QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs

Koen Claessen and John Hughes

Presented by: Jue Wang

What is QuickCheck?

- QuickCheck is an automatic specification based testing tool for Haskell.
- The programmer provides specifications as properties expressed in Haskell.
- The properties are verified on randomly generated test data.

A Simple Example

```
prop_RevApp xs ys =
    reverse (xs ++ ys) == reverse ys ++ reverse xs
      where types = (xs ::[Int], ys ::[Int])
```

A Simple Example

```
prop_RevApp xs ys =
    reverse (xs ++ ys) == reverse ys ++ reverse xs
      where types = (xs ::[Int], ys ::[Int])
```

```
Test> quickCheck prop_RevApp
OK, passed 100 tests.
```

A Bad Property

```
prop_RevAppBad xs ys =
    reverse (xs++ys) == reverse xs ++ reverse ys
        where types = (xs ::[Int], ys ::[Int])
```

A Bad Property

```
prop_RevAppBad xs ys =
    reverse (xs++ys) == reverse xs ++ reverse ys
        where types = (xs ::[Int], ys ::[Int])

Test> quickCheck prop_RevAppBad
Falsifiable, after 4 tests:
[-3,-4,-4]
[-4,-1,1,1]
```

Conditional Properties

<condition> ==> <property>

```
prop_Insert x xs =
    ordered xs ==> ordered (ins x xs)
        where types = (x ::[Int], xs ::[Int])
```

Conditional Properties

<condition> ==> <property>

```
prop_Insert x xs =
    ordered xs ==> ordered (ins x xs)
        where types = (x ::[Int], xs ::[Int])
```

```
Test> quickCheck prop_Insert
OK, passed 100 tests.
```

Pitfall of Conditional Properties

```
insBad a [] = [a]
insBad a y
   | (length y) > 4 = y ++ [a]
   | otherwise = ins a y

prop_InsertBad x xs =
   ordered xs ==> ordered (insBad x xs)
      where types = (x ::[Int], xs ::[Int])
```

Pitfall of Conditional Properties

```
insBad a [] = [a]
insBad a y
   | (length y) > 4 = y ++ [a]
   | otherwise = ins a y

prop_InsertBad x xs =
   ordered xs ==> ordered (insBad x xs)
      where types = (x ::[Int], xs ::[Int])

Test> quickCheck prop_InsertBad
OK, passed 100 tests.
```

What's wrong?

QuickCheck provides combinators to investigate test case distributions.

- <condition> ‘trivial‘ <property>
- classify <condition> <string> $ <property>
- collect <expression> $ <property>

Use verboseCheck to see all the test cases.

Observing Test Case Distribution

```
prop_InsertBad x xs =
    ordered xs ==>
        collect (length xs) $
        classify (ordered (x:xs)) "at-head" $
        classify (ordered (xs++[x])) "at-tail" $
        ordered (insBad x xs)
```

Observing Test Case Distribution

```
Test> quickCheck prop_InsertBad
47% 0, at-head, at-tail.
13% 1, at-tail.
12% 2, at-tail.
10% 1, at-head.
6% 2.
5% 1, at-head, at-tail.
3% 2, at-head.
2% 4.
2% 3.
```

Fix # 1 (Add Another Condition)

```
prop_InsFix1 x xs =
    ((ordered xs) && ((length xs) > 4)) ==>
        ordered (insBad x xs)
```

Fix # 1 (Add Another Condition)

```
prop_InsFix1 x xs =
    ((ordered xs) && ((length xs) > 4)) ==>
        ordered (insBad x xs)
```

```
Test> quickCheck prop_InsFix1
Arguments exhausted after 0 tests.
```

Generating Random Test Data

```
class Arbitrary a where
    arbitrary :: Gen a
```

- QuickCheck provides generators for most base types such as Int, Char, Float, List.
- QuickCheck provides combinators for building custom generators.
  - oneof :: [Gen a] -> Gen a
    oneof [return Heads, return Tails]
  - frequency :: [(Int, Gen a)] -> Gen a
    frequency [(1, return Heads), (2, return Tails)]
  - two :: Gen a -> Gen (a, a)

Fix # 2 (Using a Custom Generator)

```
orderedList =
    (>>=)(frequency [(1, return []),
                     (7, liftM2 (:) arbitrary arbitrary)])
          (\ a -> return (sort a))

prop_InsFix2 x =
    forAll orderedList $ \ xs -> ordered (insBad x xs)
```

Fix # 2 (Using a Custom Generator)

```
orderedList =
    (>>=)(frequency [(1, return []),
                     (7, liftM2 (:) arbitrary arbitrary)])
          (\ a -> return (sort a))

prop_InsFix2 x =
    forAll orderedList $ \ xs -> ordered (insBad x xs)
```

```
Test> quickCheck prop_InsFix2
Falsifiable, after 9 tests:
-4
[-6,-1,0,0,2,4,6,7]
```

Generators for User Defined Types

```
data Tree a =
    Leaf a
  | Branch (Tree a) (Tree a)

instance Arbitrary a => Arbitrary (Tree a) where
    arbitrary = frequency
                  [ (1, liftM Leaf arbitrary)
                  , (2, liftM2 Branch arbitrary arbitrary)]
```

Generators for User Defined Types

```
data Tree a =
    Leaf a
  | Branch (Tree a) (Tree a)

instance Arbitrary a => Arbitrary (Tree a) where
    arbitrary = frequency
                  [ (1, liftM Leaf arbitrary)
                  , (2, liftM2 Branch arbitrary arbitrary)]

Test> quickCheck prop_TreeDepth
ERROR - Garbage collection fails to reclaim sufficient
space
```

Using Size

```
sized :: (Int -> Gen a) -> Gen a

instance Arbitrary a => Arbitrary (Tree a) where
    arbitrary = sized arbTree

arbTree 0 = liftM Leaf arbitrary
arbTree n =
    frequency
        [(1,liftM Leaf arbitrary)
        ,(4,liftM2 Branch (arbTree (n `div` 2))
                          (arbTree (n `div` 2)))]
```

Generating Random Functions

```
class Coarbitrary a where
    coarbitrary :: a -> Gen b -> Gen b

promote :: (a -> Gen b) -> Gen (a -> b)

instance (Coarbitrary a, Arbitrary b) =>
                        Arbitrary (a -> b) where
   arbitrary =
      promote (\a -> coarbitrary a arbitrary)
```

The End

more information at:

http://www.cs.chalmers.se/ rjmh/QuickCheck/