CI4251 - Programación Funcional Avanzada
Entrega Tarea 5

Brian-Marcial Mendoza
07-41206
<07-41206@usb.ve>

Junio 27, 2015

Un problema de programación dinámica...

Considere una expresión booleana compuesta por una secuencia arbitraria de las palabras reservadas:

- `true`
- `false`
- `and`
- `or`
- `xor`

el problema consiste en determinar de *cuántas* maneras se pueden incorporar paréntesis *explícitos* de modo que la expresión tenga el valor `true`.
Por ejemplo, si se nos proveyera le expresión

```
true xor false and true
```

el algoritmo debería contestar `2`, pues esa expresión sólo se hace cierta si se incorporan los paréntesis

```
((true xor false) and true)
(true xor (false and true))
```

Ud. debe implantar en Haskell una solución a este problema utilizando técnicas de programación dinámica apoyadas en arreglos Haskell. En este sentido, construiremos la solución comenzando con un tipo de datos para representar los símbolos involucrados:

```
import Text.ParserCombinators.Parsec

data Symbol = SymTrue | SymFalse | SymAnd | SymOr | SymXor
            deriving (Show,Eq)
```

Escriba un reconocedor `Parsec` que sea capaz de convertir una expresión construida con los literales, y llevarla a una lista de valores de nuestro tipo algebráico. En este sentido:

- Su reconocedor debe ser capaz de reconocer *varias* expresiones, separadas entre sí por un punto y coma (`;`). Cada expresión puede ocupar una o más líneas, e incluso podría haber más de una expresión en una línea. Pero *todas* terminan con punto y coma.
- Puede haber una cantidad arbitraria de espacios en blanco antes del comienzo de la expresión, entre los literales, antes del punto y coma, y después del punto y coma. Deben ser ignorados.
- Su reconocedor debe rechazar expresiones sintácticamente incorrectas. No es necesario que se recupere de ese error.

Así, la función principal del reconocedor sería

```
expresiones :: Parser [[Symbol]]
expresiones
  = do
    strings <- line `endBy` eol
    let symbols = (map . map) toSymbol strings
    return symbols

line
  = cell `sepBy` spaces

cell
  = do
    spaces
    res <- (try (string "true")
            <|> try (string "false")
            <|> try (string "and")
            <|> try (string "or")
            <|> string "xor"
            <?> help)
    spaces
    return res

help = "true, false, and, or, xor"

eol = do
  spaces
  char ';'
  spaces
```

Escriba entonces la función

```
trueWays :: [Symbol] -> Int
```

```
trueWays sym = 42
```

que calcule la cantidad de parentizaciones que hacen `true` la expresión.
El programa principal debe recibir un nombre de archivo como argumento de línea de comandos, y si existe, aplicar el reconocedor sobre los contenidos de ese archivo e indicar la cantidad de parentizaciones para cada expresión. Sólo debe mostrar la expresión y la cantidad de parentizaciones, pero *no* necesita mostrar las parentizaciones específicas.
La solución para este algoritmo es directa y emplea técnicas de programación dinámica sobre arreglos *mutables*. Ud. puede presentar una solución utilizando arreglos mutables sobre el monad `ST`, pero sepa que es perfectamente posible hacerlo con arreglos *inmutables* si Ud. escribe *thunks* de manera astuta.

```
toSymbol :: String -> Symbol
toSymbol s = case s of
               "true" -> SymTrue
               "false" -> SymFalse
               "and" -> SymAnd
               "or" -> SymOr
               "xor" -> SymXor

toString :: [Symbol] -> String
toString sym = go sym ""
  where
    go [] acc = acc
    go (x:xs) acc = go xs (acc ++ toS x)
    toS s =  case s of
               SymTrue -> " true "
               SymFalse -> " false "
               SymAnd -> " && "
               SymOr -> " || "
               SymXor -> " ^ "

getWays :: [Symbol] -> IO ()
getWays syms
  = putStrLn
  $ "Para la expresion \""
  ++ (toString syms)
  ++ "\" el número de paréntesis explícitos son: "
  ++ show (trueWays syms)

parseFile = parse expresiones

main = do input <- getContents
          case parseFile "(stdin)" input of
```

```
            Left c  -> do putStrLn "Error: "
                          print c
            Right r -> mapM_ getWays r
```