Preface

This book derives from a course that I teach at the University of Toronto entitled Computers and Thought. The syllabus states the following:

> The goal [of the course] is to study one idea in detail, the idea that ordinary thinking as performed by people might be understood as a form of computation. We explore the connection between thinking and computing by examining what it takes to program a computer to perform certain tasks that seem to require thought. A secondary goal is to learn a certain type of computer programming in a language called Prolog.

The course is intended for first-year undergraduate students who are interested in this idea but who have no technical specialization or background other than high school mathematics. This book is intended to serve as the text for such a course.

Background

The faculty of Arts and Science at the University of Toronto decided a few years ago that it would be a good idea to offer seminar-style courses to first-year undergraduates. This would have a number of desirable effects. First, students would be exposed to research ideas very early in their academic careers and perhaps see how a research-intensive university is different from one that concentrates mainly on teaching. It would also allow first-year students to come into contact with senior faculty engaged in research, whom they might not otherwise encounter until they took upper-level (or even graduate-level) courses.

The faculty decided that each department had to offer at least one such seminar per year, guaranteeing that new students would have a wide assortment to choose from. Instructors were free to decide on the specific topic they would cover; they were asked only to provide a short abstract. Incoming students were sent these abstracts and were given the option of indicating which of the seminars they were interested in. At the time of admission, the faculty streamed students into classes according to their top three choices, limiting the enrollment of each seminar to twenty-five students. For many students, this would be the only class they would attend in their first year at the

University of Toronto with fewer than one hundred students. (Some first-year classes go into the thousands.) So another benefit of these classes is that first-year students would come into contact with a faculty member in a very small group. Instructors would get to know their students by name. In a university with over fifty thousand students, this by itself was already quite remarkable.

The first-year seminars typically cover a wide range of topics. The courses last an entire year, and while they do give students breadth and experience, they are not intended to be used as prerequisites for more advanced courses. So the curriculum can be very flexible, and students can take courses on topics that are only distantly related to what they hope to specialize in.

A typical first-year seminar in a science department (such as computer science) looks at ongoing research issues or controversies from a nontechnical point of view. Sometimes high school science or mathematics courses are required by the instructor, but with students coming from all over the world, it is difficult to depend on any specific background preparation. With a nontechnical approach, students can still be engaged in scholarly activity, tracking down references, discussing the ideas that emerge, making presentations, writing essays, and so on.

Computers and Thought

The first time I taught this course, I planned to look at my subarea of artificial intelligence (AI) research in just this way. I thought it would be worthwhile to have students understand what people in this area cared about and how they did their research. I also wanted students to read from the commentators who had written about AI (Turing, Fodor, Dennett, and Pylyshyn among them) and especially from the critics of the field (like Searle and Dreyfus) who felt that the work in AI could never illuminate what was going on in *people*.

The course did not work well. I came to feel that the students were not getting a clear enough picture of AI research to appreciate how there could be conflicting interpretations of it.

I started thinking that it might be better to give them some direct hands-on experience. Instead of having philosophers tell them why AI would never be able to do this or that, I wanted them to see for themselves what computers could do and to get a feel for some of the limitations. In a nutshell, I wanted them to try to program computers to perform activities that required thought.

This would be challenging for the students. The first-year students who take this course are not in a technical stream. I've had students majoring in sociology, criminology, commerce, history, biology, European studies, economics, political science,

curatorial studies, cinema, English, and religion. I clearly would have a hard time teaching them to write interesting AI programs in Python or Scheme in a single year.

However, I found that it was possible to teach nontechnical students to program in Prolog provided one did not insist on teaching them algorithms. It is possible, in other words, to get students to express what they need as a Prolog program and let Prolog search for answers. Small examples can be made to work well. The students get a real taste for the strength as well as the limitations of search, and especially the crushing power of combinatorics. *This is where the class had to go!*

At first I did a hybrid course, with a bit of technical work and a bit of philosophical discussion. But in the end, the philosophical part served mainly to give the students a break and keep them engaged. (When a topic is less technical, it is easier for students to feel that they can present their opinions and join in.) I came to believe that with a bit of effort on my part, there could be good class involvement and participation even in the technical part of the course. So Searle and his Chinese Room eventually fell by the wayside.

Cognitive science or artificial intelligence?

Although this book is not intended primarily for students majoring in a technical discipline like computer science or engineering, it is still a book about AI, not its sister discipline of cognitive science.

What is the difference between cognitive science and AI? While there are strong connections and overlap between the two, the main difference is that cognitive science is the interdisciplinary study of *people* as cognitive beings, whereas AI is the study of *intelligent behavior* achieved through computational means. The analogy I like (and take up in chapter 1) is the difference between studying flying animals and studying flight. Before the advent of aircraft, the only large-scale flying objects were animals like birds and bats. Cognitive science is like an interdisciplinary study of these flying animals, whereas AI is more like the study of what is sufficient for flight. Obviously, if one wants to understand flying animals, it helps to know something about flight in general; similarly, if one wants to understand the principles of flight, it helps to know something about the animals that fly. But the two areas have quite different objectives and methods.

Just to be clear, then, this is a book about thinking seen as a computational process and not about the thinkers themselves. As a technically restrained introduction to parts of AI, I believe it has a role to play in a cognitive science program. But it cannot replace the core material on the thinkers: psychology, neuroscience, biology, evolution, the social sciences, and so on.

Having said this, let me add that I do find it somewhat scandalous how far apart cognitive science and AI have drifted in the last twenty years or so. There are students studying AI who are led to believe that cognitive science is too soft and full of polemics to be of use. On the other side, there are students of cognitive science who are told that AI is mostly concerned with engineering applications, and just as well, since philosophers have demonstrated that AI cannot possibly have anything to say about the mind and what goes on in people. This is a very unfortunate state of affairs, and I can only hope that a new generation of students will put aside the prejudices of the past and see what there is to learn from both sides.

Overview of the book

The book follows the sequence of topics that I teach in the first-year course. However, it also includes some additional optional material (indicated by an asterisk before the section or chapter title) that in some cases is more advanced. The twelve chapters of the book are structured as follows:

- Chapter 1 is an introduction to the basic concepts. It talks about thinking and computing, and how they might be related. These topics are taken up again briefly in the philosophical conclusion of chapter 12.
- The next three chapters are related to Prolog. Chapter 2 introduces back-chaining informally; chapter 3 presents Prolog programs and queries; chapter 4 explains how to write the sorts of Prolog programs that will be used in the rest of the book. Additional features of Prolog are introduced as needed, including all of chapter 7 on lists in Prolog.
- The remaining chapters are case studies of various sorts of tasks where thinking appears to be required, and how they can be realized as Prolog programs. The tasks included are these: satisfying constraints (chapter 5), interpreting visual scenes (chapter 6), understanding natural language (chapter 8), planning courses of action (chapter 9), playing strategic games (chapter 10), and extending beyond Prolog to learning, explaining, and propositional reasoning (chapter 11).

Most of the chapters conclude with short bibliographic notes and exercises.

Is the book an introduction to artificial intelligence?

This book is about thinking as a computational process and, as such, falls squarely within the subject matter of AI. I hesitate to call the book an introduction to AI, however, since so much of AI is not even mentioned. Some of the missing topics might have been included in a longer book (for a longer course). Examples are knowledge discovery, Bayesian inference, and cognitive robotics.

But a lot of AI is not concerned with thinking at all (as it is understood here) and would be quite out of place in this book. Examples are sensory learning (like reinforcement conditioning), motor control (like legged locomotion), and early vision (like detecting shape from shading). There is a lot to be said about those topics, and about the interface between them and thinking (between early vision and the sort of visual interpretation seen in chapter 6, for instance). But they are so different from the thinking parts that it might be a mistake to try to combine them in a single introductory course.

Guide for the course instructor

At the University of Toronto, seminar courses like this one are full-year courses, which means two twelve-week terms, with 2 hours of lectures and 1 hour of tutorial each week. So I present the material from the book in 48 hours of lectures. (Tutorials are used for other purposes, such as going over the topics in the appendices, for example.) The breakdown of lectures is roughly as follows:

- 2 weeks to introduce thinking as computation (chapters 1, 2)
- 3–4 weeks for basic Prolog (chapters 3, 4)
- 3–4 weeks for satisfying constraints (chapter 5)
- 2 weeks for lists in Prolog (chapter 7)
- 3–4 weeks for natural language (chapter 8)
- 3–4 weeks for planning (chapter 9)
- 3 weeks for game playing (chapter 10)
- 1 week to conclude with some philosophy (chapter 12)

Because this course is not intended to be part of the students' specializations, I cover the material very slowly and make sure there is plenty of time for discussion, questions, and review in class. I skip over or breeze through many of the optional topics in the book.

I tell the students at the outset that nobody will have to drop out of the course because they find the material too difficult. We discuss how much time I expect them to spend on the course and I tell them that they can get a grade of B (or better) if they *attempt* everything that is asked of them, and get *correct answers* for the very easiest parts. Of course, to get a grade better than a B, they would need to go beyond this.

Here is how I arrange evaluation in the course. Over the entire year, there are five big assignments (requiring the students to write and submit Prolog programs), class participation in each term, and seven small homework tasks. These homework tasks require them to do various things, sometimes using pencil and paper, and sometimes using the computer, but do not require them to hand in anything. Instead, at the next tutorial, the teaching assistant goes over how to do the task, and asks them about what they have done to determine whether they have given the homework their attention. This, I believe, is a terrific way to get students (in a small class) to start thinking about what is needed for an assignment, and to clear up any confusion early in the process. Note that all the exercises in this book have been tested in the classroom.

Alternative courses using this book

The book could also work well for courses with different formats. For a half-year course (one twelve-week term, or 24 hours of lectures), I recommend something like the following:

- 2 weeks to introduce thinking as computation (chapters 1, 2)
- 3–4 weeks for basic Prolog (chapters 3, 4)
- 3–4 weeks for satisfying constraints (chapter 5)
- 2 weeks for visual interpretation (chapter 6)
- 1 week to conclude with some philosophy (chapter 12)

While this does present a somewhat impoverished view of thinking (tied to just constraint satisfaction), it nonetheless spans a range of examples, from recreational (like logic puzzles) to practical (like scheduling), and from artificial (like Sudoku) to natural (like vision).

What about a course for students with somewhat more technical facility, for example, second-year computer science students? For a half-year course with 24 contact hours, I recommend a more ambitious program:

- 1–2 weeks to introduce thinking as computation (chapters 1, 2)
- 2–3 weeks for basic Prolog (chapters 3, 4)
- 2–3 weeks for satisfying constraints (chapter 5)
- 2 weeks for lists in Prolog (chapter 7)
- 2–3 weeks for either natural language (chapter 8) or planning (chapter 9)
- 1 week to conclude with some philosophy (chapter 12)

With 36 contact hours or more, it should be possible to get through the full course, and even some of the more advanced topics like those in chapter 11.

Finally, what about a course for students with still more expertise, like students who already know Prolog? One possibility is to supplement the material in the book with algorithmic content: better algorithms for constraint satisfaction (such as arc consistency), for parsing (such as bottom-up methods), for planning (such as heuristic planning), for game playing (such as alpha-beta search). These are only hinted at in the text. Another possibility is to take the topics of chapter 11 on explanation, learning, and propositional reasoning as a springboard for a whole new section on numerical uncertainty, which has come to dominate so much of current AI research. But maybe the best option here is simply to use a more advanced text: I recommend the comprehensive textbook by Russell and Norvig [11].

Guide for the student

Let me start with a word of reassurance. *This book is not just for techies!* You will definitely need upper-level high school mathematics, and you will also need to know how to operate a computer: create text files, edit them, print them, and save them (see appendix A). You do not need to know how to program a computer, and there is actually not much mathematics in the book, mostly simple arithmetic.

So why is familiarity with mathematics necessary?

- You will need to be able to deal with variables and mathematical notation. Suppose the text refers to "seven words of English, x_1, x_2, ..., x_7." If this looks mysterious to you, you are probably not familiar with mathematical notation.

- You will need to be able to look at large symbolic expressions (or formulas), make sure they are put together correctly, and work on them precisely according to instructions. You need to be able to deal with a lot of tiny details without getting lost or bored. You might have picked up this skill in a variety of ways (like knitting from a complex pattern or building a model ship inside a bottle), but courses in mathematics are terrific practice.

Assuming you are interested in the subject matter of the book, do you have the necessary background to get through it all? There is no easy answer. I suggest you work through the first few chapters until you get to the exercises in chapter 4. If you have no idea of how to do the first few exercises there, then you should certainly consider stopping.

Going through this book as a self-study course is much harder than having an instructor who can answer questions as they come up. But if you are going through it alone, here are some things you definitely need to know:

- Any chapter or section with an asterisk ($*$) marking the title is an optional topic that may be more advanced. (A marked section within a marked chapter is even more advanced.) Skip these on first reading, or at least don't be disappointed if they are not clear to you at first.
- The exercises at the end of each chapter range from very simple to much more difficult. You should always be able to answer the first few questions, but don't be discouraged if you can't get through all of them.