Discussion questions for
QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs

COMP 150 - Applied Functional Programming
October 10, 2012

Fundamentals

(1) How does the Gen abstraction defined in Section 3 relate to the abstractions we discussed yesterday?

(2) Sections 3.1 and 3.2 show two different uses of a polymorphic function liftM2. If I remind you that

\[
\begin{aligned}
(,) &:: \text{forall a b . a \to b \to (a, b)} \\
(\cdot) &:: \text{forall a . a \to [a] \to [a]} \\
\text{arbitrary} &:: \text{forall a :: Gen a \Rightarrow a}
\end{aligned}
\]

then

(a) What do you suppose is the type of liftM2?

(b) Given the information that liftM2 is part of the Haskell language standard and not part of QuickCheck, can you imagine a more general type?

(c) There is also a liftM. What do you imagine its type is?

(d) Could any of these functions be used to solve the coin-flip problem from yesterday?

(3) In section 4 on the implementation of QuickCheck, look at the type Property and at the class definition for class Testable.

(a) What is the type of property?

(b) If the quickCheck function is applied to a value of type Bool, does the application type check? If not, why not? If so, what happens when the code runs?

(c) Why is it necessary to run quickCheck on functions returning Property rather than functions returning Bool?

Analysis

(4) Section 2.2 describes a way to formulate properties that quantify over functions, using extensional equality. Section 2.5 describes a way to test properties of infinite lists.

(a) How are these examples related?

(b) What sorts of theorems might justify these examples? 

(Hint: if a relevant theorem has computational content, the associated computations might not terminate.)

Application, Part I: Predicates

(5) Define an implies function that is overloaded on predicates of arbitrary arity. Do not extend your Predicate type class.

(6) Given your code for operating on predicates of arbitrary arity, try using QuickCheck to test these properties:

\[
\begin{aligned}
\text{even} &::= \text{complement \ odd} \\
(<) &::= \text{complement \ (>)} \\
\forall p . (\text{even \ 'conjoin' \ p}) &\implies p \\
\forall p . (\text{even \ 'disjoin' \ p}) &\implies p
\end{aligned}
\]

(7) Extend QuickCheck extensional equality to functions of arbitrary arity. In particular, if you have two functions \( f \) and \( g \) which both have type \( a \to b \to \cdots \to n \to r \), and if argument types \( a, b, \ldots, n \) are in class Arbitrary, and if the result type \( r \) is in class Eq then it should be possible to define an operator \( \equiv \equiv \) such that the expression \( f \equiv \equiv g \) is well typed.

(a) What should the type of \( f \equiv \equiv g \) be?

(b) Define \( \equiv \equiv \) by introducing a new type class and suitable instance declarations.

(8) Use your new extensional equality to test the following properties:

\[
\begin{aligned}
\text{even} &\equiv \text{complement \ odd} \\
(<) &\equiv \text{complement \ (>)} \\
\forall p . (\text{complement \ p \ 'conjoin' \ p}) &\equiv \text{falsehood} \\
\forall p . (\text{complement \ p \ 'disjoin' \ p}) &\equiv \text{truth} \\
\forall p . (\text{complement \ p \ 'disjoin' \ truth}) &\equiv p
\end{aligned}
\]

The last three properties are polymorphic and overloaded. Please test each property at multiple arities. You will need to use a type signature; if you like, you can follow the example at the end of Section 2.1.
Application, Part II: DVD Packing

(9) Section 2.4 briefly mentions the idea of a “trivial” test. What test inputs to the DVD-packing algorithm do you think would lead to “trivial” tests?

(10) What properties can you think of that the DVD-packing algorithm should satisfy?

(a) What are the properties?

(b) Which of the properties can be coded using QuickCheck?

(c) Code them.

(11) To test DVD packing, what test-case generator would you want to use? (Hint: the example I gave you is not a very challenging test.)