

Introducción a la programación

HUnit: framework de Testing Unitario para Haskell

Instalación

Requiere {Tener instalado ghc. Revisar el tutorial de instalación subido al campus}

¹Si ejecutan 'cabal -Version' y no les dice la versión, entonces no lo tienen instalado

Instalación

Requiere {Tener instalado ghc. Revisar el tutorial de instalación subido al campus}

Pasos (Windows)

- ▶ Abrir una consola de comandos (presionando la tecla “Windows” + “R” y luego escribiendo “cmd” en el cuadro de diálogo que aparece).
- ▶ Ejecutar
 - ▶ choco install cabal (sólo si no tienen instalado cabal¹)
 - ▶ cabal update
 - ▶ cabal install --lib HUnit

¹Si ejecutan 'cabal -Version' y no les dice la versión, entonces no lo tienen instalado

Instalación

Requiere {Tener instalado ghc. Revisar el tutorial de instalación subido al campus}

Pasos (Linux)

Instalación

Requiere {Tener instalado ghc. Revisar el tutorial de instalación subido al campus}

Pasos (Linux)

- ▶ Abrir una terminal (Ctrl + Alt + T)
- ▶ Ejecutar
 - ▶ sudo apt install cabal-install
 - ▶ cabal update
 - ▶ cabal install --lib HUnit (En caso de que falle la instalación probar sin el --lib)

Chequear instalación

Para validar que tenemos instalado correctamente HUnit, podemos ejecutar en una terminal lo siguiente:

Chequear instalación

Para validar que tenemos instalado correctamente HUnit, podemos ejecutar en una terminal lo siguiente:

- ▶ `ghci`
- ▶ `import Test.HUnit`

Si no mostró un mensaje de error, entonces la instalación fue exitosa!

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.
- ▶ 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.

```
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!