Project Options - CS 457/557 Functional Programming - Spring 2018

The project is intended to give you the opportunity to apply your Haskell programming skills. It should be a small application that does something interesting. Possible topics are described briefly below. We're also happy to consider alternative topics not listed here, if you would like to suggest one. Please email project proposals to Chris (`chak@pdx.edu`). Proposals are due by **5pm, Saturday, May 12**.

Any non-standard topic can be *negotiated*. If you wish to pursue one, your propals is still due by 5pm Saturday May 12th, but feel free to float rough ideas earlier than that, so we can get a feedback loop started earlier. We will negotiate with you after the Saturday deadline; negotiations must be finalized by 5pm Tuesday, May 15. If non-standard project proposals are rejected, students will instead have to do one of the sample projects.

The project will be due at **2pm, Wednesday, June 6**, the last regular class meeting. Students may do an independent project, or work in teams of two, with a corresponding slight increase in expectations.

Learning objectives

Here are some objectives that one might hope to learn in a Haskell class. Remember to leverage your existing skills and knowledge, particularly your software engineering skills (i.e. iterative development, continuous integration etc.) to help you manage your project.

- **Functional Modeling** is how to represent inherently stateful data in a language with immutable data.
- **Functional Data Structures / Algorithms** implementations can be quite different from their imperative analogues.
- **Haskell IO** which includes both "just" reading and writing to the standard IO channel as well as file IO.
- **Learning about and using (appropriate) Haskell libraries / extensions** Haskell has an incredibly large ecosystem. Learning how not to be afraid of it, per se, would be a noble learning objective. However, for this project, learning how to build on top of / utilize a particular library would be a more concrete and useful goal.

The project, in addition to the already stated goal of practicing your Haskell programming, hopes to be a vehicle for you to learn one or more of these objectives. In particular, every project should achieve at least one learning objective. If there are other objectives you can think of, you can suggest those in your proposal.

Here is an example of a project, that should illustrate the scope we're looking for, and how to roadmap your own project:

Example – Super Tic-tac-toe

You can read about Super Tic-tac-toe here (where it's called "ultimate Tic-tac-toe", surely a misnomer).

The goal of the project would be to produce an application allowing 2 users to play Super Tic-tac-toe with each other. The potential learning objectives would include

• Functional Modeling – To capture, and manipulate the game state.
• Libraries – If you implement a GUI.
• Functional Data Strucutues / Algorithms – If you implement an AI.

That is, a Super Tic-tac-toe project could be a vehicle to learn at least one, but possibly more, of these objectives. How deep you go into each objective, by including or eliding features, is up to you.

For example, you could implement a Graphical User Interface (GUI) or Command Line Interface (CLI). The former would further the objective of learning to use a Library (and possibly some design/HCI concerns), while the latter

would allow you to focus on other objectives. P.S. – For CLIs, it is probably ok to require the game be played directly inside GHCi.

Another feature your application might have is a Super Tic-tac-toe playing algorithm (AI). Depending on how you want to do it, this could further the objective of learning more about modeling (efficient representations help algorithms!). With an efficient representation, writing a perfect playing Tic-tac-toe AI is quite simple with pattern matching. Or instead, you could further the objective of learning how to write algorithms in a functional setting, say Minimax (moderate) or Monte Carlo Tree Search (ambitious!). Eliding an AI, would allow you to focus on other objectives.

Despite this agency in deciding the scoping of the project and how much of each objective you'd like to learn, there is a minimum requirement in terms of work. It is not a hard limit and perhaps best illustrated by discussing our concrete Super Tic-tac-toe example.

Too small
· A CLI Super Tic-tac-toe game that allowed 2 users to place marks on a ASCII art board.

Sufficient
· A CLI Super Tic-tac-toe game with an Super Tic-tac-toe AI building off a **perfect** *Tic-tac-toe* AI, allowing users to play against the machine.
· A GUI Super Tic-tac-toe game, allowing 2 users to play against each other. i.e. no AI, but with a GUI

More ambitious (but go for it if you want!)
· A GUI Super-or-not Tic-tac-toe game that allowed either 2 users, or 1 user (against your AI) with a perfect Tic-tac-toe AI and a good, but not perfect Super Tic-tac-toe AI.

Notice how the matrix of choices (of features) allows you to roadmap your project progress; while the CLI - 2 player Tic-tac-toe example is insufficient for a project, it might make a great halfway milestone.

Proposal

Your proposal should include

- A topic, in terms of what you want your code base to do.
- A few concrete features, or different sets of features your project might implement. These should represent possible paths your project can take, not commitments.
- List of learning objectives – Each set of features should be associated with some learning objectives. Of course, you may not know what you'll end up learning, so this can represent a guess.

  (optional) You could include brief remarks about how features relate to the learning objectives. e.g. "Implementing a Tic-tac-toe AI (feature) will help me learn more about either Functional Modeling (objective) or Functional Algorithms (objective)." You could also be more explicit: "Implementing a Tic-tac-toe AI (feature) via the Minimax algorithm will help me learn about Functional Slgorithm (objective)."

  Including these will help us to provide you with course corrections, or give you more tuned advice.
- (optional) A roadmap, in terms of milestones. This can be as simple as splitting up the project into discrete components. e.g. "My project requires a Parser, a Storage component and a Logic component." Or it could describe iterations of your product with each successive iteration having more features, "v0.0 - Prints hardcoded Tic-tac-toe board to screen; v0.1 Takes user input and places marks on Tic-tac-toe board; v0.2 - interactive Tic-tac-toe game . . . "

  This is really more for you to start thinking about how to manage your project.

Each topic description in the samples offers a lot of flexibility as to how much functionality you implement and (hence) how much work you have to do. Projects will be scored on a simple scale from 1 to 5, based primarily on how much functionality is implemented, and secondarily on organization and style of the code. To obtain a 5, your project must

be both sufficiently sized and well-executed. A well-executed but more modestly scoped project can get a 4. Of course, given the short time available, the overall scale of projects should be small: you can do a lot in 500 lines of well- designed Haskell code. Its better to do a few things well than lots of things badly. Feel free to consult us about how much to attempt to do.

Sample topics

Here are sample project ideas and (what I think) their potential learning objectives (are). You should scope and roadmap these yourself in a way similar to what we've done above.

Super Tic-tac-toe or Tetris Game

Potential learning objectives

- Functional Modeling
- Libraries – If you implement a GUI
- Functional Data Structures / Algorithms – If you implement an AI

Feature matrix

- GUI vs CLI
- AI vs Just Interactive
- Extra functionality such as undo / hints (or not)

Topics

- **Super Tic-tac-toe** – as described above
- **Tetris** – This is potentially a little more ambitious as tetris is a real time game and would *probably* require a GUI, in its usual form. However, one could envision building a slowed down CLI version.

Sudoku Solver

These are similar to the games, except that the whole point here is to write the solver / AI. i.e. a Sudoku application that just allowed 1 player to play the game is too small.

Of course, such a project would involve implementing some kind of solving strategy. i.e. your code shouldn't just try to brute force the solution.

Potential learning objectives

- Functional Modeling
- Functional Data Strucutues / Algorithms
- Libraries – If you implement a GUI

Features — There's only 1 real feature, the solver.

Time-aware TODO list manager

Build a toy TODO list manager that can keep track of a TODO list. It should be time aware in that you can attach deadlines or time periods to the TODOs. The application should be able to take input that adds TODOs, and display already added TODOs.

Potential learning objectives

- Functional Modeling – Possibly only a little
- Libraries – Date/Time Libraries
- Functional Data Strucutues / Algorithms – Many possibilities
- Haskell IO – Writing files.

Sufficient features

- TODOs with deadlines. Call these "Deadlines."
- TODOS with time periods. e.g. meeting from 3 - 5pm. Call these "Events."
- Deadline alerts / reminders.
- Warnings when an about-to-be-added Event overlaps in time with another Event.

Ambitious features

- Saving information across sessions. – Requires a data format.
- GUI
- Schedule constraint solver e.g. given a list of TODOs, each with a possible time range, find a way to fit the TODOs into a give schedule, or report which TODOs can be sacrificed to get a schedule. (This is really pretty ambitious.)
- and many more . . .

Language tools / PL projects

These projects are perhaps generally more ambitious, or at least require you to be more familiar with the necessary pre-requisite knowledge.

Sample topics

- **Liquid Haskell**

  Potential learning objectives

  - Extensions – For Liquid Haskell
  - Functional Data Strucutues / Algorithms – Tree / Heap implementation

  Design, implement and verify a ***small*** data structures library. Trees would be reasonable, Heaps a little more ambitious. e.g. verifying a AVL / Red-Black trees library with the following functions/values `empty`, `member`, `add`, `fold`, `toList`.

- **Simple lambda calculus interpreter**

  Potential learning objectives

  - Functional Data Strucutues / Algorithms
  - General / Foundational functional programming – Lambda calculus is the basis for functional programming.
  - Libraries – Possibly for parsing

  Write a Lambda Calculus interpreter, where one could select the evaluation strategy, and could display terms at each step of an evaluation.