Testeo Unitario en Haskell con HUnit

Pablo López

October, 2021

¿Qué es HUnit?

`HUnit` es una biblioteca para test unitarios de la familia `XUnit`.

Para poder utilizar HUnit es necesario importar la biblioteca:

```
import Test.HUnit
```

Aserto `assertEquals`

Para especificar un caso de prueba que compruebe la igualdad entre la salida esperada y la obtenida (`assertEquals`) se utiliza el operador `~=?`, que tiene tipo:

```
(~=?) :: (Show a, Eq a) => a -> a -> Test
```

donde el primer argumento es el valor esperado y el segundo el obtenido:

```
expected value ~=? actual value
```

Ejemplo simple de test HUnit

Para la función:

```
suma :: Num a => [a] -> a
suma [] = 0
suma (x:xs) = x +  suma xs
```

podemos especificar el test:

```
testSimple :: Test
testSimple = 55 ~=? suma [1..10]
```

Ejecución de tests HUnit

Hay que tener en cuenta que el operador `~=?` *construye* un test (tipo `Test`), pero **no lo ejecuta** (como ocurre en `JUnit` con `@Test`).

La ejecución se hace a través de la función `runTestTT`:

```
> runTestTT testSimple
Cases: 1  Tried: 1  Errors: 0  Failures: 0
Counts {cases = 1, tried = 1, errors = 0, failures = 0}
```

Aserto `assertEquals` con mensaje

Es conveniente asociar a cada aserto un mensaje (`String`) que ayude a identificar la causa del error. Para ello utilizamos el operador:

```
(~:) :: Testable t => String -> t -> Test
```

Para el ejemplo anterior:

```
testSimpleConMensaje :: Test
testSimpleConMensaje =
    "suma [1..10] /=  55" ~: 55 ~=? suma [1..10]
```

Aserto `assertTrue`

También es posible comprobar que una expresión booleana es `True` mediante el operador:

```
(~?) :: AssertionPredicable t => t -> String -> Test
```

Por ejemplo:

```
testBooleano :: Test
testBooleano = 1 <= 2 ~? "1 debe ser menor o igual que 2"
```

Combinación de tests: listas de tests

Los tests construidos con los operadores `~=?` y `~?` se pueden combinar de varias formas.

Una de las más simples es construir una lista de tests y aplicarle el constructor `TestList`:

```
TestList :: [Test] -> Test
```

Por ejemplo:

```
listaTests :: Test
listaTests =
  TestList [ "suma [1..10] /= 55" ~: 55 ~=? suma [1..10],
             1 <= 2 ~? "1 debe ser menor o igual que 2"
           ]
```

Otras combinaciones de tests

Otra forma más potente de combinar tests es mediante la función `test`:

```
test :: Testable t => t -> Test
```

Por ejemplo:

```
agrupaTests :: Test
agrupaTests =
   test [ "suma [1..10] /= 55" ~: 55 ~=? suma [1..10],
          1 <= 2 ~? "1 debe ser menor o igual que 2"
        ]
```

Ejecución de tests

Los tests se ejecutan a través de la función `runTestTT`:

```
runTestTT :: Test -> IO Counts
```

que ejecuta un test (que puede ser un test combinado) y devuelve un informe con el resultado de la ejecución.

Por defecto, el informe se imprime en pantalla (TT viene de Tele Type).

El informe indica:

- cuántos tests hay en el test (*cases*)
- cuántos se han ejecutado (*tried*)
- cuántos han acabado con fallo (*failures*)
- cuántos no han podido ejecutarse por error en tiempo de ejecución (*errors*).

Ejecución de test combinado

```
runTestTT $
    test [ "suma [1..10] /= 55" ~: 55 ~=? suma [1..10],
           1 <= 2 ~? "1 debe ser menor o igual que 2" ]
```

produce el informe:

```
Cases: 2  Tried: 2  Errors: 0  Failures: 0
Counts {cases = 2, tried = 2, errors = 0, failures = 0}
```