LVA 185.A05 Advanced Functional Programming (SS 21)

Central and Control Questions VII

Thursday, 20 May 2021

Topics: Part V, Chapters 17, 18; Part VI, Chapters 19, 20

Pretty Printing, Functional Reactive Programming, Parallel and 'Real World' Programming, Miscellaneous

(No submission, no grading; for self-assessment only)

Part V, Chapter 17 'Pretty Printing'

1. Pretty printing and parsing are closely linked to each other. Why? In what respect?
2. What are important design goals and features of a pretty printer?
3. Explain and illustrate the idea of pretty printing by means of some examples.

– The prettier printer of Philip Wadler is another fine example of

4. monadic programming.
5. abstract data type programming.
6. stream programming.
7. algorithm pattern programming.
8. generate/prune programming.
9. lawful programming.

Right or wrong?

10. The prettier printer of Philip Wadler shall improve on a pretty printer by John Hughes, often considered the 'standard pretty printer of functional programming.' In what respect?

Part V, Chapter 18 'Functional Reactive Programming'

– Complete: Hybrid systems

1. are composed of ... and ... components.
2. can be also considered ... systems.

– Functional reactive programming as presented in Chapter 19 is another fine example of

3. monadic programming.
4. abstract data type programming.
5. stream programming.
6. functorial programming.
7. algorithm pattern programming.
8. arrow programming.
9. generate/prune programming.
10. lawful programming.

Right or wrong?

– What are examples of continuous and discrete

11. physical components

12. logical notions

often occurring in or related to reactive systems?

13. What are examples of continuous and discrete often occurring in reactive systems?
14. What is the current state of Yampa? Is there still support for it? Is there ongoing development? Has it been replaced by research on another project?
15. What can be said about the origins of functional reactive programming?

Part VI, Chapter 19 'Parallel and 'Real World' Functional Programming'

– What kinds of parallelism can be distinguished, are predominant in

1. imperative programming?
2. functional programming?

3. What is an algorithmic skeloton?
4. What shall algorithmic skeletons help to accomplish?
5. What are important examples of algorithmic skeletons?
6. Functional programs shall be rich of implicit parallelism. Why?
7. What are advantages and benefits of implicit parallelism? What are disadvantages and shortcomings of it?
8. What are advantages and benefits of explicit parallelism? What are disadvantages and shortcomings of it?
9. Why is explicit parallelism often considered inadequate for functional programming?
10. Functional programming for 'real world' problems; Haskell programming for real world' problems. Is it still an utopian scheme? Are there signs indicating that the advent of real world functional and Haskell programming is close or already present?

Part VI, Chapter 20 'Conclusions, Perspectives'

1. What are important venues for presenting and publishing scientific advances in programming in general? More specifically, what are such venues for functional programming and specifically Haskell?
2. What are hot research topics in functional programming according to the call for contributions of conferences and journals in the field?
3. What is the programming contest regualarly held as part of the annual ICFP conference series? What are the most important contest rules and regulations? When are this year's deadlines?
4. What textbooks did you refer to frequently? Occasionally?
5. Is there a conference or journal article whose contribution shaped your view and understanding of functional programming?

Parts I – VI, Various Chapters

1. Let

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

   Use `iterate` to define the stream of

   (a) odd natural numbers: $1, 3, 5, ...$
   (b) fourty two's: $42, 42, 42, ...$

2. In adition to the indispensable features of test systems for systematic testing, what would be additional nice-to-have features?

3. Name a few examples of Haskell libraries supporting the testing of Haskell programs.

4. Algebraic data types are often inductively defined so that there is no upper limit on the size of a value of these types. Think e.g. of list, trees, graphs, etc. How can QuickCheck be prevented from generating test date that are unduly large?

5. Define priority queues as an abstract data type in Haskell.

6. The so-called list-of-success technique has been used for several applications throughout the course. Name a few examples and explain the respective purpose of their usage.

7. Who did coin the term of a list of successes?

8. Turing award winner Tony Hoare said that *the success of test is that they test the programmer, not the program.* What did he want to express with his statement?

9. There is a tension between testing and verification and their proponents. How did Tony Hoare view this tension in the, say 1960? How might he view it today? Do you have evidence for your statement?

10. Show that

    ```
    data Maybeplus a = NothingMp | JustMp Int a
    instance Functor Maybeplus where
      fmap g NothingMp = NothingMp
      fmap g (JustMp n x)  = JustMp (increment n) (g x)
    ```

    is an unsound functor instance.

11. Combinator parsing and monadic parsing are conceptually quite similar. Can you provide evidence for this?

12. What is the relevance of the Haskell type classes `Monad` and `MonadPlus` for monadic parsing?

13. The so-called list-of-sucesses technique has been used for several applications in the course of the lecture. Name a few and explain the particular usage there.

14. Huge search spaces can systematically be explored following different strategies. Which strategies have been implemented in the approach for logic programming functionally in Chapter 16?

15. How can a user pick a particular strategy for coping with a problem instance of interest in this approach?

16. Though not an established or widely used term, what could be considered the Münchhausen principle of functional programming? Illustrate your answer by means of an example.

17. Applicatives inherit the `fmap` operation of functors. Right or wrong?

18. List and streams are non-strict in their elements. Right or wrong?

19. What does it mean for a data structure to be strict in its elements?

20. What is functional glue? What is it good for?