
CS 421 – Spring 2007

Lecture Notes Set 1:

Introduction to Course

Elsa L. Gunter¹

Corresponding to Slides: 01-intro
Made available: January 17, 2007
Revision 1.0

1 Change Log

1.0 Initial Release.

2 Course Administration

2.1 Course Staff

The instructor for CS 421 Spring 2007 is Elsa L. Gunter. Contact information for her is as follows:

- Office: 2112 SC
- Office hours: Wednesdays 10:00am - 11:15am. Also by appointment
- Email: egunter@cs.uiuc.edu

The TAs for the course this semester are: T. Baris Aktemur, Munawar Hafiz, and Chris Osborn. They will use a common office, 0207 SC. Their contact information is as follows:

- T. Baris Aktemur
 - Email: aktemur@uiuc.edu
 - Hours: Mon 10:00am - 11:00am & Thurs 1:00pm - 2:00pm
- Munawar Hafiz
 - Email: mhafiz@uiuc.edu
 - Hours: Tues 3:00pm - 4:00pm & Fri 2:30pm - 3:30pm
- Chris Osborn
 - Email: cosborn3@uiuc.edu
 - Hours: Tues 11:00am - 12:00am & Wed 3:30pm - 4:30pm

¹© 2007, Share and Enjoy

2.2 Course Resources

An important resource for the course is the course website. The main web page for the course is found at: <http://www.cs.uiuc.edu>. This page contains a summary of news items for the course, and is frequently updated. You should check this web page on a daily basis. This web page also contains links to several other important web pages:

- **policy** - describes rules governing course
- **lectures** - contains the course syllabus and links to slides for the lectures; it also contains links the lectures from the two previous semesters
- **mps** - provides information about homework, including links to download the assignments and their accompanying files.
- **Exams** - dates and information for preparing for exams
- **Unit Projects** - some information for 4 credit students, with some suggestions as to projects, and some possible papers to read and present.
- **Resources** - tools and helpful info
- **FAQ** – information on a range of topics that sometimes cause students difficulties.

There is no required textbook for this course. Some books of relevance to this course include:

- *Essentials of Programming Languages (2nd Edition)* by Daniel P. Friedman, Mitchell Wand and Christopher T. Haynes, MIT Press 2001.
- *Introduction to the Objective Caml Programming Language* by Jason Hickey
- *Compilers: Principles, Techniques, and Tools*, (also known as "The Dragon Book"); by Aho, Sethi, and Ullman. Published by Addison-Wesley. ISBN: 0-201-10088-6.
- *Advanced Programming Language Design*, by Raphael A. Finkel. Addison Wesley Publishing Company, 1996.
- *Programming Language Pragmatics*, by Michael L. Scott. Morgan Kaufman Publishers, 2000.
- *Concepts, Techniques, and Models of Computer Programming* by Peter Van Roy and Seif Haridi, MIT Press, 2004.

The first text, *Essentials of Programming Languages (2nd Edition)* covers most of the topics in this course, but uses a different programming language (Scheme). The second item is an on-line text from Cal Tech found at <http://www.cs.caltech.edu/cs134/cs134b/book.pdf>, and is a recommended resource for learning about OCaml.

2.3 Coursework and Grades

Your course grade will be determined by your performance in three components: weekly assignments (MPs and HWs), two midterms, and a final exam. The combined weekly assignments are worth 30% of the course grade. each midterm is worth 15%, and the final is worth 40%. These percentages are approximate, with the possibility of the exams weighing more. In particular, if there is a large discrepancy between the homework and the exams, with the the homeworks being much better, then the homework may be substantially deemphasized.

The first midterm for the course is on Friday, February 9, 2007, and the second is on Friday March 16, 2007. The final is on Saturday May 5 at 8:00am - 11:00am. Do not miss these exam dates.

There will be about 9 MPs (in Ocaml) and 3 written assignments (HWs). The MPs are to be submitted electronically using the `handin` program on on the EWS computers. You must have an account on the EWS system to submit your assignments. This is covered more fully on the **policy** web page of the course website.

The following are guidelines for the degree of collaboration allowed on homeworks for this course. You may discuss homeworks and their solutions with others. Discussing the material in this course, the meaning statements of homework problems, and approaches and solution outlines with fellow students is a good way to gain a deeper understanding of the material in this course. However, you may not leave the discussion with a written solution. You must finally do homework solution alone. In particular, you must write your own solution alone without looking at another's solution. You may look at examples from class and other similar examples.

3 Course Overview

The aims of this course are

- to broaden and strengthen your knowledge of programming paradigms, with a focus on
 - higher-order functional programming,
 - modeling problems with recursive algebraic data types, and recursive programming over them
 - tail recursion and continuation passing style
- phases of an interpreter
 - lexing and parsing
 - type checking
 - evaluation
 - transformations and optimizations
- semantics (formal meaning) of programming languages, including
 - discussion of order of evaluation of expressions
 - the lambda calculus as a formal model of computation
 - and structured operational semantics as a method of describing how to evaluate a program in a given programming language, and to what it evaluates.

To help solidify your knowledge of each of these topics, over the course of this semester, we will build an interpreter for a simple functional programming language.

4 Introduction to OCaml

The compiler for OCaml is on the EWS-linux machines at `/usr/local/bin/ocaml`. It is also possible to download and install ocaml onto PCs, Macs, and Linux machines. The main website for OCaml is <http://caml.inria.fr/index>. To install OCaml on your computer see, visit the website <http://caml.inria.fr/ocaml/release.en.html>.

The following are some references for OCaml. That are not References for CAML Supplemental texts (not required):

- The Objective Caml system release 3.09, by Xavier Leroy, is an on-line manual
- Introduction to the Objective Caml Programming Language, by Jason Hickey, mentioned earlier
- Developing Applications With Objective Caml, by Emmanuel Chailloux, Pascal Manoury, and Bruno Pagano, on O'Reilly

All of these may be found through links on the course `resources` web page.

4.1 A Brief Overview of OCaml

CAML is European descendant of original ML, sometimes called classic ML, developed by Turing Award winner Robin Milner. The version of ML that has developed in Britain and America is Standard ML, or SML for short. The two languages, SML and OCaml are very similar. The ML family of programming languages was designed for implementing interactive theorem provers. The name ML in fact stands for *Meta-Language*, a language used to express languages. Classic ML was developed to be the language of implementation and interaction for the theorem prover LCF (standing for Logic of Computable Functions). SML is the meta-language for the theorem provers HOL and Isabelle, and Caml is the meta-language of HOL-Lite and Coq.

OCaml is the object-oriented extension of Caml. OCaml is the only (stable) version of the ML family that fully, natively, supports objected-oriented programming.

Despite its obscure original application area, OCaml is a full general-purpose programming language. Members of the ML family all use pattern matching as a main method of computation, and they do it very well. Thus they are very appropriate languages for doing symbolic computation, and are particularly efficient for programming tasks involving languages (eg parsing, compilers, user interfaces). For appropriate applications OCaml is fast. The winners of the 1999 and 2000 ICFP Programming Contests used OCaml. Also, various applications such a extensions to the SPIN model checker developed at AT&T Bell Labs were originally written in SML, and when they were reimplemented in C, they were 3 times as slow.

We introduce you to OCaml here because we wish to broaden your knowledge of programming paradigms. Many of the features of OCaml (or SML) are not clearly in languages you have already learned, such as C++ or Java. For those of you wishing to go on to do graduate work, or wishing to interact with research groups in industry, OCaml (or SML) is an assumed basis for much research in programming language research. It is also used a tool for developing other research tools. For example, OCaml was used at Microsoft for writing SLAM, a formal methods tool for C programs.

Features of OCaml, and of the ML family in general, include:

- It is higher order applicative language. This means the functions are treated as first class entities. They can be passed as arguments, stored in data structures, and returned as arguments.
- It uses call-by-value parameter passing, also known as eager evaluation. This means that the argument to a procedure is evaluated to a value before it is passed to the body of the procedure. Thus, any side-effects of evaluating the argument will occur before the procedure begins.
- It uses as modern syntax; expressions are written in a style that resembles mathematical expressions, and declarations are written in a style that resemble mathematical definitions.
- It uses parametric polymorphism, also known as structural polymorphism; functions may be written and compiled once, that can then be applied to a range of different types.
- It uses automatic garbage collection, so you never allocate or deallocate objects.
- It has rich support for user-defined algebraic data types. We will use these extensively through this course. They are the basis of how we will represent abstract syntax for languages.