Module H800: Symbolic Artificial Intelligence

Module (and Introductory Component) Guide

Andrew Tuson and Peter Smith (email: {andrewt, peters}@soi.city.ac.uk)

Contents

Chapter 1

Module Overview

This overview of the **H800: Symbolic Artificial Intelligence** module outlines its aims and structure, as well as the requirements and expectations of the students taking it. This guide should hopefully be able to answer most of the most basic questions that you will have concerning how the module is run.

1.1 Module Aims

This module is an introduction to the discipline of Artificial Intelligence (AI) and its application in building useful Knowledge Based Systems (KBSs) for practical problems. The module will be aimed towards achieving the objectives below:

- To provide students with a knowledge of the basic concepts in AI and KBS sufficient for further independent study.
- To give students an understanding and appreciation for what applications a KBS would be appropriate for.
- To allow students to gain experience with a tool (shell) for constructing KBSs.
- To outline basic methodologies for the principled design, construction, and validation of knowledge-based systems.
- To introduce the students to some common AI/KBS techniques with illustrative applications.

After completing this module, it is hoped that the student will be equipped to build practical small-scale AI/KBS applications in a principled manner, and to read a significant amount of the AI literature.

1.1.1 Availabilty

This module is available to students studying Department of Computing undergraduate courses. Though there are no prerequisites for this module, an interest in constructing useful 'intelligent', knowledge-based computer systems, the techniques for doing this, and the science behind it all would be an advantage.

1.2 Module Coverage and Delivery

This module is divided up into two streams, which is further divided into a number of self-contained 'components' to provide structure to the teaching and facilitate changes to the module in later years. The structure of the module, along with the lecturer(s) and the number of lectures/tutorials allocated to them are listed below in order of presentation.

1.2.1 Knowledge-Based Systems (Weeks 1-5)

The first stream aims to introduce the student to the practicial and methodological issues of designing and building knowledge-based systems, in the context of some common AI architectures.

1. Overview and Methodology, Tuson (2 lectures, 1 tutorial)

- What is AI - computing or philosophy? Why is it worthwhile - expert systems and KBSs?
- GOFAI, and other AI research paradigms.
- The Knowledge Level, Knowledge Representation and Search.
- Basic methodological issues (eg. experimental programming, neats vs scruffies).
- The KBS design process, and its differences to Software Engineering.
- What problems are suitable for a KBS?

2. Expert Systems, Tuson (4 lectures, 1 tutorial)
   - Basic Knowledge Representation (KR): introduction to production rules, logic (brief), decision trees, and semantic nets (brief).
   - The production system architecture and its applications.
   - Practical Issues: uncertainty, explanation, and meta-knowledge.
   - KBS Implementation: AI languages, expert-system shells and ESTA overview.
   - The knowledge acquisition (KA) bottleneck and overview of KA methods.
   - Issues in KBS verification and validation.
3. KBS Architectures and Modelling, Tuson (4 lectures, 2 tutorials)
   - Semantic nets (taxonomic and conceptual graphs).
   - Frames and blackboard systems.
   - Case-Based Reasoning.
   - What is a good representation?
   - Knowledge Modelling: ontologies and the CommonKADS methodology.

Note that there is a tutorial in week 3 which is dedicated to the coursework.

1.2.2 AI Search and NLP (Weeks 6-10)

The second stream complements the first by expanding upon the ideas introduced there to introduce the role of search in AI, and an important sub-field of AI: Natural Langauge Processing.

Note that there is a tutorial in week 10 which is dedicated to the coursework.

1.2.3 Component Delivery

Each component will be delivered by a combination of lectures and tutorials, along with any resources prepared for the component. The contact time totals 20 hours of lectures and 10 hours of tutorials spread over 10 weeks, once two additional tutorials for the coursework are included.

The aim of the tutorials is to support the lectures by allowing the students to obtain explanations/expansions of the material covered by the lectures from a different viewpoint, and to provide guidance/assistance on the courseworks.

Finally, the web page of this module can be found at: **`http://www.soi.city.ac.uk/~andrewt/H800/`**. Read this often for updates! This web page is only accessible from the **`*.city.ac.uk`** domain.

1.2.4 The Textbook

No single AI text covers the course in its entirety (mainly due to differences in emphasis). The approach taken here is to adopt a relatively inexpensive AI textbook which covers the majority of the material in this module, and supplement this with lecture handouts.

Alison Cawsey, ***The Essence of Artificial Intelligence*** (Essence of Computing Series), Prentice Hall, 1st Edition, ISBN 0-13-571779-5, 1998. £ 16.99, 225 pages.

There should be copies of this book for purchase in Waterstones downstairs, and it can also be ordered from **`www.amazon.co.uk`**. Some of the material covered in this module either expands upon or is additional to the contents of the textbook. In this case, supplementary materials will be handed out. Not all of this will necessarily be on-line.

Alternative explanations may also be useful, so students should feel free to read other AI texts. A list of some AI texts in the library is provided in the appedices of this document.

1.2.5 Courseworks

One coursework will be set, for which a report is to be submitted and code written. This will require the construction of a simple rule-based expert system using the ESTA expert system shell. Details of the course work are given later in this document.

1.2.6 Assessment

Assessment will be based on the coursework (30%) and a written examination (70%) with questions drawn from the material covered in lectures.

1.3 Feedback and Suggestions

Will be appreciated! As the AI provision has been greatly restructured in recent years, any feedback will be invaluable in improving it for the following year. Please feel free to contact either of the course lecturers, or use the feedback form provided at the end of the course.

Chapter 2

Component: Overview & Methodology

Lecturer: Andrew Tuson

This component will aim to give the student: an understanding of what AI is about (in terms of its scientific and engineering aims) and what it can be used for; its intellectual foundations; and provide a brief overview of AI methodology in both research and practice. This component consists of two lectures and one supporting tutorial.

2.1 Learning Outcomes

The learning outcomes from this component will result in the student be able to accomplish the following:

- Define the terms: Artificial Intelligence (AI); Knowledge-Based System (KBS); and Expert System (ES).
- Appreciate the difficulty in defining intelligence and intelligent behaviour, with reference to the 'Turing Test'.
- Understand and describe the following concepts:
  - 'Weak' and 'Strong' AI;
  - 'Knowledge Level' and the 'Knowledge Level Hypothesis';
  - 'Symbol System' and the 'Physical Symbol System Hypothesis' (PSSH);
  - '(Knowledge) Representation' and 'Inference (search)';
- Outline and differentiate the current AI research paradigms:
  - Symbolic functionalism/'good old-fashioned AI' (GOFAI);
  - Robotic functionalism;
  - Connectionism/'non-symbolic AI'.
- Appreciate the role of experimental programming in AI.
- Contrast the 'neat' and 'scruffy' approaches to AI research, and their (dis)advantages.
- Show an appreciation of the KBS design process, and describe the ways in which it differs from that of conventional software engineering.

2.2 Component Materials

Copies of slides will be given out at the start of the lecture, and will be supplemented by notes in this handout. The combination of the two should suffice. The exact coverage required for the exam will be, of course, defined by the lecture (so turn up!).

Chapter 3

Tutorial: AI Predictions

This tutorial is for the 'Overview & Methodology' component (in week 1)

In 1958 Simon and Newell made the following four predictions:

1. *That within ten years a digital computer will be the world's chess champion, unless the rules bar it from competition.*
2. *That within ten years a digital computer will discover and prove an important new mathematical theorem.*
3. *That within ten years a digital computer will write music that will be accepted by critics as possessing considerable aesthetic value.*
4. *That within ten years most theories in psychology will take the form of computer programs, or of qualitative statements about the characteristics of computer programs.*

For each of these four predictions answer the following two questions:

(a) Why do you think this prediction did not come true by 1968 or even by today?

(b) Do you think it will become true one day? If yes, what problems must first be overcome to make it true? If no, then what is the fundamental barrier which prevents it becoming true?

Acknowledgements

This tutorial was devised by Prof Alan Bundy at the University of Edinburgh. I would like to thank him for allowing us to adopt this tutorial exercise.

Chapter 4

Notes on AI/KBS Methodology

Author: Andrew Tuson

4.1 What is a KBS?

Early work in AI concentrated on the development of computer systems that *replicated* expert knowledge and performance (**expert systems**). However, it turned out that a useful system need not do this, if only because humans and computers have different strengths (eg. computers can calculate and do repetative tasks better than humans). A KBS relaxes the need for the system to perform in the same way as a human expert, allowing it to make use of its domain knowledge in whichever way the KBS designer *sees* fit. A succinct working definition of a KBS has been provided by [4]:

> "... a computer system that represents and uses knowledge to carry out a task."

Designers of such systems (**knowledge engineers**) take a strongly 'knowledge-is-power' approach to their design. This is exemplified by the **knowledge principle** [1]:

> **The Knowledge Principle**. To achieve a high level of problem-solving competence, a symbol system [ie. a KBS] must use a great deal of domain-specific, task-specific, and case-specific knowledge.

The question is, if knowledge needs to be used, how to do we describe (and thus design!) a KBS? Conventional computer systems do not make any explicit reference to knowledge so a change of viewpoint is required.

NOTE: the terms **domain-specific**, **task-specific**, and **case-specific** will be discussed later on in the module.

4.2 The Knowledge Level

The meanings of 'knowledge', 'symbol system' and 'representation' need to made clear. This is of particular importance in the context of this work because, despite the realisation that domain knowledge is required, we must be clear about how this relates to KBS. The AI literature fortunately addressed this issue some time ago. Work by [3] considers computer systems as consisting of **levels of description** such as:

- The **Knowledge Level**: a description of the knowledge contained in the system.
- The **Symbol Level**: a description of the data structures used by the system, and their manipulations.
- The **Program Level**: the implementation of an effective procedure (algorithm) that carries out the manipulations specified for the system.
- The **Hardware Level**: the physical realisation of the system.

Newell's contribution was to note that in addition to the symbol and lower levels, there was in fact a higher level of description — the **knowledge level** which corresponded to the knowledge used by the system [3].

The Knowledge Level Hypothesis. There exists a distinct computer systems level, lying immediately above the symbol level, which is characterised by knowledge as the medium and the principle of rationality as the law of behaviour.

The **principle of rationality** refers to the assumption that a problem-solving behaviour can be explained in terms of the system's body of knowledge (ie. knowledge base) [2] — in other words there is a causal relation between the agent's (ie. the KBS's) knowledge and its behaviour.

4.3 Representation and the Role of Levels

Therefore we can see that a symbol system is a symbol-level implementation (aka. **representation**) of a computer system defined at the knowledge level. In other words, a representation is the mapping between the knowledge and symbol levels of a KBS.

This discussion leads us to the main advantage of the separation of levels: if KBS design is made at the symbol level, there clearly is more than one way that a given useful item of domain knowledge can be implemented. Therefore, if the principle of rationality is to hold then the relationship between the knowledge and symbol level must be clear — as this confers the advantage that the system's behaviour can be justified in terms of its body of knowledge.

4.4 Remaining Learning Outcomes

The coverage that is provided by the slides (plus what else I say in the lecture) will be sufficent.

Chapter 5

Coursework: Building a Simple KBS

Setter: Andrew Tuson

This coursework requires you to perform the design and construction of a simple KBS using a rule-based expert system shell (ESTA), where a documented system is to be delivered and assessed. This is aimed at exercising your ability to apply the material covered in the course in a principled fashion, and to give you some experience with an expert system shell. Andrew Tuson is responsible for this coursework.

All you need to know to do this coursework is covered in the first three weeks of this module, and two tutorials have been assigned to this coursework. In addition, you are free to discuss coursework-related issues in the other tutorials (though remember that the work set in these are to help you learn the material for the exam!).

5.1 Selecting a Problem

You should select a small-scale problem to tackle. I would say that an rule base of about 15-25 *rules*[1] would be about right. You are free to select any problem you like so long as there is no scope for plagorism (eg. two people doing the same/very similar problems), and that your tutor feels that it is feasible. Andrew Tuson is the final arbiter in any disputes.

5.1.1 Restrictions

Absolutely no courseworks related to mobile telephony will be accepted. I am bored with marking them, and you should have more imagination!

5.2 The ESTA Expert System Shell

The expert system shell you will be using is ESTA (Expert System for Text Analysis), part of the Visual Prolog freeware distribution (links to the ESTA and Visual Prolog web pages can be found on the module web page). Therefore you can obtain a copy if you would like to work on a PC at home (aren't I nice!...:-). This shell *should* be available on all CSD and Sol PCs (if it is not then let me know!). It should be (fairly) straightforward to use - there is not a large syntax to learn. All you need to know is on the ESTA help facility.

A getting started guide to ESTA is included in the appendices of this document. It describes how to build something really simple with a few rules and sections. Students in previous years found that getting to this stage was the hardest part. Once something simple is up-and-running, it becomes relatively easy to extend the system to the complexity required.

Though these are the only documents available, a lecture in week 3 will be dedicated to looking at ESTA and expert system shells in general.

5.3 Checkpointing - An Early Deliverable

I feel that, given the open-ended nature of this coursework, that you should start work on this early on. To encourage you to do this up to 10% of the marks are allocated for showing and discussing with your tutor, in the

[1] By which I mean IF-constructs (paragraphs) in ESTA.

week 3 tutorial, a draft report (or if you miss that, by the end of that week - see below). The report should be 1-2 pages long and describe the following:

- the problem you propose to build a system for;
- why it is suitable for a expert system;
- how you intend to acquire the knowledge;
- and why you feel that it is feasible in the time available.

There are two deadlines, the later of which carries a reduced mark and for which I will not give much (if any!) feedback.

Deadline for 10% Mark: Tutorial Session of Week 3

and failing that, see me (Andrew Tuson) by....

Deadline for 5% Mark: 5pm Friday of Week 3

You are free to modify your proposed problem for the final report in light of, for example, your tutor's comments (In other words, I will not hold you to what you say in this deliverable!).

5.4 The Final System & Report

Once a problem has been selected, you are to produce a working, documented expert system using ESTA and evaluate its performance. The ESTA knowledge base (the `.kb` file) will be submitted and assessed. The report should include an updated version of the material in the checkpoint report. Overall it should be a full description of the following:

- the problem the system is trying to solve;
- why it is worth doing/suitable for an expert system;
- how you obtained and formulated the knowledge;
- the system/knowledge-base design (e.g. the ESTA sections used, and how the rules were arrived at);
- what issues were confronted in the system construction and how these were dealt with;
- an evaluation of how well the system performs its task and what improvements could be made.

In addition, the report will be marked on the following criteria:

- Presentation, structure, accuracy, and clarity of report;
- A discussion of the issues above with analysis and examples (where appropriate);
- Justification of any decisions made drawing on the material covered on the course (where appropriate).
- Explicit connections made between decisions at each stage.

The knowledge base[2] will be marked on the following criteria:

- Clear structure and commenting of the knowledge base source;
- The validity of the facts in the knowledge base (eg. correctness, resolution of ambiguities);
- Whether the system actually functions as intended (ie. is stable, produces no unexpected behaviour, outputs as claimed by your report).

I would estimate that 3000-4000 words (6-8 pages of A4) is probably the right length (though definitely no longer). Try to be consise and to the point — we do not mark by weight!

Feel free, if there is not too much of it (ie. < 5 pages), to include photocopies of any materials you used in your knowledge acquisition (however keep any materials to hand in case I want to check something). I plan to give *about*[3] about 60% the marks for the report, and the remainder of the marks for the knowledge base (code).

[2] I will be using the report to guide the marking of the knowledge base, therefore if there is functionality which is not mentioned in the report, I may not find it (and thus not be able to give you marks for it).

[3] This means that I can be flexible about assessing what you have done on its merits.

5.4.1 Summary of Deliverables

The deliverables are therefore as follows, the usual coursework submission procedures for these apply:

1. The Checkpoint Document (Deadline: Week 3)
2. The Final System (Deadline: 5pm Monday Week 11)
   (a) A report describing the functionality, design, construction, and testing of the system (hard copy submission);
   (b) *Hard-copy* of the ESTA knowledge base, appended to the above report;
   (c) A fully commented, well-structured, functioning, ESTA knowledge base (electronic submission).

Note that the electronic submission system at: **`http://www.soi.city.ac.uk/tsg/management/`** *must* be used (i.e. no email submissions!).

Deadline for Final System: 5pm Monday of Week 11

5.5 Final Remarks

First of all, though this coursework has been set up to make this more difficult...

Plagorism (cheating) will lead to a LARGE reduction in marks
(and don't think that you won't get caught)!

Remember - do not leave this to the last minute! It will take some time for you to get used to the ESTA environment and to collate the information you need. Good luck!

Bibliography

[1] D. B. Lenat and E. Feigenbaum. On the thresholds of knowledge. *Artificial Intelligence*, 47(1-3):185–250, 1991.

[2] E. Motta. *Resusable Components in Knowledge Modelling*. PhD thesis, Open University, UK, November 1997.

[3] A. Newell. The Knowledge Level. *Artificial Intelligence*, 18(1):87–127, 1982.

[4] M. Stefik. *Introduction to Knowledge Systems*. Morgan Kauffmann, 1995.

Appendix A

List for Further Reading

Though the recommended text and handouts should suffice for the entire module, the following is a list of books you may find useful to read for further reading or alternative explanations (there may be others!). Books that we are familiar with are marked *. Shelf-marks for City University Library are given where available.

A.1 General Reading

- *Russell and Norvig. Artificial intelligence: a modern approach, Prentice Hall, 1995 (006.3 RUS).
- *Nilsson. Artificial intelligence: a modern approach, Morgan Kauffman, 1998 (006.3 NIL).
- Garnham. Artificial intelligence: an introduction, Routledge & Kegan Paul, 1987 (006.3 GAR).
- *Rich and Knight, Artificial intelligence (2nd ed), McGraw-Hill, 1990 (006.3 RIC).
- *Winston. Artificial intelligence (3rd ed), Addison-Wesley Pub. Co, c1992 (006.3 WIN).
- *Ginsberg. Essentials of artificial intelligence, Morgan Kaufmann Publishers, 1993 (006.3 GIN).
- *Turban. Expert systems and applied artificial intelligence. Maxwell Macmillan International, c1992 (006.3 TUR).
- Ford. How machines think: a general introduction to artificial intelligence. Wiley, c1987, (006.3 FOR).
- O'Shea et al. Intelligent knowledge-based systems: an introduction. Harper & Row, 1987 (001.535 OSH).
- Simons. Introducing artificial intelligence. Manchester NCC, 1984 (006.3 SIM).
- *Jackson. Introduction to expert systems. Addison-Wesley, 1990. (006.33 JAC)
- *Luger and Stubblefield. Artificial Intelligence. Addison Wesley 1997.

A.2 Expert Systems Issues & CBR

- *Hart. Knowledge acquisition for expert systems (2nd ed). Kogan Page, 1989 (006.33 HAR).
- *Jackson. Understanding expert systems using crystal. London: Wiley, c1992 (006.33 JAC).
- *Kayney et al. Knowledge Engineering, Open University, 1989 (006.3 BRA).
- *Watson. Applying Case-Based Reasoning: Techniques for Enterprise Systems, Morgan Kaufmann, 1997 (006.33 WAT).
- Harmon and King. Expert Systems. Wiley, 1985 (006.33 HAR).
- Buchanan and Shortliffe. Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley, 1985 (006.33 BUC).

A.3 Search/Evolutionary Computation

- *Reeves. Modern heuristic techniques for combinatorial problems. Blackwell Scientific Publications, 1993 (519.7 REE).
- *Goldberg. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Pub. Co, c1989 (006.3 GOL).
- *Koza. Genetic programming: on the programming of computers by means of natural selection. MIT, c1992 (006.3 KOZ).
- *Koza. Genetic programming II: automatic discovery of reusable programs MIT, c1994 (006.3 KOZ).
- *Michalewicz. Genetic algorithms + data structures = evolution programs. Springer-Verlag, c1996 (005.1 MIC).
- *Banzhaf. Genetic Programming: An Introduction. Morgan Kaufmann. 1998.
- Pearl. Heursitics: Intellegent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984 (006.3 PEA).

Finally if you come across other books that you feel should be on this list — please let us know!

Appendix B

A Brief Guide to ESTA

B.1 Overview

In the context of the coursework, the ESTA shell has two distinct advantages.

- It is relatively simple to use.
- It is free!

ESTA is a simple expert system shell, that broadly works according to a rule-order/refraction conflict resolution strategy. It however does allow modularisation, multiple variable types, and provides a GUI environment. This guide will provide you with a walkthrough of the subset of ESTA features and programming constructs required to do the module coursework.

B.1.1 Please Note: Additional Help

This is not intended to be complete! More information can be found in the ESTA help facility. This gives a comprehensive overview of the language features with examples. As you have all had to pass programming modules to get this far, coding in ESTA should be a piece of cake! Therefore **no support** will be provided for routine programming issues such as debugging — working out such issues is why the (small) system build is assigned 40% of the final deliverable marks. The 99-00 undergraduate cohort managed fine with just the ESTA help system!

B.1.2 Obtaining a Copy of ESTA

There are three ways to use/obtain ESTA.

- It should be on all CSD PCs. Look under 'departmental software'.
- You can download a copy as a WinZip distribution from the module web page (instructions are given there).
- Buy a copy of the Visual Prolog CD-ROM which contains ESTA (costs money – I doubt that this will be a popular option). A link to the Visual Prolog web page can be found on the module web page.

Please note that ESTA can *only* be used for teaching and academic purposes. The ESTA-only distribution provided on the module web page can *only* be used in City University, or by its students and staff. If anyone else would like to use ESTA (or you would like to use ESTA/Visual Prolog for another purpose), we refer you/them to the Visual Prolog website where they can obtain a copy.

B.2 Getting Started: Defining Parameters

The next step is to define **parameters**, which are typed variables used to represent the facts in working memory. Parameters can be edited via the **Parameter** menu. Of course we now need an example problem to work from.

B.2.1 The Example Problem (Of Course!)

The (trivial) problem we will consider is that of calculating how drunk someone is, based on what they have drunk, how much, and whether they have eaten before. In this domain, shandy is half the strength of stout which is half the strength of German beer. Furthermore eating beforehand halves the effect of what you have drunk.

B.2.2 Text Parameters

First of all, we'd like to hold the name of the user of the system. This will be acheived by using a **Text** parameter

Use the menu option **Parameter → New Parameter** and type in the parameter name (**yourName**) and select **text** as the type. A code stub will appear as shown below.

```
parameter yourName : ' '
type text
explanation ' '
/* rules field */
question ' '
picture ' '
```

Ignore and remove the picture field, as well as the reference to a rules field (which is a comment encased by **/* ... */**). Now edit the text editor window to produce the following.

```
parameter yourName : 'The name of the subject'
type text
explanation 'So we know who you are!'
question 'What is your name?'
```

The structure of the parameter code is as follows. The first line declares the name of the parameter and allows a brief description of the parameter (remember: good programming practice!). The next line denotes the parameter's type. The question field specifies the text of the question that is asked. Finally, the (optional) explanation field specifies additional text that is displayed if the user asks for an explanation of the question.

Closing the editor window, will produce a prompt that asks whether you want to update the knowledge base, if so the definition is accepted after some checks that the syntax is legal.

A Title!

It may be nice to give the expert system a title. Select **Title → New Title** and type in the text for the title (e.g. **An Expert System for Drunks!**) in the editor window as before.

B.2.3 Boolean Parameters

This is a parameter type intended for true/false (**boolean**) questions. We can use this to create a parameter (**eatenYet**) that determines whether the subject has eaten before drinking. This is defined as follows.

```
parameter eatenYet : 'Whether the subject has eaten'
type boolean
explanation 'Its good to line the stomach!'
question 'Did you eat before you went to the Pub?'
```

NOTE: there is in fact a third value that this parameter can take — **unknown**. Personally I think this is daft! Feel free to use a category parameter instead if you want the unknown option removed.

B.2.4 Category Parameters

Some questions require a choice out of a number of options. This is captured by the **category** parameter type. An example of this would be specify which type of beer was drunk out of a list. The category parameter **typeBeer** is defined below.

```
parameter typeBeer : 'Type of beer drunk'
type category
explanation 'Some beer is stronger than others'
options
  option_stout - 'Stout',
  option_german - 'Strong German Beer',
  option_shandy - 'Weak Shandy'.
question 'What type of beer are you drinking tonight?'
```

There is one new field — **options** — that needs explaination. This denotes each of the options in term with declarations of the form **<option-label> - <description-string>**. The **<option-label>** term specifies one value that the parameter may take. Therefore in the example above the parameter **typeBeer** can adopt one of the values **option_stout**, **option_german**, and **option_shandy**. Finally, the **<description-string>** specifies the descriptive text that will be used when the option(s) are presented to the user.

B.2.5 Number Parameters

Of course we need to know the number of pints of beer drunk. This is acheived using the **number** parameter **numPints**. Type the following into the text editor window.

```
parameter numPints : 'Number of pints of beer consumed'
type number
explanation 'Count the empty glasses!'
range 0 20
question 'How many pints of beer have you consumed?'
```

Note the use of the **range** field, this specifies the range of values which this parameter can take.

We also should define an intermediate parameter, **effectivePints**, which is used to store the calculation of how much is *effectively* drunk, after the factors due to beer type, etc. have been taken into account. The definition is below.

```
parameter effectivePints : 'how much is effectively drunk?'
type number
range 0 50
```

Note that the question and explanation fields have been omitted. This is because this parameter obtains its value by assignment (see below) and not by questioning the user.

B.2.6 Saving Your Work

Finally, use the **File → Save as..** option to save what you have done so far as a **.kb** file with an appropriate name. I cannot guarantee the stability of ESTA (hey, its free!), so I would advise you to save your work at regular intervals (or risk losing it!).

B.3 Building the Rules: Defining Sections

Once the parameter for the domain have been defined, we can now put in the rules. ESTA uses a form of modularisation called **sections**, which break up inference. It is therefore advisable to take advantage of this and break the reasoning into stages.

In this case the reasoning was broken down into the following stages:

1. Opening up the dialogue and getting the users name (**start**).
2. Finding out how much was drunk (**getNumPints**).
3. Finding out what type of beer it was and adjusting accordingly (**getBeerType**).
4. Finding out whether the user has eaten already and adjusting accordingly (**hadDinner**).
5. Deciding and displaying the state of drunkenness (**resultBit**).

B.3.1 The 'start' Section

All ESTA knowledge bases should start with a **start** section, which is where the shell will start when the knowledge base is consulted. In this case, the section below, only moves onto the main consultation if the user submits a non-empty text string as a name.

```
section start : 'this is the root section'

if yourName <> ''
do getNumPints

if yourName = ''
(
advice 'Next Time - Give a Name!',
exit
)
```

The following points should be noted with respect to the section above:

- The operators **<>** and **=** are not equals and equals respectively.
- Parentheses are used to denote a block of statement in an **if**-statement/rule (there is no 'else'!).
- The **exit** statement ends the ESTA consultation.
- The **do** statement moves the consultation onto the next section, in this case **getNumPints**.

B.3.2 The 'getNumPints' Section

The **getNumPints** section determines how much the user has has to drink. As noted above, if a variable is used which has not received a value, the user is asked to provide one.

```
section getNumPints : 'finds out how much was drunk'

assign effectivePints := numPints

do getBeerType
```

In this case, the **assign** statement assigns the value of **numPints** to the parameter **effectivePints**. Once the assignment has taken place, control moves to the **getBeerType** section.

B.3.3 The 'getBeerType' Section

This section determines what type of beer is being drunk and updates **effectivePints** accordingly, as well as commenting on the strength of the brew.

```
section getBeerType : 'finds out what was drunk'

if typeBeer = 'option_stout'
(
advice 'Stout is of average strength',
do hadDinner,
)

if typeBeer = 'option_german'
(
advice 'German beer is of high strength',
assign effectivePints := effectivePints * 2,
do hadDinner,
)
```

```
if typeBeer = 'option_shandy'
(
advice 'Shandy is for wimps!',
assign effectivePints := effectivePints * 0.5,
do hadDinner,
)
```

As can be seen above, the **assign** statement can also make use of mathematical expressions.

B.3.4 The 'hadDinner' Section

The next stage is to find out whether the subject has eaten, and to adjust accordingly.

```
section hadDinner : 'did the subject have something to eat?'

if eatenYet
assign effectivePints := effectivePints/2

do resultBit
```

B.3.5 The 'resultBit' Section

Finally, this section (**resultBit**) displays the state of drunkeness depending on the value of **effectivePints**.

```
section resultBit : 'reports the state of drunkeness'

advice yourName ' has drunk ' numPints
' pints which is the same as '
effectivePints
' pints once beer type, etc is considered.'

if effectivePints = 0
(
advice 'You are as sober as a Judge!',
)

if effectivePints > 0 and effectivePints < 5
(
advice 'You are tipsy!'
)

if effectivePints >= 5 and effectivePints < 10
(
advice 'You are drunk!'
)

if effectivePints >= 10 and effectivePints < 15
(
advice 'You are very drunk!'
)

if effectivePints >= 15
(
advice 'Call an ambulance!'
)

advice 'Thank you for using this expert system!'
exit
```

Note that in the first **advice** statement it is possible to chain strings and parameters together to produce more complex messages.

B.4 Running the Knowledge Base

The expert system you have written can be run by selecting the **Consult** → **Begin Consultation** menu item.

If there are errors, ESTA will present a (usually) common-sense error message. In addition to this, the **Consult** → **Check Knowledge Base** menu item will perform a check on the correctness of the knowledge base.

HINT: try to deliberately introduce errors in the code, and record the error messages. This will help you later when you try to debug your system,

Appendix C

Tutor's Notes: 'AI Predictions'

I suggest you consider each prediction in turn and ask each of your tutees to comment on (a) and (b) in turn. Vary the order of tutees for each prediction. Try to generate a discussion. Let them do the speaking. If you need to ginger it along I attach some notes on each prediction that may help. Be alert for signs of 'mental vitalism' in negative answers to (b). Try to crystallise these, *i.e.* "so you are saying a computer could never do X".

1. *That within ten years a digital computer will be the world's chess champion, unless the rules bar it from competition.*

   (a) Chess has a huge search space. The combinatorial explosion defeated early chess programs based on mini-max. Since 1968 a few attempts have been made to develop more human-like and less brute-force approaches, *e.g.* based on planning. This work has largely stopped because hardware developments have made the brute-force approach viable. They have not quite developed to the point that a computer is world champion.

   (b) The consensus among computer chess experts is that this prediction is likely to become true in the next few years. The world champion was recently defeated in some matches, but not overall. Future hardware developments are likely to give computers the edge soon. It is a moot point whether this brute-force solution would tell us anything about intelligence.

2. *That within ten years a digital computer will discover and prove an important new mathematical theorem.*

   (a) Again the combinatorial explosion has defeated complete, but uniform, theorem provers, *e.g.* those based on resolution. As with chess the situation has improved dramatically since 1968 with improvements in hardware, coding techniques and heuristics, but not to the point where any important theorem has been proved completely automatically. Some open conjectures have been proved, *e.g.* by the Argonne provers, but these are in esoteric areas of maths where human intuition is weak and brute-force approaches are competitive. None of these theorems could be described as 'important'. Important theorems have been proved interactively, *i.e.* by human/machine collaboration. The best known example is the four colour theorem[1].

   (b) Unlike the chess case it looks unlikely that improvements in hardware will be sufficient. We first need to understand better human methods for guiding search.

3. *That within ten years a digital computer will write music that will be accepted by critics as possessing considerable aesthetic value.*

   (a) The main problem here is probably trying to give a computational account of 'aesthetic value'. There have been a few attempts to automatically generate music. Some of it certainly has some 'aesthetic value', but not a 'considerable' amount of it. An example is Mark Steedman's program to generate jazz.

   (b) There is some interesting work on the implicit rules which guide musical composition and performance. This is gradually improving our idea of musical 'aesthetic value'. Automated music is gradually improving as a result. Here is one area where brute-force solutions seem unlikely to batter the problem to death.

4. *That within ten years most theories in psychology will take the form of computer programs, or of qualitative statements about the characteristics of computer programs.*

[1] That four colours are always sufficient to colour any 2D map with no adjacent countries having the same colour.

(a) There are now quite a lot of computational psychological theories — many provided by Newell and Simon. But we are still a long way from 'most' theories being of this form. Partly, this may be due to the difficulty of developing such theories. Partly, it may because a huge paradigm shift in Psychology would be required to make it happen.

(b) Discussion here might focus on what might be required to cause a paradigm shift in Psychology to computational models. In particular, whether psychologists and the students will 'buy' a computational account of mind.

Acknowledgement

This solution was that provided by Alan Bundy (Edinburgh). If you have any additions/improvements to make to this, let me know for future versions.