Random Testing ofPurely 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, ...
Algebraic specification

Equational reasoning

Test cases

Property-based random testing

\{ \text{Theorem}_1 \}
\{ \vdots \}
\{ \text{Theorem}_m \}

\{ \text{Implementation}_1 \}
\{ \vdots \}
\{ \text{Implementation}_n \}
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.
Outline

- A failing example
- What went wrong?
- A solution
- Conclusion
A failing example
FIFO queues: signature

sort:
Queue

operations:
empty :: Queue
enqueue :: Int \rightarrow Queue \rightarrow Queue
isEmpty :: Queue \rightarrow Bool
front :: Queue \rightarrow Int
dequeue :: Queue \rightarrow 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)
A theorem about queues

\[
\text{isEmpty (dequeue (enqueue x empty))} = \text{True}
\]

Proof:

\[
\begin{align*}
\text{isEmpty (dequeue (enqueue x empty))} & = \{ \text{Q5} \} \\
\text{isEmpty empty} & = \{ \text{Q1} \} \\
\text{True} &
\end{align*}
\]
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
QuickCheck by example

> let p1 = property (\xs ys ->
      reverse (xs ++ ys) == reverse ys ++ reverse xs)

> quickCheck p1
  +++ OK, passed 100 tests

> let p2 = property (\x -> reverse [x] == [])

> quickCheck p2
  *** Failed! Falsifiable (after 1 test):
  0
Testable properties for queues

q₁ = property \( (\text{isEmpty} \text{ empty}) \)
q₂ = property \( (\lambda x \ q \to \neg(\text{isEmpty} \ (\text{enqueue} \ x \ q))) \)
q₃ = property \( (\lambda x \to \text{front} \ (\text{enqueue} \ x \ \text{empty}) == x) \)
q₄ = property \( (\lambda x \ q \to \neg(\text{isEmpty} \ q) \implies \text{front} \ (\text{enqueue} \ x \ q) == \text{front} \ q) \)
q₅ = property \( (\lambda x \to \text{dequeue} \ (\text{enqueue} \ x \ \text{empty}) == \text{empty}) \)
q₆ = property \( (\lambda x \ q \to \neg(\text{isEmpty} \ q) \implies \)

\[ \text{dequeue} \ (\text{enqueue} \ x \ q) == \text{enqueue} \ x \ (\text{dequeue} \ q) \]
Testable properties for queues

\[ q_1 = \text{property } (\text{isEmpty empty}) \]
\[ q_2 = \text{property } (\lambda x \ q \to \neg (\text{isEmpty} \ (\text{enqueue} \ x \ q))) \]
\[ q_3 = \text{property } (\lambda x \rightarrow \text{front} \ (\text{enqueue} \ x \ \text{empty}) \ == \ x) \]
\[ q_4 = \text{property } (\lambda x \ q \to \neg (\text{isEmpty} \ q) \implies \text{front} \ (\text{enqueue} \ x \ q) \ == \ \text{front} \ q) \]
\[ q_5 = \text{property } (\lambda x \rightarrow \text{dequeue} \ (\text{enqueue} \ x \ \text{empty}) \ == \ \text{empty}) \]
\[ q_6 = \text{property } (\lambda x \ q \to \neg (\text{isEmpty} \ q) \implies \text{dequeue} \ (\text{enqueue} \ x \ q) \ == \ \text{enqueue} \ x \ (\text{dequeue} \ q)) \]
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 (\x 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!
Did we break equational reasoning?

Yes
Did we break equational reasoning?

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

\[ x = y \implies h(x) = h(y) \]

- Invariance is what justifies “substituting equals for equals”
Did we break equational reasoning?

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

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

> qA == qB
True

> front qA
3

> front qB
2
```
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...

```latex
> let qq' = property (λ(q :≡: q') → 
  ¬(isEmpty q) ==> 
  front q == front q')
```
Lifting out the equivalence check...

```haskell
> 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 (\( \lambda x \ (q :==: q') \rightarrow enqueue \ x \ q == enqueue \ x \ q' \)
qq2 = property (\( \lambda \ (q :==: q') \rightarrow isEmpty \ q == isEmpty \ q' \)
qq3 = property (\( \lambda x \ (q :==: q') \rightarrow \neg(isEmpty \ q) ==> front \ q == front \ q' \)
qq4 = property (\( \lambda x \ (q :==: q') \rightarrow \neg(isEmpty \ q) ==> dequeue \ q == dequeue \ q' \)
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.
+++ 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.
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