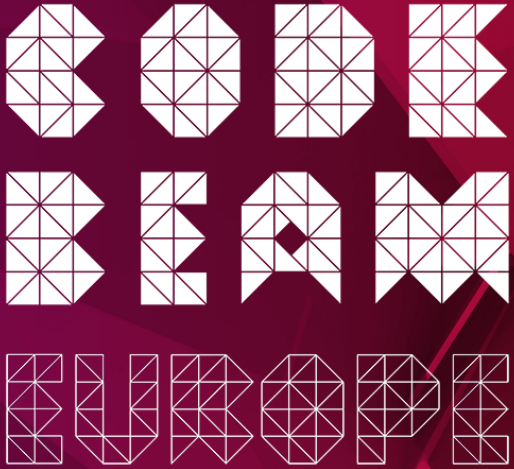


STOCKHOLM

HYBRID CONFERENCE

Improve your tests with Makina

Luis Eduardo Bueso de Barrio

May 20 | 2022

#CodeBEAM

Before:

files	blank	comment	code
4	760	383	4513

After:

files	blank	comment	code
18	500	408	1692

PBT models

Proprety-Based Testing (PBT) is a great testing methodology.

Successful tools widely used:

- Erlang QuickCheck (EQC)
- PropEr

These tools are great for testing pure functions.

They have mechanisms to test stateful programs.

PBT state-machines or models.

A PBT model is like an oracle.

Problems with PBT models

Despite their proven effectiveness:

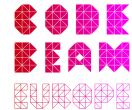
- Very slow adoption

Why?

1. Models are hard to reuse.
2. Bugs in models are hard to detect.
3. Errors are hard to understand.

All these problems made models hard to write and maintain.

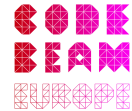
Our solution: Makina



Makina is a DSL for writing PBT models.

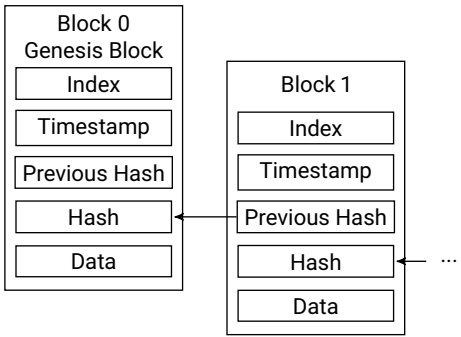
1. Models are hard to reuse.
   - Model refinement and composition.
2. Bugs in models are hard to detect.
   - Automatic type and specs generation.
3. Errors are hard to understand.
   - Automatic generation of runtime-checks.

Introduction to Makina



Ethereum Blockchain

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



Why Ethereum?

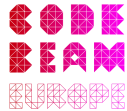
- Big community.
- Multiple implementations.

To interact we will use `etherex`:
`https://gitlab.com/babel-upm/blockchain/etherex`

The properties to test:

1. Mining blocks.
2. Account access.
3. Transactions between accounts.

Mining blocks



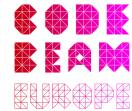
The API:

Command	Returns
`mine/0`	`:ok`
`block_number/0`	`integer()`

1. create module.
2. import `Makina`.
3. define state.
4. define invariants.
5. define commands.

```
defmodule Blocks do
  use Makina

  state height: 0

  invariants non_neg_height: height > 0

  command block_number() do
    pre true
    args []
    call Etherex.block_number
    next []
    post height == result
  end
```

Mining blocks



The API:

Command	Returns
`mine/0`	`:ok`
`block_number/0`	`integer()`

1. create module.
2. import `Makina`.
3. define state.
4. define invariants.
5. define commands.

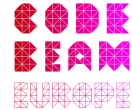
```elixir
defmodule Blocks do
  use Makina, implemented_by: Etherex

  state height: 0

  invariants non_neg_height: height > 0

  command block_number() do
    post height == result
  end

  command mine() do
    next height: height + 1
  end
end
```

Running the test

```
$ mix test

Failed! After 1 tests.

Postcondition crashed:
** invariant "non_neg_height" check failed

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

Last state: %{height: 0}
```

```elixir
defmodule Blocks do
  use Makina, implemented_by: Etherex

  state height: 0

  invariants non_neg_height: height > 0

  command block_number() do
    post height == result
  end

  command mine() do
    next height: height + 1
  end
end
```

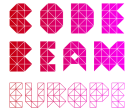
Fixing the model



```
$ mix test

Failed! After 1 tests.

Postcondition crashed:
** invariant "non_neg_height" check failed

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

Last state: %{height: 0}
```

```elixir
defmodule Blocks do
  use Makina, implemented_by: Etherex

  state height: 0

  invariants non_neg_height: height >= 0

  command block_number() do
    post height == result
  end

  command mine() do
    next height: height + 1
  end
end
```

Running the test



```
$ mix test

..................................................
..................................................

OK, passed 100 tests

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

```elixir
defmodule Blocks do
  use Makina, implemented_by: Etherex

  state height: 0

  invariants non_neg_height: height >= 0

  command block_number() do
    post height == result
  end

  command mine() do
    next height: height + 1
  end
end
```

Adding type information

```
$ mix gradient

$
```

Something changes in `Etherex`...

```
$ mix gradient

The function call Etherex.block_number()
on line 8 is expected to have type integer()
but it has type
{:ok, quantity()} | {:error, error()}

$
```

```elixir
defmodule Blocks do
  use Makina, implemented_by: Etherex

  state height: 0 :: integer()

  invariants non_neg_height: height >= 0

  command block_number() :: integer() do
    post height == result
  end

  command mine() :: :ok do
    next height: height + 1
  end
end
```

Adding documentation

```
iex> h Blocks

Contains a Makina model called Blocks.

Checks blocks are mined correctly.

## Commands

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

## State attributes

- height

## Invariants

- non_neg_height
```

```
defmodule Blocks do
  use Makina, implemented_by: Etherex

  @moduledoc """
  Checks blocks are mined correctly.
  """
  state height: 0 :: integer()

  invariants non_neg_height: height >= 0

  command block_number() :: integer() do
    @moduledoc "Gets the block number."
    post {:ok, height} == result
  end

  command mine() :: :ok do
    @moduledoc "Mines a new block."
    next height: height + 1
  end
end
```

Adding documentation

```
iex> h Blocks.Command.Mine

This module contains the functions necessary to
generate and execute the command mine.

Mines a new block.

## Definitions

- next
- call
- weight
- post
- args
- pre
```

```elixir
defmodule Blocks do
  use Makina, implemented_by: Etherex

  @moduledoc """
  Checks blocks are mined correctly.
  """
  state height: 0 :: integer()

  invariants non_neg_height: height >= 0

  command block_number() :: integer() do
    @moduledoc "Gets the block number."
    post {:ok, height} == result
  end

  command mine() :: :ok do
    @moduledoc "Mines a new block."
    next height: height + 1
  end
end
```

Adding documentation

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

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

## Available variables

### State

  - state
  - height

### Arguments

  - arguments

### Result

  - result
```

```elixir
defmodule Blocks do
  use Makina, implemented_by: Etherex

  @moduledoc """
  Checks blocks are mined correctly.
  """
  state height: 0 :: integer()

  invariants non_neg_height: height >= 0

  command block_number() :: integer() do
    @moduledoc "Gets the block number."
    post {:ok, height} == result
  end

  command mine() :: :ok do
    @moduledoc "Mines a new block."
    next height: height + 1
  end
end
```

Account access

The API:

Command	Returns
`balance/1`	`integer()`

1. create module.
2. import `Makina`.
3. define state.
4. define commands.

```
defmodule Accounts do
  use Makina, implemented_by: Etherex

  state accounts: Etherex.accounts() :: [address()],
        balances: Etherex.balances() :: %{address() => integer()}

  command balance(account :: address()) :: integer() do
    pre accounts != []
    post balances[account] == result
  end
end
```

Running the test

```
$ mix test

** (Makina.Error) argument
    ‘account‘ missing in command
    get_balance
```

```elixir
defmodule Accounts do
  use Makina, implemented_by: Etherex

  state accounts: Etherex.accounts() :: [address()],
        balances: Etherex.balances() :: %{address() => integer()}

  command balance(account :: address()) :: integer() do
    pre accounts != []
    post balances[account] == result
  end
end
```

Fixing the model

```
$ mix test

** (Makina.Error) argument
    ‘account‘ missing in command
    get_balance
```

```elixir
defmodule Accounts do
  use Makina, implemented_by: Etherex

  state accounts: Etherex.accounts() :: [address()],
        balances: Etherex.balances() :: %{address() => integer()}

  command balance(account :: address()) :: integer() do
    args account: oneof(accounts)
    pre accounts != []
    post balances[account] == result
  end
end
```

Running the test

```
$ mix test

.......................
.......................
.......................
.......................
OK, passed 100 tests

'100.0 Accounts.get_balance/1
```

```elixir
defmodule Accounts do
  use Makina, implemented_by: Etherex

  state accounts: Etherex.accounts() :: [address()],
        balances: Etherex.balances() :: %{address() => integer()}

  command balance(account :: address()) :: integer() do
    args account: oneof(accounts)
    pre accounts != []
    post balances[account] == result
  end
end
```

Generating transactions

The API to generate and check transactions:

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

We can compose `Blocks` and `Accounts`!

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

Generates a model `Transactions.Composed`.

```
iex(1)> h Transactions.Composed

# Transactions.Composed

## Commands

 - mine stored
 - get_balance
 - block_number

## State attributes

 - height
 - balances
 - accounts

## Invariants

 - non_neg_height
```

Generating transactions

Command	Returns
`transfer/3`	`hash()`

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

  command transfer(from, to, value) :: hash() do
    pre accounts != []
    args from: oneof(accounts),
         to: oneof(accounts),
         value: pos_integer()
    next balances: update(balances, from, to, value)
  end
end
```

Running the test

```
$ mix test

transfer("0xffcf8fdee72ac11",
         "0x90f8bf6a479f320",
         1)
block_number()

Postcondition failed.

block_number() -> {:ok, 1}

Last state: %{height: 0, ...}
```

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

  command transfer(from, to, value) :: hash() do
    pre accounts != []
    args from: oneof(accounts),
         to: oneof(accounts),
         value: pos_integer()
    next balances: update(balances, from, to, value)
  end
end
```

Fixing the model

```
$ mix test

Postcondition failed.

transfer("0xffcf8fdee72ac11",
         "0x90f8bf6a479f320",
         1)
block_number() -> {:ok, 1}

Last state: %{height: 0, ...}
```

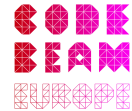
```elixir
defmodule Transactions do
  use Makina,
    implemented_by: Etherex,
    extends: [Accounts, Blocks]

  command transfer(from, to, value) :: hash() do
    pre accounts != []
    args from: oneof(accounts),
         to: oneof(accounts),
         value: pos_integer()
    next height: height + 1,
         balances: update(balances, from, to, value)
  end
end
```

Running the test

```
$ mix test

transfer("0x90f8bf6a479f320",
         "0x90f8bf6a479f320",
         1),
get_balance("0x90f8bf6a479f320")

Postcondition failed.

get_balance("0x90f8bf6a479f320")
-> {:ok, 979000}

Last state: %{
    balances: %{
        "0x90f8bf6a479f320" => 1000000
        .. }
    .. }
```

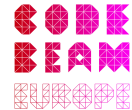
```
defmodule Transactions do
  use Makina,
    implemented_by: Etherex,
    extends: [Accounts, Blocks]

  command transfer(from, to, value) :: hash() do
    pre accounts != []
    args from: oneof(accounts),
         to: oneof(accounts),
         value: pos_integer()
    next height: height + 1,
         balances: update(balances, from, to, value)
  end
end
```

Fixing the model



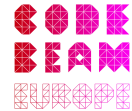
```
defmodule Transactions do
  use Makina,
    implemented_by: Etherex,
    extends: [Accounts, Blocks]

  command transfer(from, to, value) :: hash() do
    pre accounts != []
    args from: oneof(accounts),
         to: oneof(accounts),
         value: pos_integer()
    next height: height + 1,
         balances: update(balances, from, to, value)
  end
end
```

Fixing the model



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

  state transactions: [] :: [symbolic(hash())]

  command transfer(from, to, value) :: hash() do
    pre accounts != []
    args from: oneof(accounts),
         to: oneof(accounts),
         value: pos_integer()
    next height: height + 1,
         transactions: [result | transactions],
         balances: update(balances, from, to, value)
                   |> symbolic(),
  end

  command get_balance() do
    pre transactions == []
  end
end
```

Fixing the model



```
defmodule Transactions.GasCost do
  use Makina, extends: Transactions

  command gas_cost(hash :: hash())
      :: {address(), quantity()} do
    pre transactions != []
    args hash: oneof(transactions)
    next do
      from = symbolic(elem(result, 0))
      gas = symbolic(elem(result, 1))
      [transactions: List.delete(transactions, hash),
       balances: update(balances, from, gas)
                 |> symbolic()
      ]
    end
  end
end
```

Running the test

```
$ mix test

.................................................
.................................................
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
```

```
defmodule Transactions.GasCost do
  use Makina, extends: Transactions

  command gas_cost(hash :: hash())
      :: {address(), quantity()} do
    pre transactions != []
    args hash: oneof(transactions)
    next do
      from = symbolic(elem(result, 0))
      gas = symbolic(elem(result, 1))
      [transactions: List.delete(transactions, hash),
       balances: update(balances, from, gas)
                 |> symbolic()
      ]
    end
  end
end
```

Results and conclussions

Before Makina:

files	blank	comment	code
4	760	383	4513

After Makina:

files	blank	comment	code
18	500	408	1692

Libraries:

- `https://gitlab.com/babel-upm/makina/makina`
- `https://gitlab.com/babel-upm/blockchain/etherex`

Slides and code:

- `https://gitlab.com/babel-upm/makina/code_beam_2022`