QuickCheck for Testing DrScheme Programs

Yun Jin Kang (Matthew Flatt)
School of Computing
University of Utah

Testing is always required to guarantee software quality, but specific programming techniques and tools can make testing easier or more difficult. The functional style of programming (which avoids mutation of records and variables) is often more easily tested, since the result of a part of the program depends only on inputs, not on the state of other parts of the program.

QuickCheck is the lightweight tool for random testing. QuickCheck randomly generates suitable inputs, and it can check the result of tests using a programmer-specified function. The given check function need not test full correctness of the output; even simple one-checks, combined with randomly generated input, can provide a useful level of testing. QuickCheck provides combinators to define function properties, observe the distribution of test data, and define test data generators.

My work built on QuickCheck, which is Koen and John's lightweight tool for random testing of Haskell programs. In Haskell, QuickCheck is especially convenient to use, because it exploits types that exist in the program text already. However, DrScheme is the functional style of programming, and it is not statically typed. Although Scheme does not impose on a type system on its programs, many programs fit into a Haskell-like type system, anyway. Furthermore, as a matter of good style, programmers often write types in comments. Even better, they may write dynamically checked contracts using DrScheme's contract system. Contracts are a kind of lightweight, executable specification. Unlike informal types in comments, contracts are a part of the running program.

I developed DrScheme's contract system which relies on types and integrated QuickCheck-like functionality with DrScheme in such a way as to encourage contract annotations, and then to exploit contracts to automatically generate test inputs. Since contracts can include run-time checks that are more expensive than types, I expected that contracts could provide both the type information needed to generate tests and the checking function used to validate test executions.