Exhaustive testing *of* property testers

Auke Booij

March 2022

Property testers only evaluate properties at a finite number of inputs, and so even a property that passed testing might still not hold for all inputs. We can also phrase properties *of* property testers. Perhaps surprisingly, this higher-order problem is decidable using Escardó's infinite search [2], subject to some caveats.

Property testing is used to test program correctness over a range of inputs [1]. In a highly simplified setting, the property we want to test is given to the property tester as a function. For example, we can phrase the property that 0 is a right identity of + as the Haskell function:

```haskell
prop_assoc :: Natural -> Bool
prop_assoc i = (i + 0 == i)
```

A property tester such as QuickCheck would then evaluate this function several times with different input values.

- If the function returned `True` every time, the property is considered satisfied.
- If a counterexample is found, then this is reported to the user.

We can make this more concrete by defining an abstract property tester as follows.

```haskell
type Property range = range -> Bool
data Result counterexample
  = Success
  | Failure counterexample
type Tester a = Property a -> Result a
```

With the above definitions, it is possible to phrase correctness criteria *of* such `Tester`s.

Here's one concrete example of a correctness criterion *of* property testers: if running a property tester results in a `Failure`, i.e. if it found a counterexample to a property, then the property better be refuted by that countexample. Below is the Haskell encoding of that.

```haskell
prop_failureActuallyFails :: Tester Natural -> Property (Property Natural)
prop_failureActuallyFails tester prop =
  case tester prop of
    Success -> True
    -- The property tester found a counterexample, so that better be one
    Failure n -> not (prop n)
```

Having arrived at this idealized view of property testing, we can state the central idea of this abstract: such properties *of* property testers are testable *exhaustively*.

In the talk, I'll also detail some other properties that can be tested exhaustively using this technique. The general, though perhaps unhelpful, answer is that we can test those properties that can be instantiated as a function `Tester Natural -> Property (Property Natural)`.

In other words, we can *decide*, for a given value `Tester Natural`, whether `prop_failureActuallyFails` is satisfied for *all* properties of *natural numbers*, i.e. for all inputs `Property Natural`. This is a straightforward application of the exhaustibility of Cantor space [2].

To be clear, this exhaustibility of Cantor space does not imply that we can enumerate all elements of the type `Natural -> Bool`, as this would certainly not be a finite list.

There are two caveats:

1. This only allows to decide correctness criteria of property testers instantiated at (a type isomorphic to) the natural numbers. However, in general, property testers may be polymorphic. What can we say for instantiations at other types?

   (a) We can consider other types X that make $X \to \mathsf{Bool}$ exhaustible: we can exhaustively test property testers instantiated at any such X. However, this is a strong requirement on X, and so only has limited applications.

   (b) Normally, we use parametricity [3] to generalize properties of polymorphic functions from one instantiation to another. However, this works only to a very limited extent, since most property testers only work for a restricted selection of range types (e.g. types for which elements can be generated), and so are not fully polymorphic functions. So the free theorems yielded by parametricity are limited, roughly in proportion to the restrictions on the range type of the property tester.

2. Normally, property testers use (pseudo)random generators to choose input values from the range. Such a pseudorandom property tester can be interpreted as a deterministic one by choosing a seed, and thus randomized property testers correspond to a *family* of deterministic property testers. Thus, while we could exhaustively test a property tester *with a fixed seed*, strictly speaking this proves nothing about the rest of the family.

This idea has been implemented for QuickCheck; no bugs were found. The presence of IO was not problematic in the case of QuickCheck, but in general it could be.

References

[1] K. Claessen and J. Hughes. "QuickCheck: a lightweight tool for random testing of Haskell programs". In: *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada, September 18-21, 2000*. Ed. by M. Odersky and P. Wadler. ACM, 2000, pp. 268–279. DOI: 10.1145/351240.351266.

[2] M. H. Escardó. "Exhaustible Sets in Higher-type Computation". In: *Log. Methods Comput. Sci.* 4.3 (2008). DOI: 10.2168/LMCS-4(3:3)2008.

[3] P. Wadler. "Theorems for Free!" In: *Proceedings of the fourth international conference on Functional programming languages and computer architecture, FPCA 1989, London, UK, September 11-13, 1989*. 1989, pp. 347–359. DOI: 10.1145/99370.99404.