UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

CS 294-8 **Prof. R. Fateman**
Fall 2006

CS 294-8: Programming Technology for Artificial Intelligence and Symbol Manipulation

Personnel

Instructor: Richard J. Fateman, 789 Soda (fateman@cs.berkeley.edu)
Class Home Page: `http://www.cs.berkeley.edu/~fateman/294-8`.

About the course

CS294-8 has three intertwined components. The first strand is the study of a slice of symbolic artificial intelligence (AI) as reflected in the ideas and programs of the past and present. In this sense it is comparable to a literature course where one learns to appreciate and criticize others' work[1].

The second strand is "advanced programming." Reading, modifying, and creating new programs is not in itself an appropriate task for a CS graduate class (at least here, and not for credit). But there is more to what we are doing here: these tasks involves understanding advanced ideas in organizing computation, solving problems, or designing abstractions. In some cases this is a revisiting of some of the themes of CS61a which you are now mature enough to appreciate.

The third strand is high-level programming language design and implementation. Common Lisp is the implementation language for each of the case studies in the text. A Common-Lisp-centric view of programming languages can be quite helpful in AI or computer science generally: with some experience, it is usually easier to prototype a specialized programming language with a base of CL than by using other techniques, in spite of C-centric tools like LEX/FLEX or YACC/BISON (such tools are available in Lisp too).

In fact, one very effective way of studying another language is to see how one can implement it interpretively, or write a compiler for it, in a known language. Again, we revisit, though briefly, a part of the meta-circular evaluator you saw if you took CS61a!

Your assignments will consist primarily of reading, programming and analysis tasks. An important component of the course is a project. And you will be expected to present material and lead discussion in class. See below.

Prerequisites

Students who have taken this course in the past (when it was numbered CS283) have had a variety of backgrounds. Typical students have included:

[1] Although we will not look at the actual original programs. Instead, we will see them them in cleaned-up, condensed form. Benefiting from 20/20 hindsight, we can usually improve on the older programs.

- a CS senior who wanted more "hands-on" understanding of AI programming.
- a graduate student in another area (EECS CAD) who wanted to see in depth, case studies of AI projects that "work" and that might have relevance in his design tools.
- a Cognitive Science grad student.
- a CS graduate student filling in a minor program in AI.
- a CS graduate student considering an AI major, but not far along in coursework or research.

This course is **not** aimed at an advanced AI graduate student who is well along on thesis work; typically we will sacrifice depth of treatment of special topics to breadth, and thus it may seem somewhat superficial to an expert in those areas.

CS 164 (Programming Languages and Compilers) is the only official prerequisite, but can be waived. Informally, CS 188 (Introduction to Artificial Intelligence), and some experience in programming in Lisp will be helpful, but not required.

If you have never used Lisp (or Scheme) seriously or taken an AI course, but you are highly motivated, you are welcome to try this course anyway, but be prepared to spend some time picking up this material.

You absolutely must have taken a course in data structures, and you are assumed to be skilled in using some operating system enough to get your work done. We are most productive using Lisp and a suitably-aware `emacs` editor. If you have not used emacs, it is not too late to learn it. The operating system is probably irrelevant in this case. If you decline to use emacs, there are IDEs for Windows or Macintosh. All course software should run on any home computer with free open-source software, or the free demo-level software from lisp vendors. (Allegro Common Lisp version 8.0, which is available as a full system on our computers, can be downloaded from franz.com in a trial "express" version, which should be perfectly adequate for all coursework, and most projects. There are other possibilities, for which you can visit `http://wiki.alu.org/Implementation`. You have more choices if you run Linux on Intel, but there are Windows and Mac systems too.)

While it is possible to write programs in other languages for this course, languages that you are most familiar with– Java, C++, C#, etc., we do not recommend it. Python or perhaps functional programming languages such as CAML are more plausible; in fact students have (in particular) written projects in OCAML. Part of the enjoyment of the text by Norvig comes from seeing short yet complete self-contained programs for tasks that earned someone a PhD, re-written in Lisp. So you will have to read Lisp nevertheless, and might enjoy writing in it too.

Scheduling

The class is currently scheduled to meet in 405 Soda 2:30–4:00, MW.

Grades

Your grade in this course will depend on homework assignments (problems, programs), class participation and (mostly) a project. There will not be a written final. **In conformance with department policy, we ask that all students who regularly attend (even as "auditors") officially register. Undergraduate auditors should register for a P/NP, grad students for S/U grading option. The understanding is that regular attendance and participation in class will suffice for receiving a Pass. Completion of assignments or exams will not be required. Registering all attendees is the only way the department (and the instructor) gets teaching credit. This policy appears to be in line with the policy of other departments.**

In the past, projects have generally been programmed applications of techniques to novel areas, extensions of tools, or revisions of programs to make them more efficient by means of improved algorithms, data structures, etc. It is possible to construct projects that might be more reading/analysis/benchmarking, though this has been rare in the past.

Texts

REQUIRED
Peter Norvig: *Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp*, Morgan Kaufmann Publ. 1992.
RECOMMENDED
Guy L. Steele, Jr: *Common Lisp the Language* 2nd edition Digital Press, 1990. (We refer to this as CLtL2. The first edition, CLtL1 may be useful, and even preferable for its lighter weight. Unfortunately it may not include some material that we will use.) This is entirely available on-line, and may be more useful if downloaded and available for searching.

Other books on Lisp may be helpful. The best and most recent introductory/ intermediate book is by Paul Graham *ANSI Common Lisp* (Prentice Hall). Also see the free online `http://paulgraham.com/onlisp.html` for a well-written and insightful book on advanced programming in Lisp. For comments on some of the other older possibilities, see the discussion in Norvig's text starting on page xiii.

There are a number of extremely extensive online documentation projects, including the ACL documentation and the Common Lisp Hyperspec (find a nearby copy, maybe download it), as well as tutorials.

Other notes

While we will cover a number of topics from Norvig's book, as is typical for graduate courses, we will not necessarily adhere to the same syllabus as last time this course was offered. Student interests will play some role in topic coverage, and I would like to inject some newer material, perhaps on "automatic" program generation and proofs, handwriting recognition and speech recognition from my own research. I will try not to overlap (much) with courses on automated theorem proving or machine learning, but projects might.

Office Hours

My office hours (tentatively) are just after class 4-5pm MW, but also I am often in 789 Soda Hall, and I am usually willing to see you if I am around, and I usually respond to email quickly. If your email gets caught by my spam filter, I might not see it; sending from an account in the berkeley.edu domain is probably a good idea.

Computer Accounts

We expect that all your programming assignments will be written in Common Lisp, although you might be able to convince us that C, C++, Java, Python, ML, etc. could make sense. Arguably one of these may make better sense for *a project*.

Registered students who do not already have accounts on shared machines running Lisp can get accounts almost immediately. (See me).

The Common Lisp I will use is *Allegro Common Lisp 8.0* from Franz Inc. This is a commercial product with good user interfaces, packages that provide window support, graphical profiling tools, condition handler, CLOS, etc. The core language implemented is CLtL2, plus some items from

the X3J13 standards committee for Common Lisp that did not quite get into CLtL2, and a bunch of other features to support internet activity, data mining, interfaces to libraries, etc. You may need to add a few lines to your `.emacs` file to make everything work. Some students use emacs with SLIME, or the Franz-supplied emacs macros. There are many additional features that you can add to the emacs-Lisp interface.

About Sharing

This is a course where a little bit of educated help from the instructor or a fellow student can help you finish an assignment much MUCH faster. If you receive such help, you should give credit to persons who helped. You can say "I thank J. Smith for clearing up the issue of scope in this function definition" or even "J. Smith wrote the following clever function, and now that I have studied and understood it thoroughly, I can't see any way of improving it. I am using it with her permission." or "I found a program with similar functionality in the Proceedings of the Vulcan AI Academy, and have adapted it to solve this homework problem." Learning to properly attribute and acknowledge others' contributions is important; doing so will not affect your grade adversely. Conversely, it is not acceptable to portray others' work as your own.

Details on the Course Material

My intention is to initially follow the material in Norvig's text approximately in the order it is presented there, starting with Chapter 2, with some omissions and occasional supplementation. This class will be sufficiently small that student interests can and will significantly influence the coverage of topics. I am asking for an expression of interest in different topics in your questionnaire –to be turned in today. (You are not bound by your choice of interest at this time.)

As we proceed through some of the topics, students, (singly or in pairs), will meet with me to prepare and lead class discussions of some of these chapters. Some of Norvig's later material on natural language topics are rather more detailed than appropriate for the survey nature of this particular course. Even if you are not presenting material on a particular chapter or section, we expect you to read the material carefully in preparation for class discussion, and probe the speakers (or me...).

Assignment 1: due Sept 11 in class

For the programming assignments, you should provide brief but suitable documentation, stating any assumptions or limitations you make on the input. (Naturally you must have some taste in not assuming away a significant part of the problem. Ask if you have any questions about what you can assume.) Normally you should include a test run or two to exercise the programs demonstrating they work at least sometimes. Unless otherwise stated, answers to questions should be printed out on paper and handed to me. *If you get stuck with Lisp, remember that you are welcome to ask for help!*

1. Read chapters 1, 2, 3 in Norvig. If you are unable to get the text, at least review any of a number of books mentioned in class, some of which are free and on-line. Paul Graham's "On Lisp" is a possibility. (http://paulgraham.com/onlisp.html).

2. Try all of the exercises in Norvig, chapter 3. Most have the answers provided. You may need to refer to other Common Lisp reference material for some details in understanding the programs. No need to hand them in, but you should understand the discussion of the answer.

3. Here is a function that has an obvious description.

```
(defun LengthAtLeast(x n)
"return T if x is a lisp list of length n or more else return NIL"
 (> (length x) n))
```

Since `x` and `n` could be of any type or size, many kinds of error and inefficiencies could crop up. Try to make this function as bullet-proof and efficient (in time) as possible. Add to the documentation as necessary. Consider `check-type` and `assert`.

4. Using idioms from other languages or unnecessarily obscure constructions are common defects in programs written in Lisp or for that matter, any other programming language. A Fortran-trained programmer with a taste for somewhat exotic Lisp constructions might come up with this strained solution to a certain problem. Although it works, it is ugly. About the only way to understand it is to test it.

```
(defun lon (n) ;; n is an integer.
 (do ((i 1 (1+ i))
      (ans nil `(,@ans ,i))) ;; that's a back-quote
     ((> i n) ans)))
```

What's going on here? Rewrite it so that it is easier to understand. It might even get faster and smaller. (Optional: You can see if it is faster by timing it with `time`, make it faster with `compile` after playing with declarations, and see if it is smaller by `disassemble`.)

6. Write a function `permp` of one argument, l which is a list of n numbers. Return non-nil iff l is a permutation of the integers from 1 to n. For example, `(3 1 2)` is a permutation of length 3, but `(4 3 1)` is not a permutation at all. You may want to separate the check that the numbers are all integers from the check that the list l is a permutation.

Is your program "linear time" in n?

You may notice that there is a function `permutations` defined by Norvig on page 150. We mention this only so we can point out that it is not needed for a good solution.

SURVEY

Please fill out this page and return it to me by the end of class today:

Name:_____ e-mail:_____
Undergrad (year):_____ Graduate student(year):_____
Major Field:_____

Grading option: Letter Grade___ / (P/NP):_____

Prerequisites (AI course? Compiler course?)

Programming (List your ‘‘native’’ programming language(s) in order of familiarity)

What do you hope to get out of this course? (answer on reverse)

What topics below (with chapters in Norvig, indicated) might be of interest to you? If your level of interest is high, would you wish to lead discussion on that topic?
(No Answer = Don't Know)

(estimated) level of interest:	low	medium	high	might lead discussion
Advanced Lisp programming (chapters 2, 3, 24, 25)				
Efficiency (9, 10)				
Scheme implementation (22)				
Compiling lisp (23)				
5 topics in Classic AI --				
GPS: General Problem Solver (4)				
Eliza (5)				
Building software tools (6)				
STUDENT (7)				
Symbolic Math (8, 15)				
Other advanced topics				
Logic Programming (11, 12)				
Object-oriented programming (13)				
Knowledge representation (14)				
Expert Systems (16)				


Line-labeling (17)
Search/Othello (18)
Natural Language (19, 20, 21)
OCR (not in Norvig)
(Other.. you specify!)