# Paradigms of Aritificial Intelligence Programming: Case Studies in Common Lisp by Peter Norvig

## A Book Report

Earl Spillar[1]

[1]Representing only himself

ABQ Lisp/Scheme

- Learning Lisp: specifically reading through a bunch of good code to get the hang of it
- Learning some of of the classical techniques of AI programming

# Writing for AI and LISP programming

His goal is evidently to teach people what I want to learn. The book is organized into 5 parts serving the audiences nicely:

- Part 1: Introduction to Lisp, for the Patzer- but still useful
- Part 2: Early AI programs: for Lispers and people interested in AI
- Part 3: Tools and Techniques: (efficiency, logic programmig, CLOS, etc:: LISPers
- Part 4: Advanced AI Programmers

# First Three Chapters

- The first chapter give a pretty good introduction to common lisp.
- The second chapter's example is a rule driven program which creates simple English sentences; the emphasis is on creating a rule based "programable" solution.
- The third chapter gives a rather brief overview of part of common lisp. It also gives a few guidlines on style and de-bugging.

The second part presents the re-construction of several seminal AI programs in common lisp. As each of the programs is implemented, a core set of libraries which will be used through the rest of the book is developed.

- “The General Problem Solver, developed in 1957 by Alan Newell and Herbert Simon, em- bodied a grandiose vision: a single computer program that could solve any problem, given a suitable description of the problem.”
- The claim: “It is not my aim to surprise or shock you. . . . But the simplest way I can summarize is to say that there are now in the world machines that think, that learn and create. Moreover, their ability to do these things is going to increase rapidly until-in a visible future-the range of problems they can handle will be coextensive with the range to which the human mind has been applied. ”

GPS was a goal seeking program.

- States for the system are specified
- Goals are specified
- a list of actions- things which transform the system, and can only perform based on preconditions, is specified

The program implements a general set of algorithms for searching the space of actions and solving the problem. It is amusing that the whole program is a couple of pages! What I learned was the use of symbols in the code- I kept wanting to make strings!

- Eliza is the familiar “psychologist”
- First, a general pattern matcher is implemented
- Second, the pattern matcher is augmented by a set of transformation rules
- Once again, the program is quite compact!

Several tools are written; the main one is a general search tool. The problem is factored into four components:

- Start
- Goals
- Successors
- Strategy

Search strategies such as depth first, breadth first, and beam are discussed.

- Based on Bobrow's 1964 thesis.
- "If the number of customers Tom gets is twice the square of 20% of the number of advertise- ments he runs, and the number of advertisements is 45, then what is the number of customers Tom gets?"
- That wasn't the problem: the problem is write a program that parses that sentence and SOLVES that word problem!

- First, use the parser we already have
- second, translate phrases into equations
- write a transformer function that solves these equations

STUDENT was reprinted by Minsky, and has been revisited frequently

Goal is to write a small subset of a code like Macsyma. Some interesting tidbits:

- Lisp grew out of a program to develop solutions
- SAINT (1963) solved integration as would an undergraduate- pattern matching and goal seeking
- Risch (1970) developed algorithms which eliminated the need for search
- One of the knottier problems is the context dependent meaning of “Simplify”

Algebra up through basic integration is covered. The implementation is basically a repeatedly used set of pattern recognition and substitution rules.

I believe this is a rather unique section in its depth and breadth

- Caching of results: memoization: of course a general macro!
- Compilation: of course using lisp's built in compiler to compile patterns
- Delaying computation (ala scheme)
- Instrumenting code to find what needs optimizing (premature optimization satisfies no one)

- Using the disassembler to see what effects your changes have (Interesting, he mostly counts lines)
- Using declarations: nice; many treatments are too terse
- Avoiding generic functions and complex argument lists
- Avoiding unnecessary consing
- Using the right data structures

Describing and implementing Prolog: I skipped this on first reading

Unlike the “fashionable” disdain for OO programming chic in current lisp circles, Norvig in this book regards it as a useful tool. CLOS was still not totally approved at the time

- A rather nice introduction to CLOS
- A re-implementation of searching tools.

Sadly, I'm working through this chapter-

- 15: Symbolic Mathematics with Canonical Forms
- 16: Expert Systems
- 17: Line-Diagram Labeling by Constraint Satisfaction
- 18: Search and the Game of Othello
- 19: Introduction to Natural Language
- 20: Unification Grammers
- 21: A Grammer of English

- 22: Scheme: An Uncommon Lisp
- 23: Compiling Lisp
- 24: ANSI Common Lisp (loop macro, etc.)
- 25: Troubleshooting

- nicely readable book
- gives a clear impression of this history of “traditional” “MIT” AI programming
- excellent code to read
- excellent demonstration of iterative development
- somewhat dated: 1992: e.g. no ASDF
- More focused on AI than Practical Common Lisp
- Amazon: 6 5 star reviews, 1 3 star