Random Testing of Purely Functional Abstract Datatypes

Stefan Holdermans

Abstract datatypes

- Defined only by their operations
- Independent from a concrete implementation
- Implementations can change without affecting client codes
- Client codes can easily switch between implementations

Algebraic specification

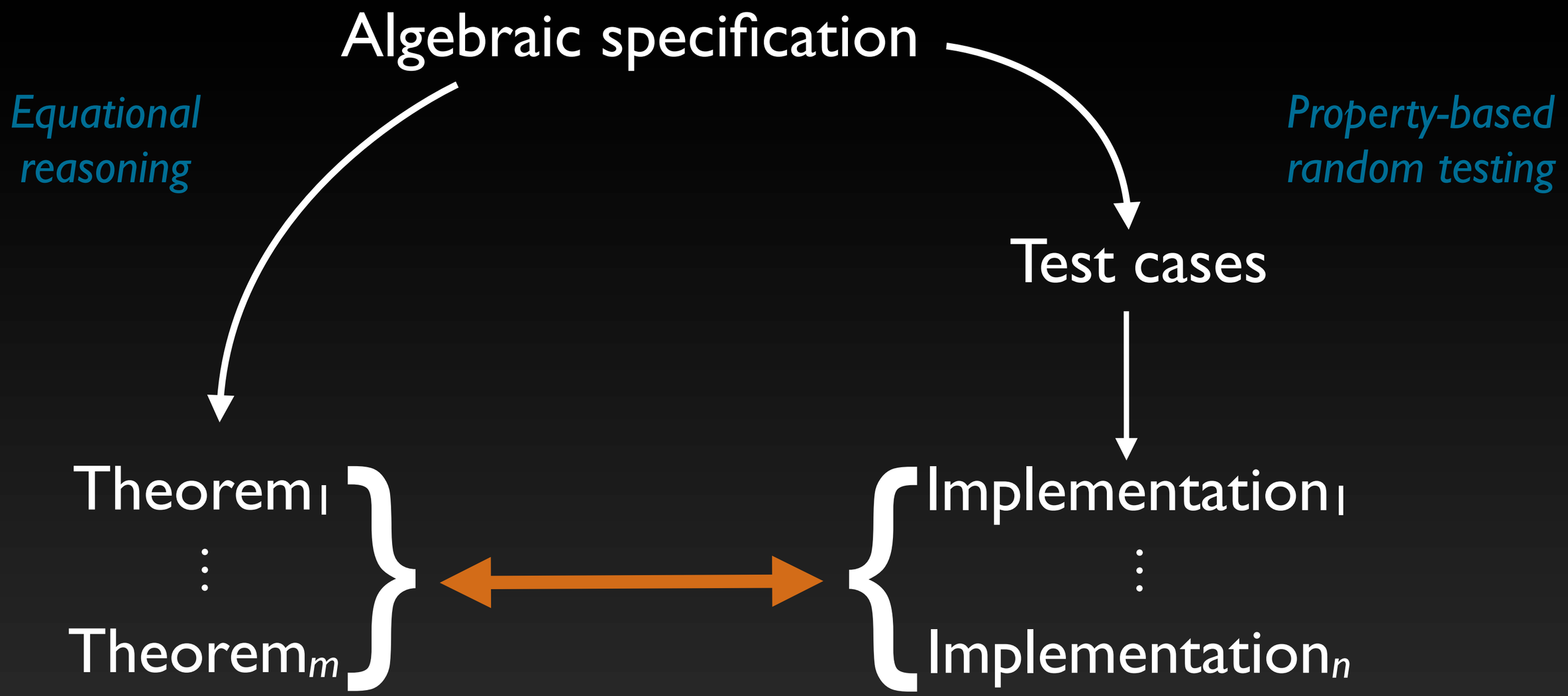
- Fitting framework for the definition of abstract datatypes
- In particular in the context of *purely* functional languages
- Enables equational reasoning

Equational reasoning

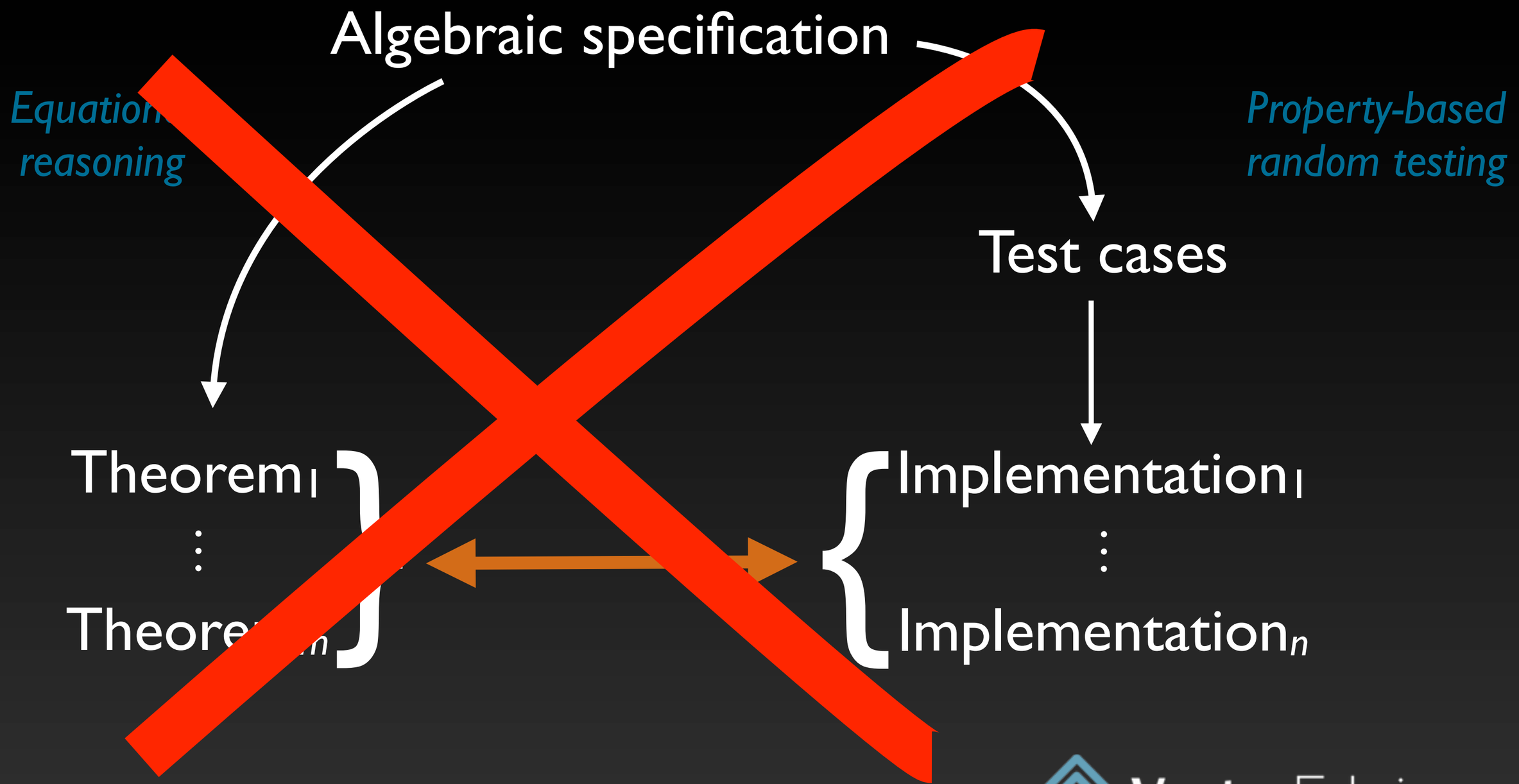
- Substituting “equals for equals”
- Deriving a whole class of theorems from only a handful of axioms
- Implementors only need to make sure that the axioms hold

Property-based random testing

- Map axioms to testable properties
- Obtain an arbitrary large set of executable test cases
- Excellent fit for purely functional languages
- QuickCheck, Gast, ...



Vector Fabrics



Vector Fabrics

Koopman et al. (IFL 2011)

“Without detailed study of the internals of the [implementation of an] ADT it is undecidable if a set of logical properties is sufficient [to assert its correctness with respect to the specification].”

Vector Fabrics

Outline

- A failing example
- What went wrong?
- A solution
- Conclusion

A failing example

FIFO queues: *signature*

sort:

```
Queue
```

operations:

```
empty     ::  Queue
enqueue   ::  Int → Queue → Queue
isEmpty   ::  Queue → Bool
front     ::  Queue → Int
dequeue   ::  Queue → Queue
```

FIFO queues: *axioms*

```
Q1:  isEmpty empty                = True
Q2:  isEmpty (enqueue x q)        = False
Q3:  front (enqueue x empty)      = x
Q4:  front (enqueue x q)          = front q                    (if isEmpty q = False)
Q5:  dequeue (enqueue x empty)    = empty
Q6:  dequeue (enqueue x q)        = enqueue x (dequeue q)      (if isEmpty q = False)
```

Vector Fabrics

A theorem about queues

```
isEmpty (dequeue (enqueue x empty)) =  True
```

Proof:

```
  isEmpty (dequeue (enqueue x empty)
=  {Q5}
  isEmpty empty
=  {Q1}
  True
```

Vector Fabrics

Another theorem about queues

```
front (dequeue
  (enqueue x (enqueue y (enqueue z empty)))) =  y
```

Proof:

```
  front (dequeue (enqueue x (enqueue y (enqueue z empty))))
=  { Q6, Q2 ; Q6, Q2 }
  front (enqueue x (enqueue y (dequeue (enqueue z empty))))
=  { Q5 ; Q4, Q2 }
  front (enqueue y empty)
=  { Q3 }
  y
```

Vector Fabrics

QuickCheck by example

```
> let p1 = property (λxs ys →
    reverse (xs ++ ys) == reverse ys ++ reverse xs)

> quickCheck p1
+++ OK, passed 100 tests

> let p2 = property (λx → reverse [x] == [])

> quickCheck p2
*** Failed! Falsifiable (after 1 test):
0
```

Testable properties for queues

```
q1 = property (isEmpty empty)
q2 = property (λx q → ¬(isEmpty (enqueue x q)))
q3 = property (λx → front (enqueue x empty) == x)
q4 = property (λx q → ¬(isEmpty q) ==> front (enqueue x q) == front q)
q5 = property (λx → dequeue (enqueue x empty) == empty)
q6 = property (λx q → ¬(isEmpty q) ==>
       dequeue (enqueue x q) == enqueue x (dequeue q))
```

Testable properties for queues

```
q1 = property (isEmpty empty)
q2 = property (λx q → ¬(isEmpty (enqueue x q)))
q3 = property (λx → front (enqueue x empty) == x)
q4 = property (λx q → ¬(isEmpty q) ==> front (enqueue x q) == front q)
q5 = property (λx → dequeue (enqueue x empty) == empty)
q6 = property (λx q → ¬(isEmpty q) ==>
       dequeue (enqueue x q) == enqueue x (dequeue q))
```

Vector Fabrics

Batched queues

```
data Queue = BQ [Int] [Int] deriving Show

bq [] r = BQ (reverse r) []
bq f r  = BQ f r

empty                 = bq [] []
enqueue x (BQ f r) = bq f (x : r)
isEmpty (BQ f r)    = null f
front (BQ f r)       = last f
dequeue (BQ f r)    = bq (tail f) r
```

Batched queues

```
data Queue = BQ [Int] [Int] deriving Show

bq [] r = BQ (reverse r) []
bq f r  = BQ f r

empty                = bq [] []
enqueue x (BQ f r) = bq f (x : r)
isEmpty (BQ f r)    = null
front (BQ f r)       = last f          -- incorrect!
dequeue (BQ f r)    = bq (tail f) r
```

Equality for batched queues

```
instance Eq Queue where
  q1 == q2  =  toList q1 == toList q2

toList (BQ f r) = f ++ reverse r
```

Testing batched queues

```
> mapM_ quickCheck [q1,q2,q3,q4,q5,q6]
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
```

What went wrong?

One more property...

```
> let q7 = property (λx y z →
    front (dequeue (enqueue x (enqueue y (enqueue z empty)))) == y)

> quickCheck q7
*** Failed! Falsifiable (after 2 tests):
0
1
0
```

But... q7 represents one of our theorems!

Vector Fabrics

Did we break equational reasoning?

Yes

Did we break equational reasoning?

- A stable basis for equational reasoning requires that operations are *invariant* under equality:

  $x = y \Rightarrow h(x) = h(y)$

- Invariance is what justifies "substituting equals for equals"

Vector Fabrics

Did we break equational reasoning?

```
> let qA = BQ [2,3] [5]

> let qB = BQ [2] [5,3]

> qA == qB
True

> front qA
3

> front qB
2
```

Vector Fabrics

A solution

Key idea

Systematically extend the set of testable properties
with properties for operation invariance

A first attempt

```
> let qq = property (λq q' →
    q == q' ∧ ¬(isEmpty q) ==>
    front q == front q')

> quickCheck qq
*** Gave up! Passed only 1 test.
```

A type of equivalent values

```
data Equiv a  =  a :==: a  deriving Show
```

For example:

```
> let eq  =  BQ [2,3] [5] :==: BQ [2] [5,3]
```

(More details in the paper)

Lifting out the equivalence check...

```
> let qq' = property (λ(q :==: q') →
    ¬(isEmpty q) ==>
    front q == front q')
```

Vector Fabrics

Lifting out the equivalence check...

```
> let qq' = property (λ(q :==: q') →
    ¬(isEmpty q) ==>
    front q == front q')

> quickCheck qq'
*** Failed! Falsifiable (after 4 tests):
BQ [-1,-2] [2] :==: BQ [-1] [2,-2]
```

Testable invariance properties

```
qq1 = property (λx (q :==: q') → enqueue x q == enqueue x q‘                )
qq2 = property (λ  (q :==: q') → isEmpty q == isEmpty q‘                    )
qq3 = property (λx (q :==: q') → ¬(isEmpty q) ==> front q == front q‘       )
qq4 = property (λx (q :==: q') → ¬(isEmpty q) ==> dequeue q == dequeue q')
```

Testing against all properties

```
> mapM_ quickCheck ([q1,q2,q3,q4,q5,q6] ++ [qq1,qq2,qq3,qq4])
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
+++ OK, passed 100 tests.
*** Failed! Falsifiable (after 5 tests):
BQ [-1,0] [1] :==: BQ [-1] [1,0]
+++ OK, passed 100 tests.
```

Vector Fabrics

Conclusion

Summary

- A framework for tests and proofs for purely functional ADTs
- An extension for dealing with implementations that are not UR
- Key idea: derive testable properties for operation invariance

Future work

- Assess and quantify impact on real-world applications
- Automatic derivation of testable properties from specifications
- EDSL for algebraic specifications of ADTs
- Testable properties for `Equiv`-generators