What is Property Based Testing?

May 14, 2016

•

David R. MacIver

I get asked this a lot, and I write property based testing tools for a living, so you'd think I have a good answer to this, but historically I haven't. This is my attempt to fix that.

Historically the definition of property based testing has been "The thing that QuickCheck does". As a working definition this has served pretty well, but the problem is that it makes us unable to distinguish what the essential features of property-based testing are and what are just accidental features that appeared in the implementations that we're used to.

As the author of a property based testing system which diverges quite a lot from QuickCheck, this troubles me more than it would most people, so I thought I'd set out some of my thoughts on what property based testing is and isn't.

This isn't intended to be definitive, and it will evolve over time as my thoughts do, but it should provide a useful grounding for further discussion.

There are essentially two ways we can draw a boundary for something like this: We can go narrow or we can go wide. i.e. we can restrict our definitions to things that look exactly like QuickCheck, or things that are in the same general family of behaviour. My inclination is always to go wide, but I'm going to try to rein that in for the purpose of this piece.

But I'm still going to start by planting a flag. The following are *not* essential features of property based testing:

1. Referential Transparency.
2. Types
3. Randomization
4. The use of any particular tool or library

As evidence I present the following:

1. Almost every property based testing library, including but not limited to Hypothesis and QuickCheck (both Erlang and Haskell).
2. The many successful property based testing systems for dynamic languages. e.g. Erlang QuickCheck, test.check, Hypothesis.
3. SmallCheck. I have mixed feelings about its effectiveness, but it's unambiguously property-based testing.
4. It's very easy to hand-roll your own testing protocols for property-based testing of a particular result. For example, I've previously done this for testing a code formatter: Run it over a corpus (more on whether running over a corpus "really" counts in a second) of Python files, check whether the resulting formatted code satisfies PEP8. It's classic property-based testing with an oracle.

So that provides us with a useful starting point of things that are definitely property based testing. But you're never going to find a good definition by looking at only positive examples, so let's look at some cases where it's more arguable.

First off, lets revisit that parenthetical question: Does just testing against a large corpus count?

I'm going to go with "probably". I think if we're counting SmallCheck we need to count testing against a large corpus: If you take the first 20k outputs that would be generated by SmallCheck and just replay the test using those the first N of those each time, you're doing exactly the same sort of testing. Similarly if you draw 20k outputs using Hypothesis and then just randomly sample from them each time.

I think drawing from a small, fixed, corpus probably *doesn't* count. If you could feasibly write a property based test as 10 example based tests in line in your source code, it's probably really just example based testing. This boundary is a bit, um, fuzzy though.

On which note, what about fuzzing?

I have previously argued that fuzzing is just a form of property-based testing - you're testing the property "it doesn't crash". I think I've reversed my opinion on this. In particular, I think the style of testing I advocate for getting started with Hypothesis, *probably* doesn't count as property based testing.

I'm unsure about this boundary. The main reason I'm drawing it here is that they do feel like they have a different character - property based testing requires you to reason about how your program should behave, while fuzzing can just be applied to arbitrary programs with minimal understanding of their behaviour - and also that fuzzing feels somehow more fundamental.

But you can certainly do property based testing using fuzzing tools, in the same way that you can do it with hand-rolled property based testing systems - I could have taken my Python formatting test above, added python-afl to the mix, and that would still be property based testing.

Conversely, you can do fuzzing with property-based testing tools: If fuzzing is not property based testing then not all tests using Hypothesis, QuickCheck, etc. are property based tests. I'm actually OK with that. There's a long tradition of testing tools being used outside their domain - e.g. most test frameworks originally designed as unit testing tools end up getting being used for the whole gamut of testing.

So with that in mind, lets provide a definition of fuzzing that I'd like to use:

Fuzzing is feeding a piece of code (function, program, etc.) data from a large corpus, possibly dynamically generated, possibly dependent on the results of execution on previous data, in order to see whether it fails.

The definitions of "data" and "whether it fails" will vary from fuzzer to fuzzer - some fuzzers will generate only binary data, some fuzzers will generate more structured data. Some fuzzers will look for a process crash, some might just look for a function returning false.

(Often definitions of fuzzing focus on "malformed" data. I think this is misguided and fails to consider a lot of things that people would obviously consider fuzzers. e.g. CSmith is certainly a type of fuzzer but deliberately only generates well formed C programs).

And given that definition, I think I can now provide a definition of property-based testing:

Property based testing is the construction of tests such that, when these tests are fuzzed, failures in the test reveal problems with the system under test that could not have been revealed by direct fuzzing of that system.

(If you feel strongly that fuzzing *should* count as property-based testing you can just drop the 'that could not have been etc.' part. I'm on the fence about it myself.)

These extra modes of failure then constitute the properties that we're testing.

I think this works pretty well at capturing what we're doing with property based testing. It's not perfect, but I'm pretty happy with it. One of the things I particularly like is that it makes it clear that property-based testing is what *you* do, not what the computer does. The part that the computer does is "just fuzzing".

Under this point of view, a property-based testing library is really two parts:

1. A fuzzer.
2. A library of tools for making it easy to construct property-based tests using that fuzzer.

Hypothesis is very explicitly designed along these lines - the core of it is a structured fuzzing library called Conjecture - which suggests that I may have a bit of bias here, but I still feel that it captures the behaviour of most other property based testing systems quite well, and provides quite a nice middle ground between the wider definition that I wanted and the more tightly focused definition that QuickCheck orients people around.
