Functional Programming and Haskell

CSCI-541/CSCI-641 Section 01
Prof. Matthew Fluet

MWF 11:00am – 11:50am
GOL-2690
Functional Programming and Haskell

Course Overview
Course Description

The goal of this course is to introduce the students to a programming paradigm and an appropriate programming language chosen from those that are currently important or that show high promise of becoming important. A significant portion of the learning curve occurs through programming assignments with exemplary solutions discussed later in class. The instructor will post specifics prior to registration. With the approval of the program coordinator, the course can be taken for credit more than once, provided each instance deals with a different paradigm and language. [CSCI-641: A term project involving independent investigation is also required.]

This course instance is “Functional Programming and Haskell”.
Course Goals

In each offering of the course, students will be introduced to a new programming paradigm and a language that supports it. Both are chosen from technologies that are currently important or that show promise to become important.

A practitioner often has to become proficient in a new language and/or paradigm. The Programming Skills courses are designed to facilitate this learning process.

The course goals allow the faculty to react to current trends in software development while still maintaining a critical perspective.

A student should be allowed to retake this course as long as the topics are different from the other time(s) she/he has taken it. Undergraduate students can “mix and match” undergraduate and graduate offerings as long as the topics are different from the other time(s) she/he has taken them.
Today

- Introductions and course mechanics
- Background/Introduction to Functional Programming and Haskell
Introductions

Who am I?

- Matthew Fluet
- Hooked on functional programming by sophomore PL course
- Studied PL & FP in graduate school and beyond
  - theory, implementation, design
  - Majority of my publications: ICFP, IFL, JFP
  - MLton (a Standard ML compiler)
  - Type- and Control-Flow Analysis (a program analysis refined by types)
  - Manticore (a heterogeneous parallel functional language)
  - Transactional Events (a novel concurrency abstraction)
  - Delta ML (a language for self-adjusting computation)
  - Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

- Matthew Fluet
- Hooked on functional programming by sophomore PL course
- Studied PL & FP in graduate school and beyond
  - theory, implementation, design
  - Majority of my publications: ICFP, IFL, JFP
    - at the *International Conference on Functional Programming*
    - at the *Symp. on Implementation and Application of Functional Languages*
    - in the *Journal of Functional Programming*
- MLton (a Standard ML compiler)
- Type- and Control-Flow Analysis (a program analysis refined by types)
- Manticore (a heterogeneous parallel functional language)
- Transactional Events (a novel concurrency abstraction)
- Delta ML (a language for self-adjusting computation)
- Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

- Matthew Fluet
- Hooked on functional programming by sophomore PL course
- Studied PL & FP in graduate school and beyond
  - theory, implementation, design
  - Majority of my publications: ICFP, IFL, JFP
  - MLton (a Standard ML compiler)
    - a whole-program optimizing Standard ML compiler; actively used in both industry and academia
    - ongoing: whole-program compilation of next-gen language features; automatically managing spatial and temporal locality in a high-level garbage-collected parallel functional PL
  - Type- and Control-Flow Analysis (a program analysis refined by types)
  - Manticore (a heterogeneous parallel functional language)
  - Transactional Events (a novel concurrency abstraction)
  - Delta ML (a language for self-adjusting computation)
  - Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

- Matthew Fluet
- Hooked on functional programming by sophomore PL course
- Studied PL & FP in graduate school and beyond
  - theory, implementation, design
  - Majority of my publications: ICFP, IFL, JFP
  - MLton (a Standard ML compiler)
  - Type- and Control-Flow Analysis (a program analysis refined by types)
    - a program analysis refined by types:
      - control-flow (for first-class functions) + type-flow (for polymorphic types)
    - insight: type- and control-flow are mutually beneficial and decidable
    - ongoing: efficient computation; extend to richer type systems
- Manticore (a heterogeneous parallel functional language)
- Transactional Events (a novel concurrency abstraction)
- Delta ML (a language for self-adjusting computation)
- Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

► Matthew Fluet
► Hooked on functional programming by sophomore PL course
► Studied PL & FP in graduate school and beyond
  ► theory, implementation, design
  ► Majority of my publications: ICFP, IFL, JFP
  ► MLton (a Standard ML compiler)
  ► Type- and Control-Flow Analysis (a program analysis refined by types)
  ► Manticore (a heterogeneous parallel functional language)
    ► an effort to design and implement a parallel functional PL supporting heterogeneous parallelism (parallelism at multiple levels)
    ► results: nested schedulers with composable cancellation; novel implicitly-parallel constructs; lazy-tree splitting; data-only flattening; GC for multicore NUMA; partial-abort STM
    ► ongoing: extend to declarative mutable state
  ► Transactional Events (a novel concurrency abstraction)
  ► Delta ML (a language for self-adjusting computation)
  ► Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

- Matthew Fluet
- Hooked on functional programming by sophomore PL course
- Studied PL & FP in graduate school and beyond
  - theory, implementation, design
  - Majority of my publications: ICFP, IFL, JFP
  - MLton (a Standard ML compiler)
  - Type- and Control-Flow Analysis (a program analysis refined by types)
  - Manticore (a heterogeneous parallel functional language)
  - Transactional Events (a novel concurrency abstraction)
    - a novel concurrency abstraction:
      - first-class synchronous message-passing events + atomic transactions
    - insight: atomicity enhances the expressive power of message-passing
    - ongoing: programming idioms; efficient implementation; fairness
- Delta ML (a language for self-adjusting computation)
- Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

▶ Matthew Fluet

▶ Hooked on functional programming by sophomore PL course

▶ Studied PL & FP in graduate school and beyond
  ▶ theory, implementation, design
  ▶ Majority of my publications: ICFP, IFL, JFP
  ▶ MLton (a Standard ML compiler)
  ▶ Type- and Control-Flow Analysis (a program analysis refined by types)
  ▶ Manticore (a heterogeneous parallel functional language)
  ▶ Transactional Events (a novel concurrency abstraction)
  ▶ Delta ML (a language for self-adjusting computation)
  ▶ Cyclone (a safe dialect of C w/ region-based memory management)

▶ Big Caveat: Haskell is not my “native” functional language
Introductions

Who are you?

- Why are you taking Functional Programming and Haskell?
- What topics in Functional Programming and Haskell are you especially interested in learning about?
- What concerns do you have about the course?

Homework #00

- A brief, but extremely useful survey, on myCourses
- Things like your background, concerns about the course, etc.
- (Also helps me learn names)
Course Administration

Instructor: Matthew Fluet

- E-mail: mtf@cs.rit.edu
- Office: GOL-3555
- Office hours: Mon. 3:30pm – 4:30pm
  Wed. 2:30pm – 4:30pm
  Thr. 9:00am – 10:00am

Website

- http://www.cs.rit.edu/~mtf/teaching/20191/psfp
- http://mycourses.rit.edu
Course Rhythm

- Lectures: 1 – 3 class periods (some with pre-lecture quizzes)
- Recitation Programming Exercise: 1 class period (some with pre-recitation quizzes)
- Homework programming assignment
- Research Paper Summary/Presentation
- Project

**Warning:** I am making some pedagogical “experiments” this semester. Constructive feedback is always welcome.
Assignments, Exams, & Grades

<table>
<thead>
<tr>
<th></th>
<th>CSCI-541</th>
<th>CSCI-641</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attendance &amp; Participation:</td>
<td>5%</td>
<td>4%</td>
</tr>
<tr>
<td>Quizzes &amp; Recitations:</td>
<td>15%</td>
<td>12%</td>
</tr>
<tr>
<td>Homework Assignments (≈ 8):</td>
<td>70%</td>
<td>60%</td>
</tr>
<tr>
<td>Research Paper Summary/Presentation:</td>
<td>10%</td>
<td>8%</td>
</tr>
<tr>
<td>Project:</td>
<td></td>
<td>16%</td>
</tr>
</tbody>
</table>

More details in syllabus and on course website.
Attendance & Participation

Attendance is strongly encouraged
► Will post slides and/or code to website (after lectures)
  ► (but trying to use slides less and board more)
► Take notes
  ► (not everything from a lecture is in the slides and/or texts)

Participation means being an engaged student
► Asking and answering questions
  ► (not simply attending class)
► Let me know if pace is too fast or too slow
► When I enter your grade, will I know who you are?
Quizzes & Recitations

Reading Quizzes
▶ On myCourses (Quizzes)
▶ Available from end of class meeting before lecture/recitation on topic
▶ Short (approx. 15min w/ 60min time limit) and easy, demonstrating that you have read

Recitations
▶ Available from end of the class meeting before recitation
▶ Read and code for at most 1hour before recitation
▶ Complete pre-recitation survey (if any) before recitation
▶ Pair discussion and programming during recitation
▶ Presenting and discussing solutions during recitation
Homework Programming Assignments

Approx. 8 homework assignments

- A significant portion of the learning curve occurs through programming assignments with exemplary solutions discussed later in class.

- programming components in Haskell
  - practice programming the language, not practice searching the libraries
  - submissions that have parse errors or type errors will receive no credit
  - submissions that violate code style guidelines will lose up to 25%

- written components
  - short-answer questions
  - submitted via comments in code

Submitted work must be your own

- Programmed by you alone
- Feel free to discuss homeworks (but do not share solutions)
- Must cite any people or resources consulted
Research Paper Summary/Presentation

Haskell started as and continues to be a “research” programming language. Explore a research paper on Haskell and/or functional programming.

- Pitch a research paper topic as a future course module
- Choose a partner and a recent research paper
- Read and understand the paper
- Formulate a novel assignment based on the topic
- Write a (∼3 page) summary of the paper and describe assignment
- Give a 10min presentation on the paper and assignment

**Partner & Paper Selection:** (Nov. 11 (Mon.)) 10%
**Summary:** (Dec. 3 (Tue.)) 30%
**Presentation:** (Dec. 9 (Mon.) & Dec. 18 (Wed.)) 30% (content) + 30% (clarity)

More details on course website.
Final Project *(CSCL-641 only)*

Write a significant Haskell application or library; 
*must include some design element; must include HUnit and/or QuickCheck testing*

▶ Choose a partner (different from Research Paper Summary/Presentation partner).
▶ Write a project proposal (summary description of overall goal of project; one or more typical “use cases”; sketch of modular components; thoughts on testing; . . ).
▶ Initial design and implementation . . . .
▶ Submit checkpoint code (*README* and *working, but incomplete* code; . . ).
▶ More design and complete implementation . . . .
▶ Submit final project code and give a demo and code walk through.

**Proposal:**  (Oct. 21 (Mon.))  10%
**Checkpoint:**  (Nov. 27 (Wed.))  20%
**Final Project & Demo:**  (Dec. 16 (Mon.) – Dec. 18 (Wed.); 30min slots)  70%

More details on course website.
Academic Integrity & Late Policy

Academic Integrity

- Read course policy (and linked policies)

Late Policy

- Assignments generally due Mo/We/Fr at 11:59pm
- 4 “extension tokens”
  - grants a 24-hour extension on a single homework assignment (excludes research paper summary/presentation and project)
  - must request an extension by e-mail before due date/time
  - up to one extension per assignment
  - (won’t answer questions about assignment after original due date)
Schedule/Topics (tentative)

1. Introduction to Functional Programming and Haskell
2. Programming with (structural) recursion
3. Programming with first-class and higher-order functions
4. Types, polymorphic types, and user-defined datatypes
5. Type classes
6. Purely Functional Data Structures and Algorithms
7. Haskell I/O using monads
8. Monads and Monad Transformers
9. Lazy Evaluation
10. GADTs
11. Application: QuickCheck
12. Application: Combinator Libraries (parsing, pretty-printing, ...)
13. Application: Concurrency
Textbook(s)

Required:
- *Programming in Haskell*, Graham Hutton

Additional:
- *Haskell: The Craft of Functional Programming*, Simon Thompson
- *Thinking Functionally with Haskell*, Richard Bird
- *Get Programming with Haskell*, Will Kurt
- *Learn You a Haskell for Great Good!*, Miran Lipovaca
- *Haskell Programming from First Principles*, Christopher Allen and Julie Moronuki
- *Real World Haskell*, Bryan O’Sullivan, Don Stewart, and John Goerzen
- *Beginning Haskell: A Project-Based Approach*, Alejandro Serrano Mena
- *Practical Haskell: A Real World Guide to Programming*, Alejandro Serrano Mena
- *Parallel and Concurrent Programming in Haskell*, Simon Marlow

Additional resources on course website
Final Metacomment

Acknowledging others is crucial...

This course will draw heavily on:

- similar courses elsewhere:
  - University of Pennsylvania (Stephanie Weirich)
  - University of Pennsylvania (Joachim Breitner, Noam Zilberstein, Richard Eisenberg)
  - RIT (Arthur Nunes-Harwitt)

- texts:
  - Hutton (*PiH*)
  - Thompson (*H:CFP*)
  - O’Sullivan, Stewart, Goerzen (*RWH*)

...