

STOCKHOLM

HYBRID CONFERENCE

Improve your tests with Makina

Luis Eduardo Bueso de Barrio

May 20 | 2022

The problem



files	blank	comment	code	files	blank	comment	
4	760	383	4513	18	500	408	1

Introduction to PBT



Writing unit-tests is hard and time-consuming:

```
reverse([]) == []
reverse([1]) == [1]
reverse([1 ,2]) == [2 ,1]
...
```

Property-Based Testing (PBT) philosophy: Don't write tests, generate them.

A test execution in PBT consists of:

- 1. Data generation.
- 2. Property checking.
- 3. An shrinking strategy (if the property doesn't hold).

A property:

```
forall list <- list() do
   list == reverse(reverse(list))
end</pre>
```

In each test:

list() generates a random list:

```
[8 ,10 ,6] ...
```

2. Checks the property:

```
[8, 10, 6] == reverse(reverse([8, 10, 6]))
```

3. If the property doesn't hold returns a counter-example.

Testing stateful programs



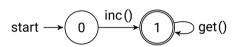
Imagine a simple counter:

Command	Returns		
start/1	:ok :error		
stop/0	:ok :error		
inc/0	:ok		
get/0	integer()		

Unit test:

```
:ok = start(0)
:ok = inc()
1 = get()
:ok = stop()
```

This test can be represented:



To successfully test this program we need to:

- Generate sequences of commands.
- An internal state to track the changes in the program.
- A way to interact with the program under test.

PBT of stateful programs



Basic property of stateful programs:

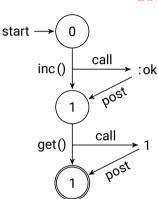
```
forall cmds <- commands(Counter) do
  :ok == run_commands(Counter, cmds)
end</pre>
```

where:

commands/1 generates sequences commands:

```
[start(0), inc(), get(), stop()] ...
```

run_commands executes the generated sequence.



PBT state machines



Very slow adoption.

Introduced by Erlang QuickCheck.

Adopted by other PBT libraries such as Proper.

These libraries provide a DSL to model the system.

Proven effectiveness.

Problems:

Whv?

- Hard to write.
- · Cryptic errors.
- Usually buggy.
- Code reuse is very hard.
- Hard to maintain.

Makina



Makina is a DSL to write PBT state machines.

- Fully compatible with Erlang QuickCheck and PropEr.
- Improved usability.
- Better error messages.
- Improved error detection in models.

A Makina model contains:

- state
- command
- invariants

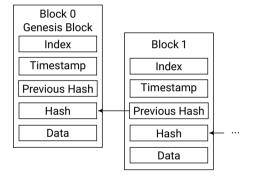
Provides code reuse mechanisms:

- imports
- extends
- composition

Ethereum Blockchain



A blockchain is a distributed ledger that enables peer-to-peer transactions.



Ethereum is one of the largest blockchains.

Introduced smart-contracts.

etherex a library to interact with Ethereum using Elixir.

The properties to test:

- 1. Mining blocks.
- Account access.
- 3. Transfers between accounts.

Mining blocks



The API:

Command	Returns			
mine/0	:ok			
block_number/0	{:ok,	integer()}		

- 1. state?
- 2. invariants?
- 3. commands?

```
defmodule Blocks do
 use Makina, implemented_by: Etherex
  state height: 0
  invariants non_neg_height: height > 0
  command block_number() do
    post {:ok, height} == result
  end
 command mine() do
   call Etherex.Time.mine()
    next height: height + 1
 end
end
```

Consulting callback docs

```
iex> h Blocks.Command.Mine.post

# def post(state, arguments, result)
```

```
@spec post(dynamic_state(), arguments(), result()) :: boolean()
```

This callback contains a predicate that should be true after the execution of mine

```
## Available variables
```

State

- state contains the complete dynamic state of the model.
- height attribute defined in the state declaration.

Arguments

- arguments contains all the generated arguments of the command.

Result

- result variable that contains the result of the command execution.



Adding type information

```
8 9 9 £
8 £ 9 M
EUROPE
```

```
defmodule Blocks do
      use Makina. implemented by: Etherex
      state height: 0 :: integer()
6
      invariants non_neg_height: height >= 0
      command block_number() :: {:ok, integer()} do
        post {:ok, height} == result
10
      end
11
12
      command mine() :: :ok do
13
        call Etherex.Time.mine()
14
        next height: height + 1
15
      end
16
    end
    $ mix gradient
    The function call Etherex.block_number() on line 8 is expected
    to have type result() but it has type {:ok, quantity()} | {:error, error()}
```

Adding documentation

```
600 E EPM EUROPE
```

```
defmodule Blocks do
  use Makina. implemented by: Etherex
 @moduledoc """
  Specifies the mining facilities of the blockchain.
  state height: Etherex.block_number!() :: non_neg_integer()
  invariants non_genesis_block: height >= 0
  command block_number() :: {:ok, non_neg_integer()} do
    @moduledoc "Retrieves the block number from the blockchain."
    post {:ok, height} == result
  end
  command mine() :: :ok do
    @moduledoc "Mines a new block."
    call Etherex. Time.mine()
    next height: height + 1
  end
end
```

Consulting module docs

iex> h Blocks

Blocks

Contains a Makina model called Blocks.

Specifies the mining facilities of the blockchain.

Commands

- mine stored at Blocks.Command.Mine
- block_number stored at Blocks.Command.BlockNumber

Detailed information about each command can be accessed inside the interpreter:

iex> h Blocks.Command.NAME

State attributes

- height

. . .

```
$ mix test
Starting Quviq QuickCheck version 1.45.1
Failed! After 1 tests.
    Blocks.block_number/0,
    Blocks.mine/0.
Postcondition crashed:
** (Makina.Error) invariant "non_neg_height" check failed
Shrinking x.(1 \text{ times})
    Blocks.block_number/0
Last state: %{height: 0}
Finished in 0.1 seconds (0.00s async, 0.1s sync)
1 properties, 1 failure
```

Fixing the model



```
The error message:

Postcondition crashed:
** invariant "non_neg_height" check failed

Shrinking x.(1 times)
[
Blocks.block_number/0
]

Last state: %{height: 0}

What happened?
```

Invariant doesn't hold even on the initial state!

```
defmodule Blocks do
  use Makina, implemented_by: Etherex
  state height: 0
  invariants non neg height: height >= 0
  command block_number() do
    post {:ok, height} == result
  end
  command mine() do
    call Etherex.Time.mine()
    next height: height + 1
 end
end
```



\$ mix test
Starting Quviq QuickCheck version 1.45.1

.....

OK, passed 100 tests

51.5 Blocks.mine/0
48.5 Blocks.block_number/0

Finished in 8.6 seconds (0.00s async, 8.6s sync)

Account access



```
The API:
```

Command Returns
get_balance/1 :ok

- 1. state?
- 2. invariants?
- 3. commands?

defmodule Accounts do use Makina, implemented_by: Etherex alias Etherex. Type state accounts: Etherex.accounts!() :: [Type.address()], balances: Etherex.balances!() :: %{Type.address() => integer()} command get_balance(account :: Type.address()) :: {:ok, Type.quantity()} | {:error, Type.error()} do pre accounts != [] post {:ok, balances[account]} == result end end



Starting Quviq QuickCheck version 1.45.1

```
1) property Accounts (ExamplesTest)
  ** (Makina.Error) argument 'account' missing in command get_balance
  stacktrace:
  (makina 0.1.0) lib/makina/error.ex:9: Makina.Error.throw_error/1
  (examples 0.1.0) lib/accounts.ex:13: Accounts.Command.GetBalance.check_args/1
  (examples 0.1.0) lib/accounts.ex:1: Accounts.Behaviour.next_state/3
  Finished in 0.1 seconds (0.00s async, 0.1s sync)
```

1 properties, 1 failure

Randomized with seed 763550

Fixing the accounts model



```
defmodule Accounts do
  use Makina, implemented_by: Etherex
  alias Etherex. Type
  state accounts: Etherex.accounts!() :: [Type.address()],
    balances: Etherex.balances!() :: %{Type.address() => integer()}
  command get_balance(account :: Type.address()) ::
  {:ok, Type.quantity()} | {:error, Type.error()} do
    pre accounts != []
    args account: oneof(accounts)
    valid_args account in accounts
    post {:ok. balances[account]} == result
  end
end
```



```
$ mix test
Starting Quviq QuickCheck version 1.45.1
```

OK, passed 100 tests

'100.0 Accounts.get_balance/1

Finished in 4.6 seconds (0.00s async, 4.6s sync) 1 properties, 0 failures

Generating transactions



The API to generate and check transactions:

Command	Returns		
mine/0	:ok		
block_number/0	{:ok,	integer()}	
get_balance/1	{:ok,	integer()}	
transfer/3	{:ok,	hash()}	

We can compose Blocks and Accounts!

```
defmodule Transactions do
   use Makina,
     extends: [Blocks, Accounts],
   implemented_by: Etherex
end
```

Generates a model Transactions.Composed.

State is the union:

- :heiaht
- :accounts
- :balances

Invariants are the union:

:non_genesis_block

Commands are the union:

- mine/0
- block_number/0
- get_balance/1

Generating transactions

```
defmodule Transactions do
 use Makina, implemented_by: Etherex, extends: [Accounts, Blocks]
 alias Etherex.Type
 command transfer(
            from :: Type.address().
            to :: Type.address().
            value :: Type.guantity()
          ) :: {:ok, Type.hash()} do
    pre accounts != []
    args from: oneof(accounts), to: oneof(accounts), value: pos_integer()
    valid_args from in accounts and to in accounts
   next balances: update_balances(balances, from, to, value)
 end
 def update_balances(balances, from, to, value) do
   balances |> Map.update!(from. &(&1 - value)) |> Map.update!(to. &(&1 + value))
 end
end
```



```
$ mix test
Starting Quviq QuickCheck version 1.45.1
Failed! After 1 tests
    Transactions.transfer("0xffcf8fdee72ac11b5c542428b35eef5769c409f0".
                           "0x90f8bf6a479f320ead074411a4b0e7944ea8c9c1".
                          423319221061516289).
    Transactions.get_balance("0x90f8bf6a479f320ead074411a4b0e7944ea8c9c1"),
    Transactions.mine().
    Transactions.mine(),
    Transactions.block_number()
Postcondition failed.
. . .
Shrinking xxx.xx.x.x.x(4 times)
```

```
600 K 8 E A M EUROPE
```

```
Transaction.transfer("0xffcf8fdee72ac11b5c542428b35eef5769c409f0",
                         "0x90f8bf6a479f320ead074411a4b0e7944ea8c9c1".
    Transactions.block_number()
Postcondition failed.
. . .
Transactions.block_number() -> {:ok. 1}
Last state: %{height: 0. ...}
Finished in 0.8 seconds (0.00s async, 0.8s sync)
1 properties, 1 failure
```

Fixing the transactions model

```
8 9 D E B E P M EUROP E
```

```
defmodule Transactions do
 use Makina, implemented_by: Etherex, extends: [Accounts, Blocks]
 alias Etherex.Type
 command transfer(
   from :: Type.address(),
   to :: Type.address().
   value :: Type.quantity()
  ) :: {:ok, Type.hash()} do
   pre accounts != []
    args from: oneof(accounts), to: oneof(accounts), value: pos_integer()
   valid_args from in accounts and to in accounts
   next balances: update balances(balances, from, to, value).
         height: height + 1
 end
 def update_balances(balances, from, to, value) do
   balances |> Map.update!(from, &(&1 - value)) |> Map.update!(to, &(&1 + value))
 end
end
```

```
SODE
BEAM
EUROPE
```

```
$ mix test --only transactions
Starting Quviq QuickCheck version 1.45.1
..Failed! After 2 tests.
    . . .
Postcondition failed.
. . .
Shrinking xxxx.x.x.x.x.x(6 times)
    Transactions.transfer("0x90f8bf6a479f320ead074411a4b0e7944ea8c9c1".
                          "0x90f8bf6a479f320ead074411a4b0e7944ea8c9c1".
    Transactions.get_balance("0x90f8bf6a479f320ead074411a4b0e7944ea8c9c1")
Postcondition failed
```

#CodeBEAM



```
Commands:
. . .
Transactions.get_balance("0x90f8bf6a479f320ead074411a4b0e7944ea8c9c1")
 Last state:
%{
 balances: %{
  . . .
 }.
 . . .
Finished in 1.1 seconds (0.00s async, 1.1s sync)
3 properties, 1 failure
```

Fixing the transactions model

Our model does not consider gas consumption.

command gas_cost(hash :: Type.hash()) :: {:ok, Type.hash()}

pre transactions != []

args hash: oneof(transactions)

valid_args hash in transactions

We need to change:

- 1. A symbolic attribute to store transactions.
- 2. balances needs to be symbolic.
- 3. get_balance/0 precondition.
- a new command gas_cost/1.

```
next do
defmodule Transactions do
                                                     from = symbolic(Kernel.elem(result, 0))
                                                     gas = symbolic(Kernel.elem(result, 1))
  state transactions: [] :: [symbolic(Type.hash())]
                                                       transactions: List.delete(transactions, has
  command get_balance() do
                                                       balances: discount_gas(balances, from, gas)
    pre IO.inspect(transactions) == []
                                                                  l> symbolic()
  end
                                                    end
  command transfer(...) do
                                                 end
    next transactions: [symbolic(elem(result, 1)) | transactions],
  end
```

```
#CodeBEAM
```



```
$ mix test
Starting Quviq QuickCheck version 1.45.1
```

OK, passed 100 tests

```
'25.5 Transactions.mine/0
```

- '24.9 Transactions.block_number/0
- '23.6 Transactions.transfer/3
- '14.3 Transactions.gas_cost/1
- '11.8 Transactions.get_balance/1

Finished in 15.8 seconds (0.00s async, 15.8s sync) 1 properties, 0 failures