería `Text.Parsec` para escribir un reconocedor predictivo, posiblemente con *backtracking*, capaz de procesar una cadena correpondiente a una expresión aritmética, produciendo la representación abstracta de la misma usando el tipo de datos `Exp`

```
parseExp :: String -> Either ParseError Exp
```

El reconocedor sólo será usado en GHCi, con salida similar a

```
ghci> parseExp "42"
Right (Const 42)
ghci> parseExp "21+3*7"
Right (Add (Const 21) (Mul (Const 3) (Const 7)))
ghci> parseExp "(x+2)*(x+3)"
Right (Mul (Add (Var "x") (Const 2)) (Add (Var "x") (Const 3)))
ghci> parse "2-3"
Left "tarea-05" (line 1, column 2):
unexpected '-'
expecting digit, "*", "+" or end of input
ghci> parseExp "2*"
Left "tarea-05" (line 1, column 3):
unexpected end of input
expecting term
```

Su reconocedor debe utilizar la gramática

$$E \to T + E$$
$$E \to T$$
$$T \to F * T$$
$$T \to F$$
$$F \to (E)$$
$$F \to \textbf{id}$$
$$F \to \textbf{num}$$

que *tiene* prefijos izquierdos comunes, los cuales **deben** mantenerse.

El terminal **id** se define como cualquier secuencia de una o más letras, mientras que el terminal **num** se define como una secuencia de uno o más dígitos.

1.2. Manipulando expresiones (1 punto)

Escriba la función

```
derivada :: Exp -> String -> Exp
```

que calcula la derivada simbólica de una expresión, en relación a una variable particular.

Escriba la función

```
simplifica :: Exp -> Exp
```

que realice simplificaciones básicas sobre las expresiones:

- Suma de constantes han de convertirse en la constante suma.
- Simplificar el elemento neutro de suma y multiplicación.

2. Aprovechando QuickCheck

2.1. Instanciando Arbitrary (2 puntos)

Escriba una instancia de `Arbitrary` para `Exp` tal que permita generar expresiones asegurando que no sean "triviales" y que haya a lo sumo *dos* operaciones idénticas anidadas, i.e. no puede haber `Add (Add (Add ...))` pero si puede haber `Add (Add (Mul (Mul (Add ...))))`.

2.2. Escribiendo propiedades (4 puntos)

- Defina la propiedad `prop_singlevar` para verificar que para toda expresión, la expresión derivada no contiene más variables que la expresión original.
- Defina la propiedad `prop_simplifies` para verificar que evaluar una expresión es lo mismo que evaluar la expresión después de simplificarla.
- Defina la propiedad `prop_cleans` para verificar que después de simplificar una expresión, esta no contiene nada simplificable.
- Defina la propiedad `prop_idempotent` para verificar que evaluar la derivada, es lo mismo que evaluar la simplificación de la derivada.