CS 6371: Advanced Programming Languages

Course Information

Title: CS 6371: Advanced Programming Languages
Course Registration Number: 21539
Times: TR 1:00-2:15
Location: ECSS 2.305
Instructor: Dr. Kevin Hamlen (hamlen AT utdallas)
Instructor's Office Hours: TR 2:15-3:15 in ECSS 3.704
Teaching Assistant: TBA
TA's Office Hours: TBA

Course Summary

This course will cover functional and logic programming, concepts of programming language design, and formal reasoning about programs and programming languages. The following are the course learning objectives:

1. Functional Programming (ML/OCaml)
2. Logic programming
3. Small-step and large-step operational semantics
4. Denotational semantics
5. Fixpoints, fixpoint induction
6. Axiomatic semantics
7. Type theory
8. Untyped and typed lambda calculi
9. Partial evaluation, non-determinism

Through taking this course, students will learn the tradeoffs of imperative vs. non-imperative programming languages, issues involved in designing a programming language, the role of formal semantics and type-systems in reasoning about programs and languages, and proof techniques related to programming language design.
The course is open to Ph.D. students and Masters students. Interested undergraduates should see the instructor for permission to take the course.
Prerequisites: Discrete Structures (CS 3305/5333 or equivalent), Algorithm Analysis and Data Structures (CS 3345/5343 or equivalent), Automata Theory (CS 4384/5349 or equivalent). A solid background in all three of these areas will be heavily assumed throughout the course!

To Prepare for the Course...

STUDENTS MUST ATTEND AT LEAST TWO OF THE FIRST THREE CLASSES. IF YOU MISS MORE THAN ONE OF THE FIRST THREE CLASSES, YOU WILL NOT BE PERMITTED TO TAKE THE COURSE. The first three classes will cover functional programming in the OCaml programming language, which will be the basis for most of the rest of the course. To better understand the in-class OCaml demos, you should do the following as preparation:

- Download OCaml from the INRIA website. Feel free to use any version you find easiest to get running. If you are comfortable with Unix, I recommend using one of

the Unix versions. If you prefer Windows, I recommend the Microsoft-based native Win32 precompiled binary. (Using the Cygwin binaries can be difficult unless you are already familiar with Cygwin and how to configure it.) A pre-compiled Win32 installer for OCaml 4 is available at the GitHub repository.

- Once you've successfully installed OCaml, try creating a simple program and compiling it. OCaml programs are plain text files, just like C/Java programs. Using your favorite text-editor, create a file named "fib.ml" containing the code found in section 1.9 of the online OCaml manual. At the Unix or DOS prompt, type one of the following to compile the program:
  - On Unix, type: `ocamlc -o fib fib.ml`
  - At the DOS prompt, type: `ocamlc -o fib.exe fib.ml`

  (Be sure that the `ocamlc.exe` binary is in your path and that the `fib.ml` file you created is in your current directory.) If it compiles successfully, type `fib 10` to get it to print the 10th Fibonacci number.
- Start experimenting with the other examples found on page 3 of the OCaml manual. The other examples on that page use OCaml in interactive mode (where you use OCaml sort of like a calculator instead of a compiler). You can either go ahead and use OCaml in interactive mode or incorporate the examples into your `.ml` file and recompile it to see the results.
- OCaml is an extremely powerful language and has many features that we won't be using in the course. However, in most of the programming assignments you will be free to use any OCaml language features you wish, so the more OCaml you learn, the easier you will find the assignments.

Using OCaml from the UTD Server

If you can't get OCaml to work on your personal machine, you can use OCaml on the UTD CS Department Linux servers. To do so:

- ssh to `cs1.utdallas.edu`
- at the Unix prompt, type `ocaml` to enter interactive mode, or type `ocamlc` to use the compiler
- to exit interactive mode, at the OCaml prompt type: `exit 0;;`

Using Prolog from the UTD Server

You can install your own local version of SWI Prolog or you can access the version installed on the UTD linux servers as follows:

- ssh to `cs1.utdallas.edu`
- at the Unix prompt, type `pl`
- to exit Prolog, type control-C then e

Grading

Homework (25%): Homeworks will be assigned approximately once per 1.5 weeks, and will consist of a mix of programming assignments and written assignments. Programming assignments will be done in OCaml, Prolog, or possibly Coq. Written assignments will typically involve discrete math proofs. Homeworks must be turned in at

the start of class (i.e., by 1:05pm) on the due date. **No late homeworks will be accepted.**

Quizzes (15%): On indicated assignment due dates (see the course schedule below), students will solve one or two problems individually at the start of class as a quiz. The quiz problems are essentially extra homework problems solved individually in class without the help of internet solution sets or collaboration with other students. The quizzes will be closed-book and closed-notes.

Midterm (25%): There will be an in-class midterm exam in class on Thursday, March 5th. The exam will cover functional programming, operational semantics, denotational semantics, and fixpoints.

Final (35%): The final exam for the course is scheduled for TBA. The exam will be cumulative, covering all material in the course. Students will have 2 hours and 45 minutes to complete it.

Homework Policy

Students may work individually or together with other students presently enrolled in the class to complete the assignments, but they must CITE ALL COLLABORATORS AND ANY OTHER SOURCES OF MATERIAL that they consulted, even if those sources weren't copied word-for-word. Copying or paraphrasing someone else's work without citing it is plagiarism, and may result in severe penalties such as an immediate failing grade for the course and/or expulsion from the computer science program. Therefore, please cite all sources!

Students may NOT consult solution sets from previous semesters of this course, or collaborate with students who have such solutions. These sources are off-limits because such "collaborations" tend to involve simply copying someone else's answer to a similar homework problem, which does not prepare you for the quizzes and exams.

Texts

The course has no required textbook, but we will make use of several online references:

- OCaml References:
  - The OCaml Manual by Xavier Leroy
  - Developing Applications With Objective Caml by Emmanuel Chailloux, Pascal Manoury, and Bruno Pagano; published by O'Reilly France and now available for online reading
  - 99 Practice Problems (with solutions) in OCaml
- Coq References:
  - The Coq Homepage
  - Software Foundations, by B. C. Pierce, C. Casinghino, M. Greenberg, C. Hri□cu, V. Sjöberg, and B. Yorgey, University of Pennsylvania, 2012.
- Operational & Denotational Semantics References:
  - *The Formal Semantics of Programming Languages: An Introduction* by Glynn Winskel (placed on reserve at the UTD library)
  - Denotational Semantics: A Methodology for Language Development by David Schmidt, out of print but available online
- Type Theory References:
  - *Types and Programming Languages* by Benjamin C. Pierce

- Logic Programming Resources:
  - Logic, Programming and Prolog (2nd ed.) by Ulf Nilsson and Jan Maluszynski
  - SWI Prolog downloads and manuals

Tentative Course Schedule

<table>
<tr><th>Date</th><th>Topic</th><th colspan="2">Assignments</th></tr>
<tr><td colspan="2">Functional Programming</td><td></td><td></td></tr>
<tr><td>Lecture 1:
Tue 1/13</td><td>Course Introduction: Functional vs. Imperative programming, type-safe languages, intro to OCaml</td><td rowspan="4">Assignment 1 due 1/22 (OCaml Intro)</td><td></td></tr>
<tr><td>Lecture 2:
Thu 1/15</td><td>OCaml: Parametric polymorphism</td><td rowspan="2">Assignment 2 due 1/29 (SIMPL Interpreter)</td></tr>
<tr><td>Lecture 3:
Tue 1/20</td><td>OCaml: List folding, tail recursion, exception-handling</td></tr>
<tr><td colspan="2">Operational Semantics</td></tr>
<tr><td>Lecture 4:
Thu 1/22</td><td>Large-step Semantics: Intro</td><td></td></tr>
<tr><td>Lecture 5:
Tue 1/27</td><td>Large-step Semantics: Proof techniques</td><td></td><td rowspan="2"></td></tr>
<tr><td>Lecture 6:
Thu 1/29</td><td>Small-step Semantics</td><td rowspan="5">Assignment 3 due 2/10 (Operational Semantics)</td></tr>
<tr><td colspan="2">Denotational Semantics</td><td></td></tr>
<tr><td>Lecture 7:
Tue 2/3</td><td>Denotational Semantics: Semantic domains and valuation functions</td><td></td></tr>
<tr><td>Lecture 8:
Thu 2/5</td><td>Denotational Semantics: Fixed points</td><td></td></tr>
<tr><td>Lecture 9:
Tue 2/10</td><td>Fixed-point Induction</td><td rowspan="5">Assignment 4 due 2/19 (Fixpoints)</td></tr>
<tr><td>Lecture 10:
Thu 2/12</td><td>Semantic Equivalence</td><td></td></tr>
<tr><td>Lecture 11:
Tue 2/17</td><td>Formal methods: Program-proof co-development</td><td></td></tr>
<tr><td colspan="2">Type Theory</td><td></td></tr>
<tr><td>Lecture 12:
Thu 2/19</td><td>Type Theory: Introduction</td><td rowspan="3">Assignment 5 due 3/3 (SIMPL Type-checker)</td></tr>
<tr><td>Lecture 13:
Tue 2/24</td><td>Type Theory: Type-soundness, Progress and Preservation</td><td></td></tr>
<tr><td>Lecture 14:</td><td>Type Theoy: Type-based</td><td></td></tr>
</table>

Thu 2/26	Information Flow Security		
Lecture 15: Tue 3/3	**Midterm Review**		
Midterm: Thu 3/5	**Midterm Exam**		
Untyped Lambda Calculus			
Lecture 16: Tue 3/10	**Untyped Lambda Calculus:** Introduction	Assignment 6 due 3/26 (Lambda calculus)	
Lecture 17: Thu 3/12	**Untyped Lambda Calculus:** Encodings and reductions		
No Class: Tue 3/17	**No Class:** Spring break		
No Class: Thu 3/19	**No Class:** Spring break		
Typed Lambda Calculus			
Lecture 18: Tue 3/24	**Simply-typed Lambda Calculus**		
Lecture 19: Thu 3/26	**System F:** Type-inhabitation, Curry-Howard Isomorphism		Assignment 7 due 4/9 (Functional SIMPL)
Lecture 20: Tue 3/31	**Summary/Comparison of Modern Language Features:** Weak vs. strong typing, type-safety, function evaluation strategies		
Lecture 21: Thu 4/2	**Summary/Comparison of Modern Language Features:** Hindley-Milner type-inference, type polymorphism		
Logic Programming			
Lecture 22: Tue 4/7	**Logic Programming:** Part I		
Lecture 23: Thu 4/9	**Logic Programming:** Part II	Assignment 8 due 4/21 (Prolog)	
Lecture 24: Tue 4/14	**Logic Programming:** Part III		
Formal Verification			
Lecture 25: Thu 4/16	**Axiomatic Semantics:** Hoare Logic		
Lecture 26:	**Axiomatic Semantics:** Loop		Assignment 9

Tue 4/21	invariants		due 4/30 (Hoare Logic)
Lecture 27: Thu 4/23	**Axiomatic Semantics:** Weakest precondition, strongest postcondition		
Lecture 28: Tue 4/28	**Final Review**		
Lecture 29: Thu 4/30	**Final Review**		