Discussion questions for *QuickCheck*

COMP 150FP

October 4, 2013

Language design

Answer the first question *quickly*:

1. QuickCheck expresses universal algebraic properties such as

   ```
   forall x . forall y . forall z . (x + y) + z == x + (y + z)
   ```

   Why didn't the designers of QuickCheck extend Haskell with an explicit `forall` construct?

Application

2. QuickCheck works with *representations*, but some data types are *abstract*. In functional languages, abstract data types are typically specified by means of algebraic laws.

   (a) Write algebraic laws for the specification of a queue.

   (b) Explain how you would use QuickCheck to test a queue whose value constructors are hidden.

3. What would `coarbitrary` be used for? Imagine a scenario where `coarbitrary` is used and its use leads to a test-case failure. What happens next?

4. From the talk last week, you know that QuickCheck continues to be used. I'd like you to imagine how it may have evolved from 2000 to 2013. To provide structure for your imagination, revisit Section 5 of the paper, which contains almost three pages of case studies.

   - How do you imagine that QuickCheck may have evolved or been improved over the years?
   - What do you imagine may still need to be improved?

Questions to take home and ponder

5. As both the authors and some readers note, QuickCheck defines type-indexed generation functions, and to get the right generation functions, you have to tweak the types by introducing `newtype` declarations. How inconvenient is this going to be to the programmer? What strategies would be useful to mitigate the inconvenience? Could the inconvenience be further mitigated if we were to abandon the within-pure-Haskell model and use an external tool to analyze programs or add instrumentation to them?

6. Agree on a reasonable decomposition of QuickCheck into components, and evaluate the merits and/or contribution of each component independently. Reach consensus on the strength of each contribution. Are any of these components useful by themselves?

7. In what sense is it possible, at least in principle, for QuickCheck to evaluate test *coverage*?

   In what sense is it *impossible*, even in principle, for QuickCheck to evaluate test coverage?

8. Which do you find easier to explain, the `promote` function or the `coarbitrary` method?

9. Would it be clearer or more confusing to put the overloaded `arbitrary` and `coarbitrary` functions in a single type class? Would such a merger create more or less work for the programmer, or about the same?

10. Is there a mechanical way to determine the proper use of `variant` based on the structure of the type? More generally, if we had a mechanism to use to automate `deriving`, would it be possible to automatically derive `Coarbitrary`, introducing `oneof` and `variant` based on the structure of the type? If so, would it be a good idea?