Less arbitrary waiting time LambdaDays2022

Michał J. Gajda https://www.migamake.com

2022-07-28

# Plan

- Property testing
- Agile
- Problems with generators
- Generic solution

#### Property testing

```
Tests on sets not values
```

```
prop_showRead :: MyType -> Bool
prop_showRead x = read (show x) == x
```

main = quickCheck prop\_showRead

> +++ OK, passed 100 tests.

#### Property testing

```
Tests on sets not valuesLess work to make exhaustive tests
```

```
prop_showRead :: MyType -> Bool
prop_showRead x = read (show x) == x
main = quickCheck prop_showRead
```

```
> +++ OK, passed 100 tests.
```

#### Property testing

```
> Tests on sets not values
> Less work to make exhaustive tests
> Problem with recursive data structures
prop_showRead :: MyType -> Bool
prop_showRead x = read (show x) == x
main = quickCheck prop_showRead
> +++ OK, passed 100 tests.
```

#### Problem with generators

```
data MyType =
   Add MyType MyType
   | Mul MyType MyType
   | Const Int
   deriving (Eq, Ord, Show,Read,Generic)
```

#### instance Arbitrary Expr where

```
-- default method
```

#### Problem with generators

```
data MyType =
   Add MyType MyType
   | Mul MyType MyType
   | Const Int
   deriving (Eq, Ord, Show,Read,Generic)
```

#### instance Arbitrary Expr where

```
-- default method
```

<<loop>>

#### Problem with generators

```
data MyType =
    Add MyType MyType
    | Mul MyType MyType
    | Const Int
    deriving (Eq, Ord, Show,Read,Generic)
```

#### instance Arbitrary Expr where

```
-- default method
```

<<loop>>

Branching factor 2x for 2 of 3 constructors

HSExpr has 30 constructors and crazy branching factor..

#### Automatic generator

# instance Arbitrary Expr where arbitrary = oneOf [ Add <\$> arbitrary <\*> arbitrar ,Mul <\$> arbitrary <\*> arbitrary

,Const <\$> arbitrary]

Automatic generator – analysis

```
instance Arbitrary Expr where
arbitrary =
    oneOf [-- Doubles:
        Add <$> arbitrary <*> arbitrary
        -- Doubles:
        ,Mul <$> arbitrary <*> arbitrary
        -- Terminates
        ,Const <$> arbitrary]
```

#### Manual generator 1

# instance Arbitrary Expr where arbitrary = frequency [(1, Add <\$> arbitrary <\*> arbitrary) ,(1, Mul <\$> arbitrary <\*> arbitrary) ,(3, Const <\$> arbitrary )]

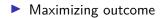
#### Manual generator 1

```
instance Arbitrary Expr where
arbitrary =
  frequency [(1, Add <$> arbitrary <*> arbitrary)
      ,(1, Mul <$> arbitrary <*> arbitrary)
      ,(3, Const <$> arbitrary )]
```

Termination probability is greater than branching factor.

#### Manual generator 2

Explicit termination count.



- Maximizing outcome
   Minimizing offent
- Minimizing effort

- Maximizing outcome
- Minimizing effort
- Choosing outcome

- Maximizing outcome
- Minimizing effort
- Choosing outcome
- ... finding manager who knows it

Easier to test on sets, but...

More time spent

Easier to test on sets, but...

- More time spent
- Effort in manual generators

Easier to test on sets, but...

- More time spent
- Effort in manual generators
- Looping forever is bad practice

Easier to test on sets, but...

- More time spent
- Effort in manual generators
- Looping forever is bad practice
- Async-based test runner will not even give error message

Maximize test coverage with property testing

#### Goal

Maximize test coverage with property testing

Minimum effort to write generators

#### Goal

- Maximize test coverage with property testing
- Minimum effort to write generators
- Always terminate

#### Goal

- Maximize test coverage with property testing
- Minimum effort to write generators
- Always terminate
- Work for mutually recursive data structures

#### instance LessArbitrary MyType where

instance \_ => Arbitrary MyType where
arbitrary = fasterArbitrary

State monad tracking cost of generated structure

State monad tracking cost of generated structureGeneric detects terminating constructors

- State monad tracking cost of generated structure
- Generic detects terminating constructors
- Bonus:

- State monad tracking cost of generated structure
- Generic detects terminating constructors
- Bonus:
  - expected size of structure

- State monad tracking cost of generated structure
- Generic detects terminating constructors
- Bonus:
  - expected size of structure
  - ignore branching factor

#### Solution: monad

```
newtype Cost = Cost Int
  deriving (Eq,Ord,Enum,Bounded,Num)
```

```
newtype CostGen s a =
    CostGen {
        runCostGen :: State.StateT (Cost, s) QC.Gen a }
    deriving (Functor, Applicative, Monad, State.MonadFix)
```

spend :: Cost -> CostGen ()
spend c = CostGen \$ State.modify (\(b, s) -> (b-c, s))

Solution: budget check operator

To make generation easier, we introduce budget check operator:

(\$\$\$?) :: CostGen a -> CostGen a -> CostGen a cheapVariants \$\$\$? costlyVariants = do budget <- CostGen State.get if | budget > (0 :: Cost) -> costlyVariants | budget > -10000 -> cheapVariants | otherwise -> error \$ "Recursive structure with no loop breaker."

The operator also reports non-terminating example generation.

# Solution with class

class LessArbitrary			s		a where	
lessArbitrary	::	CostGen	s		a	
default lessArbitrary	::	(Generic			a	
		,GLessArbitrary	s	(Rep	a))	
	=>	CostGen	s		a	
<pre>lessArbitrary = genericLessArbitrary</pre>						

#### Generic implementation

class GLessArbitrarysdatatype wheregLessArbitrary:: CostGen s(datatype p)cheapest:: CostGen s(datatype p)

Binary tree only (2 lines of datatype).

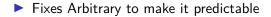
Implementation	Execution time	Lines
Generic arbitrary	$\infty$	2
Arbitrary with halving	177.0 $\mu$ s	8
Less arbitrary	341.8 $\mu$ s	1

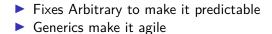
# Benchmarks (2)

Implementation	Execution time	Lines
Generic arbitrary	$\infty$	2
Arbitrary with halving	177.0 $\mu$ s	8
Less arbitrary	341.8 $\mu$ s	1
Feat	133.9 $\mu$ s	6+6

Feat needs 6 loc + 6 declarations of *driver*.







- Fixes Arbitrary to make it predictable
- Generics make it agile
- State argument for extra data in generator

- Fixes Arbitrary to make it predictable
- Generics make it agile
- State argument for extra data in generator
- Error message in case of loop

- Fixes Arbitrary to make it predictable
- Generics make it agile
- State argument for extra data in generator
- Error message in case of loop
- Simplicity can be copied to other languages