Specification Based Testing with QuickCheck

(Tutorial Talk)

John Hughes
Chalmers University of Technology and QuviQ AB

ABSTRACT

QuickCheck is a tool which tests software against a formal specification, reporting discrepancies as minimal failing examples. QuickCheck uses *properties* specified by the developer both to generate test cases, and to identify failing tests. A property such as

$$\forall \texttt{xs} : \texttt{list(int())}.\ \texttt{reverse(reverse(xs))} = \texttt{xs}$$

is tested by generating random lists of integers, binding them to the variable `xs`, then evaluating the boolean expression under the quantifier and reporting a failure if the value is `false`. If a failing test case is found, QuickCheck "shrinks" it by searching for smaller, but similar test cases that also fail, terminating with a minimal example that cannot be shrunk further. In this example, if the developer accidentally wrote

$$\forall \texttt{xs} : \texttt{list(int())}.\ \texttt{reverse(xs)} = \texttt{xs}$$

instead, then QuickCheck would report the list $[0, 1]$ as the minimal failing case, containing as few list elements as possible, with the smallest absolute values possible. The approach is very practical: QuickCheck is implemented just a library in a host programming language; it needs only to execute the code under test, so requires no tools other than a compiler (in particular, no static analysis); the shrinking process "extracts the signal from the noise" of random testing, and usually results in very easy-to-debug failures.

First developed in Haskell by Koen Claessen and myself, QuickCheck has been emulated in many programming languages, and in 2006 I founded QuviQ to develop and market an Erlang version. Of course, customers' code is much more complex than the simple `reverse` function above, and requires much more complex properties to test it. The challenge in applying QuickCheck to real code is in finding ways to formulate properties that are simple enough for people to use easily, concise enough to make property-based testing cost-effective, and avoid the trap of replicating the mistakes of the implementation in the specification. To this end we have extended QuickCheck with state machine formalisms, and standardized serializability properties that can expose harmful race conditions in concurrent code. I will present examples using these formalisms, and discuss our experiences of applying property-based testing in the telecoms, automotive, and distributed database industries.

SHORT BIOGRAPHY

John Hughes began research in functional programming in 1980 as a D.Phil. student at the University of Oxford, graduating in 1983. He wrote *Why Functional Programming Matters* in 1985 while a post-doc at Chalmers University—a functional manifesto which is still one of the most widely read papers in the field. He took up a Chair at Glasgow University from 1985–1992, where he was a founder member of the Haskell design committee (and later its co-Chair for the Haskell 98 standard). In 1992 he moved to a Chair at Chalmers University, Gothenburg. He and Koen Claessen began work on QuickCheck in 1998—initially just for fun—and published the first paper on it in 2000. In 2006, he and Thomas Arts founded Quviq AB to commercialize the QuickCheck approach, and since then he has shared his time between Chalmers and Quviq. In 2010, the first QuickCheck paper received the ACM SIGPLAN award for the Most Influential Paper of ICFP 2000.