Property-Based Testing in the Wild!

An Analysis of QuickCheck usages in Open-Source Haskell Projects

CSE3000 Research Project

Ye Zhao (y.zhao-33@student.tudelft.nl)
Supervisors: Andreea Costea, Sára Juhošová
Delft University of Technology

Background & Motivation

What is Property-based testing (PBT)?
A powerful approach to software testing that verifies the correctness of programs by checking general properties over large sets of automatically generated inputs.

Why QuickCheck?
QuickCheck, introduced in 2000 by Claessen and Hughes[1], is one of the first and most influential PBT tools. Heavily used in the Haskell community, it also inspired many testing frameworks in other languages, such as Hypothesis in Python.

Research Questions

Main question:
How is Property-Based Testing with QuickCheck applied in real-world Haskell open-source projects?

- RQ 1. What kinds of properties are typically tested using QuickCheck?
- RQ 2. Which types of quantifiers and logical connectives are used in these properties?
- RQ 3. How does property-based testing complement other testing strategies such as unit tests?
- RQ 4.How and when are generators implemented?
- RQ 5. In which scenarios do developers explicitly define shrinking strategies?

Methodology

Data Collection:

- GitHub API to identify Haskell repos using QuickCheck
- Manual filtering for active and meaningful projects (actively maintained, high stars, etc.)

Data Analysis:

- Sampling. Stratified hand-random draw: ⇒ 217 for manual coding.
- Classify property types based on existing literature[2]
- Check Logical constructs (forAll, ==>, .&&./.||.)
- Detect generator and shrinking usage
- Visualisations

References

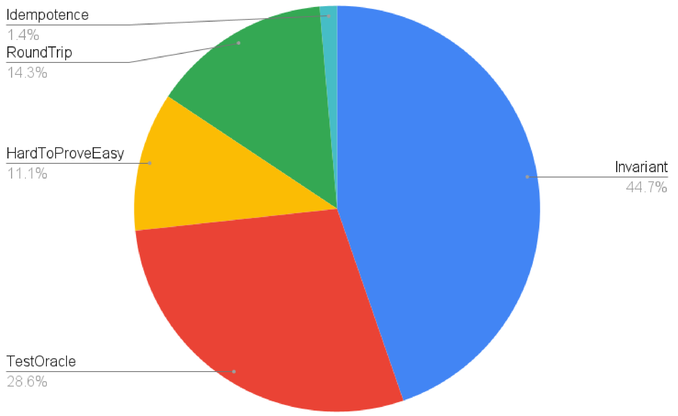
[1] K. Claessen and J. Hughes, "QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs," Proc. ICFP 2000, pp. 268–279.
[2] https://fsharpforfunandprofit.com/posts/property-based-testing-2/

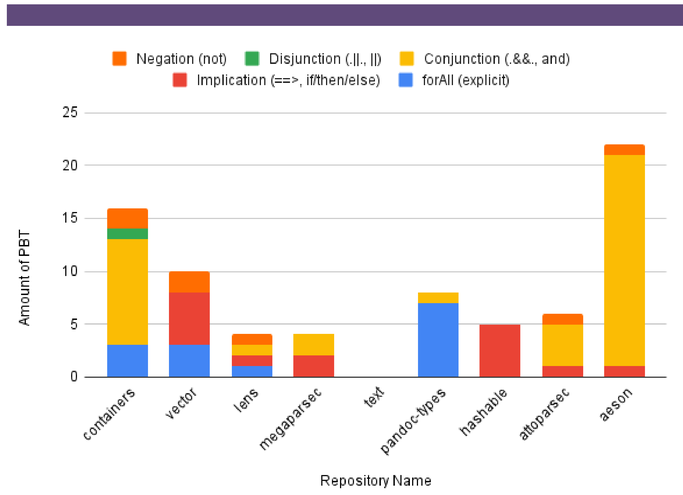
Limitations

- Research based on open-source GitHub repositories. This might miss how QuickCheck is used in private or enterprise projects.
- Static extractor. Regex scan misses Template-Haskell-registered tests; a few properties could be uncounted.
- The manual classification of properties and generators could introduce subjectivity.
- Manual coding may contain small counting slips (±2 %).
- We look at the latest release of each repo, not how property suites evolve over time.

Findings

Repository Name	Total PBT	Other tests	PBT Percentage
aeson	287	2163	13.3 %
attoparsec	70	0	100 %
containers	1171	423	73.5 %
hashable	21	15	58.3 %
lens	25	55	31.3 %
megaparsec	386	573	40.3 %
pandoc-types	23	114	17.2 %
text	492	262	65.3 %
vector	171	2637	6.1 %





Conclusion

Three high-leverage idioms dominate.
Invariant, Test-Oracle, and Round-Trip cover ≈ 87 % of all sampled properties.

Generator effort pays off.
Just 12 module-level generators power over hundred tests (56 %); 4 defined instance shrink appear in 25 % sampled test.

Division of labor.
QuickCheck rules pure, deterministic logic; example / golden tests lock down I/O, UX, and performance details.

Future

Auto-derive recursive generators.
Type-driven synthesis could replace the dozen hand-written generators that already power half the tests.

Shrink user studies. Measure debugging time saved by custom shrinkers and defaults.

Cross-ecosystem replication.
Apply the same codebook to more language and framework to test if the "high-leverage trio" is language-agnostic.