

Introducción a la programación

HUnit: framework de Testing Unitario para Haskell

Formato de un caso de test en HUnit

"Nombre del test" ~: <res obtenido> ~?= <res esperado>

Donde:

res obtenido es el valor que devuelve la función que queremos testear.

res esperado es el valor que debería devolver la función que queremos testear.

Ejemplos:

"El doble de 4 es 8" ~: (doble 4) ~?= 8

"Maximo repetido" ~: (maximo [2,7,3,7,4]) ~?= 7

"esPar de impar" ~: (esPar 5) ~?= False

Ejercicio

Crear test unitarios para la función `fib: Int -> Int` que devuelve el *i*-ésimo número de Fibonacci.

Considerar la siguiente especificación e implementación en Haskell:

$$fib(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ fib(n-1) + fib(n-2) & \text{en otro caso} \end{cases}$$

```
problema fib (n: ℤ) : ℤ {  
  requiere: { n ≥ 0 }  
  asegura: { res = fib(n) }  
}
```

```
fib :: Int -> Int  
fib 0 = 0  
fib 1 = 1  
fib n = fib (n-1) + fib (n-1)
```

Modularizando el código

- ▶ Crear un módulo llamado `MisFunciones` con las funciones que queremos testear.
- ▶ El nombre del archivo debe coincidir con el nombre del módulo.

```
module MisFunciones where
```

```
fib :: Int -> Int
```

```
fib 0 = 0
```

```
fib 1 = 1
```

```
fib n = fib (n-1) + fib (n-1)
```

Módulo para los tests

- ▶ Crear otro módulo llamado `TestsDeMisFunciones` con los casos de tests (opcional).
- ▶ Ambos archivos deben estar guardados en la misma carpeta.
- ▶ Importar el módulo `Test.HUnit` para poder crear casos de test utilizando `HUnit`.
- ▶ Importar el módulo `MisFunciones` para poder utilizar las funciones allí definidas.
- ▶ Al importar el archivo con los test, se importa automáticamente el archivo con la implementación, ya que definimos `import MisFunciones`.

```
module TestsDeMisFunciones where
```

```
import Test.HUnit
```

```
import MisFunciones
```

— *Casos de test*

Agregando casos de test

- ▶ Un test para cada caso base.
- ▶ Un test para el caso recursivo.

```
module TestsDeMisFunciones where
```

```
import Test.HUnit
```

```
import MisFunciones
```

```
— Casos de test
```

```
run = runTestTT tests
```

```
tests = test [  
    " Caso base 1: fib 0" ~: (fib 0) ~?= 0,  
    " Caso base 2: fib 1" ~: (fib 1) ~?= 1,  
    " Caso recursivo 1: fib 2" ~: (fib 2) ~?= 1  
]
```

Corriendo los casos de test

Para correr los tests en `ghci` debemos:

- ▶ Cargar el archivo `TestsDeMisFunciones.hs`.
- ▶ Evaluar la función `run`.

```
ghci> run
#### Failure in: 2:" Caso recursivo 1: fib 2"
TestsDeMisFunciones.hs:12
expected: 1
  but got: 2
Cases: 3   Tried: 3   Errors: 0   Failures: 1
```

Podemos ver que el ultimo test falló. Por qué?

Corrigiendo el error

Revisemos la implementación de la función `fib`.

```
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-1) — Aca esta el error
```

Debería restar 2 en lugar de 1. Lo corregimos, guardamos y recargamos.

```
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Volvemos a correr los test.

```
ghci> run
Cases: 3   Tried: 3   Errors: 0   Failures: 0
```

Ahora todos los test son exitosos!

Comparando listas

Ejercicio 3.7 Guía 5:

```
problema pares (s: seq<Z>) : seq<Z> {  
  requiere: { True }  
  asegura: { resultado tiene todos los elementos pares de la  
             lista s sin repetidos, en cualquier orden }  
}
```

¿Está bien usar este caso de test?

```
tests = test [  
  "MuchosPares" ~: (pares [2,4,4,6]) ~?= [2,4,6]  
]
```

Comparando listas

Ejercicio 3.7 Guía 5:

problema pares ($s: seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$ {
 requiere: { *True* }
 asegura: { *resultado* tiene todos los elementos pares de la
 lista *s* sin repetidos, en cualquier orden }
}

¿Está bien usar este caso de test?

```
tests = test [  
  "MuchosPares" ~: (pares [2,4,4,6]) ~?= [2,4,6]  
]
```

¿Es [6,4,2] una solución correcta para el problema? ¿Pasaría el caso de test?

Comparando listas

Ejercicio 3.7 Guía 5:

problema pares ($s: seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$ {
 requiere: { *True* }
 asegura: { *resultado* tiene todos los elementos pares de la
 lista *s* sin repetidos, en cualquier orden }
}

¿Está bien usar este caso de test?

```
tests = test [  
  "MuchosPares" ~: (pares [2,4,4,6]) ~?= [2,4,6]  
]
```

¿Es [6,4,2] una solución correcta para el problema? ¿Pasaría el caso de test?

Tenemos que comparar las listas de otra forma:

```
tests = test [  
  "MuchosPares" ~: (sonIguales (pares [2,4,4,6])  
                                  [2,4,6]) ~?= True  
]
```

Funciones útiles para el TP

Motivación

```
problema indiceMax (s: seq<Z>) : Z {  
  requiere: { True }  
  asegura: { resultado es igual al índice del mayor elemento en  
             s. En caso de tener repetidos, devolver cualquier índice  
             donde esté el máximo. }  
}
```

¿Está bien usar este caso de test?

```
tests = test [  
  "MuchosMaximos" ~: (indiceMax [2,4,4,1]) ~= 1  
]
```

Funciones útiles para el TP

Motivación

```
problema indiceMax (s: seq<Z>) : Z {  
  requiere: { True }  
  asegura: { resultado es igual al índice del mayor elemento en  
             s. En caso de tener repetidos, devolver cualquier índice  
             donde esté el máximo.}  
}
```

¿Está bien usar este caso de test?

```
tests = test [  
  "MuchosMaximos" ~: (indiceMax [2,4,4,1]) ~= 1  
]
```

¿Es 2 una solución correcta para el problema? ¿Pasaría el caso de test?

Funciones útiles para el TP

Motivación

```
problema indiceMax (s: seq<Z>) : Z {  
  requiere: { True }  
  asegura: { resultado es igual al índice del mayor elemento en  
             s. En caso de tener repetidos, devolver cualquier índice  
             donde esté el máximo. }  
}
```

¿Está bien usar este caso de test?

```
tests = test [  
  "MuchosMaximos" ~: (indiceMax [2,4,4,1]) ~= 1  
]
```

¿Es 2 una solución correcta para el problema? ¿Pasaría el caso de test?

Tenemos que decir cuáles son todas las posibles soluciones válidas!
Para esto podemos usar directamente *expectAny*.

Funciones útiles para el TP

expectAny: podemos defirla como:

```
problema expectAny (actual:  $a$ , expected:  $seq\langle a \rangle$ ) : Test {  
  requiere: {  $True$  }  
  asegura: {  $res$  es un Test Verdadero si y sólo si  $actual$   
             pertenece a la lista  $expected$  }  
}
```

Donde Test Verdadero significa que el test pasa satisfactoriamente;
caso contrario, el Test falla.

```
tests = test [  
  "MuchosMaximos" ~: expectAny (indiceMax [2,4,4,1]) [1,2]  
]
```

Esta y otras funciones ya están definidas en el Template del TP y
pueden usarlas directamente.

Funciones útiles para el TP

Si queremos ver que dos listas de \mathbb{R} sean iguales, ¿Podemos usar la función *sonIguales()*?

Funciones útiles para el TP

Si queremos ver que dos listas de \mathbb{R} sean iguales, ¿Podemos usar la función *sonIguales()*?

Rta: No! Los números decimales en general no pueden representarse en binario de forma exacta. Además, dependiendo de la arquitectura, la precisión de decimales puede variar. Entonces vamos a decir que son iguales si “son suficientemente iguales”.

Funciones útiles para el TP

expectlistProximity

problema expectlistProximity (actual: $seq\langle\mathbb{R}\rangle$, expected: $seq\langle\mathbb{R}\rangle$) : Test {
 requiere: { *True* }
 asegura: { *res* es un Test Verdadero si y sólo si
 $|actual| = |expected|$ y para todo $i \in \mathbb{Z}$ tal que
 $0 \leq i < |actual|$, $|actual[i] - expected[i]| <$
 $margenFloat()$ }
}

Donde $margenFloat() = 0,00001$

Funciones útiles para el TP

expectlistProximity

problema expectlistProximity (actual: $seq\langle\mathbb{R}\rangle$, expected: $seq\langle\mathbb{R}\rangle$) : Test {
 requiere: { *True* }
 asegura: { *res* es un Test Verdadero si y sólo si
 $|actual| = |expected|$ y para todo $i \in \mathbb{Z}$ tal que
 $0 \leq i < |actual|$, $|actual[i] - expected[i]| <$
 margenFloat() }

Donde *margenFloat()* = 0,00001

Ejemplo de uso:

```
tests = test [  
  "sonIguales" ~: expectlistProximity [0.000011, 3.333334]  
  [0.00001, 3.33333]  
]
```

Funciones útiles para el TP

expectAnyTuplaAprox

problema expectAnyTuplaAprox (actual: $(\text{Char} \times \mathbb{R})$, expected: $\text{seq}((\text{Char} \times \mathbb{R}))$) : Test {
 requiere: { *True* }
 asegura: { *res* es un Test Verdadero si y sólo si para algún
 $i \in \mathbb{Z}$ tal que
 $0 \leq i < |\text{expected}|$, $(\text{expected}[i])_0 = (\text{actual})_0$ y
 $|(\text{expected}[i])_1 - (\text{actual})_1| < \text{margenFloat}()$ }
}

Donde $\text{margenFloat}() = 0,00001$

Funciones útiles para el TP

expectAnyTuplaAprox

problema expectAnyTuplaAprox (actual: $(\text{Char} \times \mathbb{R})$, expected: $\text{seq}((\text{Char} \times \mathbb{R}))$) : Test {
 requiere: { *True* }
 asegura: { *res* es un Test Verdadero si y sólo si para algún $i \in \mathbb{Z}$ tal que
 $0 \leq i < |\text{expected}|$, $(\text{expected}[i])_0 = (\text{actual})_0$ y
 $|(\text{expected}[i])_1 - (\text{actual})_1| < \text{margenFloat}()$ }
}

Donde $\text{margenFloat}() = 0,00001$

Ejemplo de uso

```
tests = test [  
  "esIguar" ~: expectAnyTuplaAprox ('c',0.000012)  
    [ ('c',0.00001) ],  
  "algunoEsIguar" ~: expectAnyTuplaAprox ('c',0.000012)  
    [ ('a',0.00001), ('c',0.00001) ]  
]
```

Funciones útiles para el TP

expectPermutacion (similar a soniguales)

```
problema expectPermutacion (actual:  $seq\langle a \rangle$ , expected:  $seq\langle a \rangle$ )
: Test {
  requiere: { True }
  asegura: { res es un Test Verdadero si y sólo si para todo
             elemento e de tipo a,  $\#Apariciones(actual, e) =$ 
              $\#Apariciones(expected, e)$  }
}
```

Funciones útiles para el TP

expectPermutacion (similar a sonIguales)

```
problema expectPermutacion (actual: seq<a>, expected: seq<a>)  
: Test {  
  requiere: { True }  
  asegura: { res es un Test Verdadero si y sólo si para todo  
             elemento  $e$  de tipo  $a$ ,  $\#Apariciones(actual, e) =$   
              $\#Apariciones(expected, e)$  }  
}
```

Ejemplo de uso

```
tests = test [  
  "esPermutacion" ~: expectPermutacion [1,2,5,4,4]  
    [1,2,4,4,5]  
]
```

Funciones útiles para el TP

Resumen

- ▶ `expectAny`
- ▶ `expectlistProximity`
- ▶ `expectAnyTuplaAprox`
- ▶ `expectPermutacion`