

# Property-based Testing con PropEr

Miguel Emilio Ruiz Nieto

29 de noviembre de 2021

# Contenidos

- 1 Motivación
- 2 Definiciones
- 3 Erlang
- 4 PropEr
- 5 Un caso real
- 6 Bibliografía

Añadir conclusiones

```
-module(sort_lib).  
  
-export([sort/1]).  
% Implementation of quicksort algorithm  
-spec sort(list(integer())) -> list(integer()).  
sort([]) -> [];  
sort([P|Xs]) ->  
    sort([X || X <- Xs, X < P]) ++  
        [P] ++ sort([X || X <- Xs, P < X]).
```

# Motivación. Test unitarios

```
-module(sort_lib_eunit).  
  
-include_lib("eunit/include/eunit.hrl").  
  
sort_test_() ->  
    [test_zero(), test_two(), test_four()].  
test_zero() ->  
    ?_assertEqual([], sort_lib:sort([])).  
test_two() ->  
    [?_assertEqual([17,42],  
                    sort_lib:sort([X,Y]))  
      || {X,Y} <- [{17,42}, {42,17}]  
    ].  
test_four() ->  
    ?_assertEqual([1,2,3,4],  
                    sort_lib:sort([3,1,4,2])).
```

## Preguntas

- ¿Son buenos estos tests?
- ¿Harían falta más?
- En caso de que sí, ¿cuántos más?

Las metodologías de testing tradicionales son útiles ya que:

- Obliga a los desarrolladores a escribir casos de prueba del software desarrollado
- Para cada input se debe generar un cierto output con el fin de comprobar el correcto funcionamiento del sistema

Pero tienen sus inconvenientes:

- Consumen tiempo (€€€)
- No se garantiza que la batería de tests cubra todos los casos

La alternativa

**Property-based testing**



# Property-based Testing. Definición

- Es una técnica para hacer pruebas sobre las propiedades de nuestro sistema
- Los tests no son sobre casos de uso, sino sobre el comportamiento del propio sistema
- Muy común en lenguajes de programación funcional (i.e Quickcheck)

PBT tiene dos fases 1) generar datos de entrada de manera aleatoria 2) hacer verificaciones de las funciones aplicadas a esos datos

- Son reglas generales que describen el comportamiento de una función o un programa
- Han de ser aplicables a cualquier tipo de entrada y salida del propio programa bajo sus propias condiciones
- La salida debe verificar ciertas características deseadas

Revisar si existe ?EXISTS

Al principio puede resultar no tan trivial como los tests unitarios ya que:

- El desarrollador ha de tener una visión más nítida de los casos de uso y del comportamiento del sistema

No obstante:

- Asegura encontrar un mayor número de “casos esquina” y bugs dentro del código

Añadir en conclusiones: Merece la pena usar PBT porque te genera casos de test muy 'rápido'

Añadir en conclusiones: Hay casos en los que puede merecer la pena más hacer test unitarios

Añadir en conclusiones: La realidad es que no se usa porque está muy centrado en lenguajes funcionales

- Lenguaje de programación desarrollado en Ericsson
- Orientado a sistemas distribuidos:
  - Modelo de actores
  - Paso de mensajes
  - Tolerancia a fallos
  - Alta disponibilidad
  - Filosofía “Let it crash”

```
-module(sucessions).  
-export([fib/1]).
```

```
fib(0) -> 0;  
fib(1) -> 1;  
fib(N) -> fib(N-1) + fib(N-2).
```

```
1> [First | TheRest] = [1,2,3,4,5].  
2> First.  
1  
3> TheRest.  
[2,3,4,5]
```

# Sintaxis. Tuplas

```
4> X = 10, Y = 4.
```

```
4
```

```
5> Point = {X,Y}.
```

```
{10,4}
```

```
6> PreciseTemperature = {celsius, 23.213, 45}.
```

# Sintaxis. Funciones de Orden Superior

```
7> Add_3 = fun(X) -> X + 3 end.  
#Fun<erl_eval.7.126501267>  
8> lists:map(Add_3, [1,2,3]).  
[4,5,6]
```

Poner un ejemplo de pattern matching



- Herramienta para realizar property-based testing en Erlang
- Inspirada en Quickcheck
- Completamente integrada con los tipos de Erlang

Nos centraremos en los siguientes aspectos:

- Generadores
- Estructura de las propiedades
- Propiedades sin estado
- Propiedades con estado
- Reducción de casos con ?SHRINK

- Funciones que *generan* entradas de una manera “aleatoria”
- Proporcionan datos en base al tipo del generador y a los filtros dados
- Pueden ser:
  - En base a los tipos de Erlang
  - Customizados por el desarrollador

# PropEr. Generadores. Ejemplos

Basados en los tipos de Erlang

| Generador               | Muestra                |
|-------------------------|------------------------|
| <code>integer()</code>  | 89234                  |
| <code>boolean()</code>  | true, false            |
| <code>list(Type)</code> | [true, true, false]    |
| <code>tuple()</code>    | {true, 13.321123, -67} |

Customizados por el desarrollador

```
?LET(InstanceOfType, TypeGenerator, Transform).
```

- **InstanceOfType** : La variable que contendrá los datos generados por el generador del segundo argumento
- **TypeGenerator** : La función generadora que produce los datos que se almacenan en el argumento anterior
- **Transform** : Expresión que transforma los datos de la propia función y los acumula en el argumento anterior

```
% Customized Generator
list_no_dupls(T) ->
    ?LET(L, list(T), remove_duplicates(L)).

% Helper
remove_duplicates([]) -> [];
remove_duplicates([A|T]) ->
    case lists:member(A, T) of
        true -> remove_duplicates(T);
        false -> [A|remove_duplicates(T)]
    end.
```

`?FORALL(InstanceOfType, TypeGenerator,  
PropertyExpression).`

- `InstanceOfType` : La variable que contendrá los datos generados por los generadores
- `TypeGenerator` : La función generadora que produce los datos que se almacenan en el argumento anterior
- `PropertyExpression` : Expresión booleana que especifica la propiedad que se desea verificar

# PropEr. Estructura de las propiedades. Ejemplos

```
-module(prop_sort_lib).  
  
-include_lib("proper/include/proper.hrl").  
  
% Propiedad con generador con tipo de Erlang  
prop_same_length() ->  
    ?FORALL(L, list(integer()),  
        length(L) == length(sort_lib:sort(L))).  
  
% Propiedad con generador customizado  
prop_same_length_no_dupls() ->  
    ?FORALL(L, list_no_dupls(integer()),  
        length(L) == length(sort_lib:sort(L))).
```



# PropEr. Propiedades sin estado

Qué ejemplo puedo poner aqui

# PropEr. Propiedades con estado

Qué ejemplo puedo poner aqui

Decir que hasta ahora hemos visto propiedades sin estado

Poner de ejemplo de propiedad con estado Persona puede conducir

- En algunos casos, puede haber ciertos tests que no cumplan las propiedades que hemos definido
- Por ello, es conveniente *reducir* el número de muestras que PropEr genera para pasar dicho test con ?SHRINK

# PropEr. Macro ?SHRINK. Ejemplo

*%% TODO cómo funciona shrinking internamente*

# Un caso real

Hablar sobre este "caso real" ??

- Baleen. Biblioteca de validadores escritos en Erlang
- El caso de Coowry

# Un caso real

```
-type validator(A, B) -> (fun(A) -> result(B) end).  
-type result(A) -> {ok, A} | {error, binary()}.
```

- Getting Started with Erlang  
[https://www.erlang.org/doc/getting\\_started/intro.html](https://www.erlang.org/doc/getting_started/intro.html)
- PropEr Guide <https://proper-testing.github.io>
- Property-Based Testing with PropEr, Erlang and Elixir  
<https://www.propertesting.com/>
- Baleen repo <https://github.com/coowry/baleen>