Discussion questions for *Why Functional Programming Matters* COMP 150FP

September 6, 2013

Why Functional Programming Matters can be viewed as an *advocacy* paper, arguing that real-world programmers should pay attention to functional programming because it helps solve real-world problems.

Please answer the main questions first; then you may choose which other sets of questions (laziness, modularity, and design) you find most interesting. (Some questions appear on the other side of this sheet.)

Main questions

You can expect to spend a lot of time on some of the questions and hardly any time on others.

1. Most papers answer a question or advance a claim. Just to be sure we're all on the same page, what is Hughes's *technical* question or claim?
2. What did you find eye-opening about Hughes's paper?
3. What parts of Hughes's paper leave you feeling skeptical?
4. What do you like about lazy evaluation? What are the advantages?
5. What do you dislike about lazy evaluation? What are the disadvantages? How much do these disadvantages concern you?
6. What do you like about higher-order functions? What are the advantages?
7. What do you dislike about higher-order functions? What are the disadvantages? How much do these disadvantages concern you?
8. Which is more interesting, lazy evaluation or higher-order functions? Why?
9. This paper was written almost 20 years ago. Has the "real world" been convinced that functional programming is "vitally important?"

Details of laziness

10. What would it be like if function application were strict and only datatype constructors were lazy?
11. Are Hughes's functions `anytrue` and `anyfalse` (page 4) as efficient as they can be? If not, how could they be improved?

Modularity

12. What is modularity? Is there more than one legitimate definition? If so, in what sense does Hughes use the term?

13. Hughes says that lazy evaluation is the "most powerful" modularization tool in the programmer's repertoire (page 7). What evidence or argument does he provide to substantiate that claim? Are you convinced?

14. In the game-tree example, how would you exploit symmetry? (Example: by rotational symmetry, tic-tac-toe boards  and  have the same solutions.) Is your solution modular?

Design

15. Do you prefer dataflow that is explicit through function application (so-called "point-free" programming), explicit through let-binding of named values, or implicit through mutable state?

16. Is shorter always better?

17. Does "more abstract" mean "harder to understand"?

18. Hughes asks for a characterization that will provide a "yardstick of program quality" and indicate what a programmer should strive for. Does he succeed in providing such a characterization? What is it?

19. Can you characterize your own favorite language in a way that shows a programmer what to strive for?