Auxiliar 5

Tipos inductivos, recursión primitiva, inducción estructural

Profesor: Federico Olmedo
Auxiliar: Damián Árquez

Compilación con máquinas abstractas

En este ejercicio explotaremos los tipos inductivos para crear nuestro propio compilador.

1. Para partir, declare un nuevo tipo inductivo `Expr` que represente un valor entero (`Val`), la suma (`Add`) y la resta (`Sub`). Note que cada constructor lleva consigo argumentos. Además, por simplicidad, utilizaremos solo valores `Int`.

2. Escriba una función `eval` que calcule el valor de una expresión utilizando la suma y resta primitiva de haskell.

3. Note que la función `eval` definida anteriormente no permite razonar sobre cual sumando se evalúa antes o después. Por ejemplo, en la expresión `(1 + 2) + (2 + 3)`, ¿Podría usted decir si Haskell evalúa primero `(1 + 2)` o `(2 + 3)`?
   Para obtener una ejecución controlada por nosotros escribiremos una maquina abstracta para evaluar una expresión en el orden que nosotros queramos.
   Primero, declare un nuevo tipo `Stack` como una lista de enteros que nos permitirá guardar el estado de un programa. Luego, declare un nuevo tipo inductivo `Op` que represente las posibles operaciones: `PUSH Int`, `ADD`, `SUB`. La operación `PUSH` apila un valor en el tope del `Stack` y las operaciones `ADD` y `SUB` sacan los dos elementos al tope del stack y aplican la operación correspondiente, apilando el resultado de vuelta en el stack. Finalmente, declare el tipo `Code` como una lista de operaciones.

4. Escriba una función `comp` que tome una expresión y retorne una lista de operaciones (es decir, un `Code`).

Código 1: addPadding

```
> comp (Add (Subs (Val 2) (Val 3)) (Val 4))
  [PUSH 4,PUSH 3,PUSH 2,SUB,ADD]

```

5. Escriba la función `exec` que a partir de un código compilado y un stack de valores, ejecute paso a paso el código, retornando un nuevo stack.

Código 2: addPadding

```
> exec [PUSH 4,PUSH 3,PUSH 2,SUB,ADD] []
  [3]

```

6. Note que del ejercicio anterior se puede observar que la ejecución de nuestro código compilado es equivalente a la evaluación de la expresión por haskell. Es decir, tenemos la ecuación:

   `exec (comp e) [] = [eval e]`.

   Pruebe, utilizando inducción estructural, que dicha ecuación es cierta. **Hint**: Para poder aprovechar la hipótesis inductiva es necesario generalizar aún más la propiedad. En particular, debe generalizarla para cualquier stack, no solo un stack vacío. Si al partir con un stack vacío se termina con un singleton donde el único elemento es el valor final, ¿cómo podría agregar la noción de un stack en el lado derecho de la ecuación?

7. Note que en `eval` y `comp` se presenta el mismo patrón de recursión y *pattern matching*. Dicho patrón es el mismo visto en clases: `fold`. Escriba una función `foldExpr` que abstraiga dicho patrón de recursión primitiva.

8. Reescriba las funciones `evalF` y `compF` análogas a las anteriores pero que hagan uso de `foldExpr`.