Auxiliar 3

Recursión y QuickCheck

Profesor: Federico Olmedo
Auxiliar: Damián Árquez

Transmisión binaria de Strings

Un número en base-10 o notación decimal es una secuencia de dígitos del 0 al 9 en que cada dígito es multiplicado por un factor de 10 mientras nos movemos a la izquierda:

$$2385 = (2 * 1000) + (3 * 100) + (8 * 10) + (5 * 1)$$

En contraste, un número binario es una de bits (0s y 1s) en que cada bit es multiplicado por un factor de 2 mientras nos movemos a la izquierda:

$$1101 = (1 * 8) + (1 * 4) + (0 * 2) + (1 * 1)$$

De esta manera, el número binario 1011 representa al número decimal 13. Para fines prácticos, de esta pregunta, utilizaremos la notación binaria escrita en **en reversa**, es decir, ahora el número 13 será representado por 1011 en vez de 1101:

$$1011 = (1 * 1) + (0 * 2) + (1 * 4) + (1 * 8)$$

Si, por ejemplo, tomamos un número de 4 bits $abcd$, utilizando un poco de álgebra podemos notar que podemos escribir dicha igualdad de una forma más conveniente:

$$\begin{aligned} abcd &= (a * 1) + (b * 2) + (c * 4) + (d * 8) \\ &\quad a + (b * 2) + (c * 4) + (d * 8) \\ &\quad a + 2 * (b + (c * 2) + (d * 4)) \\ &\quad a + 2 * (b + 2 * (c + (d * 2)) \\ &\quad a + 2 * (b + 2 * (c + 2 * (d + 2 * 0))) \end{aligned} \tag{1}$$

Así, notamos que convertir una lista de 4 bits $[a, b, c, d]$ en un entero decimal solo toma sumar cada elemento con el doble de la conversión del resto de la lista, reemplazando la lista vacía con 0 (caso base).

1. Escriba una función `bin2int` que tome una lista de enteros (0s y 1s) y la convierta en el número decimal correspondiente, a través de la propiedad ya mostrada.

2. Escriba una función `int2bin` que haga exactamente lo contrario.

3. Ahora queremos que todos nuestros números binarios tengan el mismo largo. Escriba la función `addPadding` que tome el largo final esperado y una lista de bits, le agregue tantos 0s como haga falta para completar dicho largo. En caso de que la lista de bits sea más larga que el largo esperado debe arrojar un error.

Código 1: addPadding

```
> addPadding 8 [1, 0, 1, 1]
  [1, 0, 1, 1, 0, 0, 0, 0]

```

4. Ya estamos listos para codificar un String arbitrario. Escriba la función `encode` que reciba un String y retorne una lista de bits representado dicho String, codificando cada carácter en 8 bits. *Hint*: utilice la función `ord` que convierte un `Char` en un `Int`, puede utilizar la función `concat` que recibe una lista de listas y las concatena. **Nota**: Para utilizar `ord` debe importar el módulo `Data.Char`, escribiendo `import Data.Char` al inicio de su archivo.

Código 2: encode

```
> encode "abc"
  [1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0]

```

Nota: Note que el resultado tiene 24 bits ya que el string inicial tenía 3 caracteres.

5. Ya que podemos codificar, y teniendo la función `bin2int` a mano, también queremos decodificar. Escriba la función de utilidad `chop` que reciba n el número de bits en que está codificado un String y una lista de bits, y retorne una lista con listas de n bits. *Nota*:

Código 3: chop

```
> chop 8    [1,0,0,0,0,1,1,0,0,1,0,0,0,1,1,0,1,1,0,0,0,1,1,0]
   [[1,0,0,0,0,1,1,0],    [0,1,0,0,0,1,1,0],    [1,1,0,0,0,1,1,0]]

```

6. Escriba la función `decode` que reciba una lista de bits y retorne el string correspondiente a una codificación en 8 bits.

7. Finalmente, podemos definir la siguiente función `transmit`, que simula un canal de comunicación en que se codifica y se decodifica un string:

Código 4: transmit

```
channel :: [Int] -> [Int]
channel = id

transmit :: String -> String
transmit = decode . channel . encode

```

Utilizando QuickCheck escriba un test que verifique que dicha transmisión no ha afectado el mensaje dado. Recuerde que las variables del tipo `Char` en realidad se codifican en 32 bits (no 8), ¿Genera ésto un problema para el test? ¿Es posible arreglar el eventual problema utilizando una pre-condición en el test? ¿Qué pasa si aumenta los bits con los que codifica los strings en su programa? ¿Cómo afecto eso al test que escribió?

QuickCheck

1. Utilizando la librería Quickcheck, escriba un test que verifique la siguiente propiedad:

$$reverse\ (xs + +ys) == reverse\ ys + +reverse\ xs$$

2. Los números de fibonacci son conocidos por tener varias propiedades interesantes. Una de ellas es que al dividir un número fibonacci por su antecesor tiende al número aureo, $\phi = \frac{1+\sqrt{5}}{2}$. Es decir $\frac{fib(n+1)}{fib(n)} \approx \phi$, donde $fib(0) = 1$, $fib(1) = 1$ y $fib(n) = fib(n-1) + fib(n-2)$.

   Escriba un test que verifique que el cuociente entre cualquier número fibonacci y su antecesor no está a más distancia que 0,7 de ϕ. Note que es necesario que los valores que le entregue QuickCheck sean positivos. Utilice una precondición para asegurarse de ello.

3. Escriba un test que demuestre que la relación antes descrita es cada vez más cierta a medida que crece el n. Es decir, la distancia entre el $\frac{fib(n+2)}{fib(n+1)}$ y ϕ es menor o igual que la distancia entre $\frac{fib(n+1)}{fib(n)}$ y ϕ, para cualquier n.

   Nota: Dado lo 'caro' que es computar números de fibonacci muy grande, limite el número máximo de test necesarios a 10 o 15 (como se vió en clases), y utilice como pre-condición que los n no sean mayores a algún valor entre 20 o 30. De lo contrario, es muy probable que su test no termine de computar o le tome mucho tiempo hacerlo.