LVA 185.A05 Advanced Functional Programming (SS 2020)

Self-Assessment Test 5

Wednesday, 22 April 2020

Topics: Part III, Chapters 5 and 6

Testing, Verification

(No submission, no grading)

Part III, Chapter 5 'Testing'

1. What are arguments in favour of testing? What are arguments in favour of verification?
2. What are properties or features a test system should fulfil to call it a system for systematic testing?
3. What are possibilities we have for defining the meaning of a program? What are advantages, disadvantages of these possibilities?
4. What are possibilities to specify program properties of Haskell programs in QuickCheck? How do they differ in their pros and cons?
5. Define a QuickCheck property allowing to check, if floating point addition is commutative.
6. Experience shows that QuickCheck to reveal common kinds of flaws in occuring often during program development. Name a few.
7. Is QuickCheck an appropriate tool for testing that the initial and the final implementation of a functional pearl problem are equivalent? Why? Or, why not? Or, does it depend on the problem under consideration?
8. Compare the approaches of testing against abstract models and algebraic specifications. Illustrate the idea underlying these approaches (also) by an example.
9. What are methods offered by QuickCheck to control the size of test data it generates?
10. What are reporting facilities supported by QuickCheck?
11. Specify a QuickCheck property allowing to test if a function for list reversal twice yields the original argument.
12. Extend the property definition such that the length of lists is reported together with the percentage of cases in which they are used as test data, i.e., the report shall yield a histogram of the distribution of the length of lists used as test data.
13. Write a QuickCheck property allowing to tes tthat `binom_stpg` and `binom_memo` of *Assignment 1* are functionally equivalent. Make sure that only non-negative integers are generated as test cases.
14. Can you refine the property definition such that the generated test data for n and k satisfy the inequality $0 \leq k \leq n$?
15. Turing award winner Tony Hoare said that *rigorous testing regimes rapidly persuade error-prone programmer (like him) to remove themselves from the profession [of programming]*. Do you agree? Absolutely? To some extent? Wh?
16. Working with QuickCheck you sometimes get messages like *Arguments exhausted after 68 tests.* Why? What does it mean? Is a QuickCheck terminating with such a message useless?
17. Give an example, where a QuickCheck run is likely to terminate with such a message.
18. QuickCheck supports random testing, i.e., testing with randomly generated test data. Richard Hamlet investigated the efficacy of random testing in the 1990s. What were some of his key findings?

19. QuickCheck has been presented in 2000. Is it still unmodified since then? If not, name a few examples of changes.

20. Given the possibilities QuickCheck offers for systematic program testing, it is a small (in terms of lines of code) program, library and a good example for demonstrating the expressiveness of functional programming. Do you agree?

Part III, Chapter 6 'Verification'

1. The Fibonacci numbers F_i, $i \geq 0$, are given by the recursion:

   $$F_0 = 0,\ F_1 = 1, F_{n+2} = F_{n+1} + F_n \quad \text{for } n \geq 0$$

   Consider claim (C):

   $$\forall\, n > 1.\ F_{n+1}F_{n-1} - F_n^2 = (-1)^n \qquad (C)$$

   Prove or refute claim (C).

2. In case you start proving claim (C), what proof principle seems (most) appropriate? Why?

3. Properties valid for lists need not hold for streams.

4. Properties valid for streams do also hold for lists.

5. Properties valid for every PL-approximant of a stream do also hold for the stream; the reverse does not hold.

6. Give one statement each where

   (a) natural induction

   (b) strong induction

   (c) structural induction

   seems (most) appropriate for proving it.

7. What is the principle of natural induction? Explain the various parts it is composed of.

8. Prove:

   $$\forall\, n \in I\!N.\ \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

9. What is the German term for strong induction?

10. What is *Agda*? What is it good for?

11. What are partial lists? What are they good for?

12. What are proof principles at our disposal for proving properties of defined lists?

13. What changes, adaptions are required in these proof principles if lists may contain undefined values?

14. Explain why the value of an expression can be undefined.

15. What are proof principles at our disposal for proving equality of streams? How does one proceed when using these principles to show the equality of two streams?

16. Represent the value $\frac{1}{13}$ by two different labelled transition systems.

17. Prove that the streams represented by these transition systems are equal.

18. What is a predicate? What is an admissible predicate?

19. What is the relevance of admissible predicates for fixed point induction?

20. Let `xs` be a defined list. Prove: `head (reverse xs) = last xs`.

Parts I – IV, Various Chapters

1. John Hughes considers two features of functional programming languages particularly important. Which ones? Why?
2. Stream programming can not really be thought of without an other property or feature of functional programming languages. Which one? Why?
3. What is a monad? What is a sound monad?
4. Testing, verification, correctness by construction. Which of these are *a priori* approaches, which ones *a posterori* approaches for gaining trust in a program's correctness?
5. What is corecursion? What is it good for?
6. What means consistency and completeness of an abstract data type definition?
7. Give a few examples of data types, which are good candidates for being defined as abstract data types.
8. Haskell is well-prepared for defining abstract data types. Do you agree? Why? Or, why not?
9. Explain the essence of the *divide and conquer* algorithm pattern. How does it work? What is required for using it?
10. Give a few examples of problems being a good match for the *divide and conquer* pattern.
11. Give an example of a bottom-up algorithm pattern.
12. The type class hierarchy of Haskell'98 has changed in more recent Haskell versions. Can you name a few examples of changes related to type constructor classes.
13. Use

    ```
    iterate :: (a -> a) -> (a -> a)
    iterate f x = x : iterate f (f x)
    ```

    to define the streams of:

    (a) natural numbers: $0, 1, 2, 3, ...$
    (b) even natural numbers: $0, 2, 4, ...$
14. What is the essence of corecursive definitions? Why are they useful?
15. Are corecursive definitions useful in any functional programming language?
16. If a data type can be made a monoid, it is always obvious how to do that? Right or wrong? Why?
17. Make `(Either a)` an instance of `MonadPlus`.
18. Is your `(Either a)` instance of `MonadPlus` sound? Proof or counter-example.
19. If your `(Either a)` instance of `MonadPlus` is not sound, can you revise the instance definition to make it sound? Prove soundness.
20. What is the `(<$>)` operator? What is its meaning? What does it serve for?
21. What is the identity monad good for? Illustrate your answer (also) by means of an example.
22. Without monads there were no sequencing in Haskell. Right or wrong? Why?
23. Consider Example 12.4.5.2. Compute stepwise the value of:

    ```
    (h length >>= return >>= f cp p) "Fun"
    ```
24. Implement a solver for the *tree label renaming* problem *without* using monadic programming.
25. Compare your *non-monadic implementation* with the monadic one of Chapter 12.5.3. Do you have a preference for one of the two? Why? Or, why not?