SCHOLAR Study Guide

SQA Higher 2004-5
Computing Unit 3a
Artificial Intelligence

Approved for use in the SCHOLAR Forum schools in 2004-5.

David Bethune
Heriot-Watt University

Andy Cochrane
Heriot-Watt University

Tom Kelly
Heriot-Watt University

Ian King
Heriot-Watt University

Richard Scott
Heriot-Watt University

Interactive University

Edinburgh EH12 9QQ, United Kingdom.

First published 2004 by Interactive University

Interactive University publishes and delivers, through Local Learning Partners around the world, a range of programmes from universities and colleges in Scotland.

British Library Cataloguing in Publication Data

Interactive University

SCHOLAR Study Guide Unit 3a: Computing

1. Computing

ISBN 1 904647 35 9

Typeset by: Interactive University, Wallace House, 1 Lochside Avenue, Edinburgh, EH12 9QQ.

Printed and bound in Great Britain by Graphic and Printing Services, Heriot-Watt University, Edinburgh.

Part Number 2004-1113

Contents

Acknowledgements

Thanks are due to the members of Heriot-Watt University's SCHOLAR team who planned and created these materials, and to the many colleagues who reviewed the content.

Programme Director: Professor R R Leitch

Series Editor: Professor J Cowan

Subject Directors: Professor P John (Chemistry), Professor C E Beevers (Mathematics), Dr P J B King (Computing), Dr P G Meaden (Biology), Dr M R Steel (Physics), Dr C G Tinker (French)

Subject Authors:

Biology: Dr J M Burt, Ms E Humphrey, Ms L Knight, Mr J B McCann, Mr D Millar, Ms N Randle, Ms S Ross, Ms Y Stahl, Ms S Steen, Ms N Tweedie

Chemistry: Mr M Anderson, Mr B Bease, Dr J H Cameron, Dr P Johnson, Mr B T McKerchar, Dr A A Sandison

Computing: Mr I E Aitchison, Dr P O B Holt, Mr S McMorris, Mr B Palmer, Ms J Swanson, Mr A Weddle

Engineering: Mr J Hill, Ms H L Jackson, Mr H Laidlaw, Professor W H Müller

French: Mr M Fermin, Ms B Guenier, Ms C Hastie, Ms S C E Thoday

Mathematics: Mr J Dowman, Ms A Johnstone, Ms O Khaled, Mr C McGuire, Ms J S Paterson, Mr S Rogers, Ms D A Watson

Physics: Mr J McCabe, Mr C Milne, Dr A Tookey, Mr C White

Learning Technology: Dr W Austin, Ms N Beasley, Ms J Benzie, Dr D Cole, Mr A Crofts, Ms S Davies, Mr A Dunn, Mr M Holligan, Dr J Liddle, Ms S McConnell, Mr N Miller, Mr N Morris, Ms E Mowat, Mr S Nicol, Dr W Nightingale, Mr R Pointon, Mr D Reid, Dr R Thomas, Dr N Tomes, Ms J Wang, Mr P Whitton

Cue Assessment Group: Ms F Costigan, Mr D J Fiddes, Dr D H Jackson, Mr S G Marshall

SCHOLAR Unit: Mr G Toner, M G Cruse, Ms A Hay, Ms C Keogh, Ms B Laidlaw, Mr J Walsh

Media: Mr D Hurst, Mr P Booth, Mr G Cowper, Mr C Gruber, Mr D S Marsland, Mr C Nicol, Mr C Wilson

Administration: Ms L El-Ghorr, Dr M King, Dr R Rist,

We would like to acknowledge the assistance of the education authorities, colleges, teachers and students who helped to plan the SCHOLAR programme and who evaluated these materials.

Grateful acknowledgement is made for permission to use the following material in the SCHOLAR programme:

To the Scottish Qualifications Authority for permission to use Past Papers assessments.

The financial support from the Scottish Executive is gratefully acknowledged.

Topic 1

What is Intelligence?

Contents

Learning Objectives

After studying this topic, you should be able to:

- *state some definitions of human intelligence;*
- *state some definitions of artificial intelligence (AI);*
- *explain the difficulties in defining intelligence and AI;*
- *list some aspects of intelligence;*
- *describe the Turing Test and explain its flaws.*

1.1 Introduction

Most of the computing you have studied so far has been about how computers systems work, how they can be programmed, and how you can use standard applications to do useful things. Artificial Intelligence is different! Artificial Intelligence is one of the most exciting and varied areas of computing research and development. But what is it all about?

In this topic, you will explore how AI has developed since its beginnings around 1950 to the present. This will help you to understand the wide range of areas of research, and some of the issues raised by this fascinating area.

As you study the topic, try to think about the links to what you already know about computing. The twists and turns of the story of the development of AI are often linked to the hardware and software and expertise of the time. Sometimes people had interesting ideas, but the hardware available was not up to the task of implementing the idea. Sometimes, too, a development in AI has led to improvements in hardware and software that benefit much wider areas of Computing.

This topic can only help you get started. There is lots more to learn if you are interested! Hopefully, this will whet your appetite!

1.2 What is artificial intelligence (AI)?

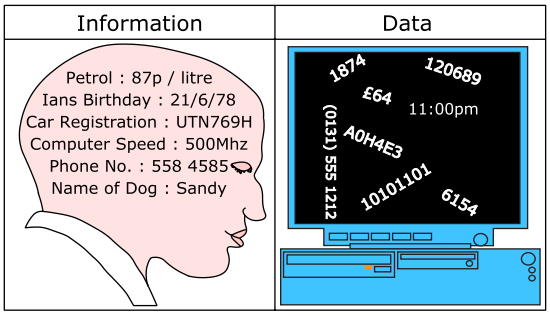
Learning Objective

After studying this topic, you should be able to:

- explain some difficulties in defining AI;
- explain some difficulties in defining intelligence;
- list some aspects of human intelligence.

Artificial Intelligence is one of the most exciting and varied areas of Computing research.

So what is it all about? One of the problems (or maybe, one of the delights) of the subject is that nobody seems to be able to agree what it is! That makes it hard to pin down, but it also means that the scope is unlimited.



Here are a few definitions that have been suggested over the last 50 years:

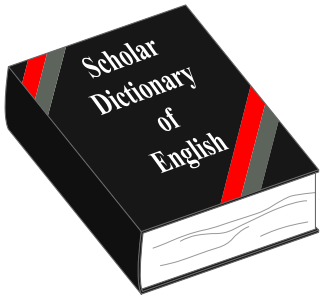
- "The science of making machines do things that would require intelligence if done by men";
- "Ultimately, the goal of Artificial Intelligence is to build an artificial human - or more humbly, an animal ....";
- "A field of study that seeks to explain and emulate intelligent behaviour in terms of computational processes";
- "The study of how to make computers do things which, at the moment, people do better";
- "The use of computers in such a way that they perform functions normally associated with human intelligence";
- "Artificial Intelligence is concerned with building machines that can act and react appropriately, adapting their response to the demands of the situation".

Definitions of AI

10 min

Look up "artificial Intelligence" in any dictionary or on-line encyclopaedia that you have access to, and write down any definitions you find. Use as many resources as you can. For example, you may have access to:

- an ordinary dictionary;
- a dictionary of Computing terms;
- a passing teacher!;
- an on-line encylopaedia (such as *Encarta*).



Look at the definitions you have written down.

Are they all much the same?

Compare the definitions you have written down with the ones in the notes above.

Are they similar?

There are several reasons why no-one can agree on a definition. Here are three:

1. It is a subject which is developing all the time, so the definition has to keep changing;
2. It is a subject which involves computer scientists, biologists, psychologists and philosophers (among others), and they all see the subject from their own perspective;
3. It is difficult to define human intelligence, let alone artificial intelligence.

Marvin Minsky was a very influential thinker in the earliest days of AI. He said that AI is the "science of making machines do things that would require intelligence if done by men".

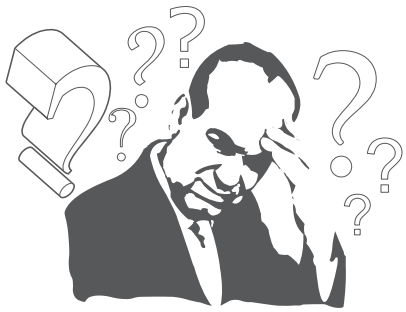
We need to ask what **intelligence** means! What kind of activities "require intelligence"?

20 min

Does it require "intelligence"?

Make a list of "things which (in your opinion) require intelligence".

Compare this with another student in your group.



Here are some things you may or may not have included in your list. For each one, decide whether or not it "requires intelligence".

- understand Einstein's Theory of General Relativity;
- get an "A" in Higher Maths;
- be a (successful) Air Traffic Controller;
- be Prime Minister;
- be a teacher;
- be a street sweeper;
- carry out (successful) open heart surgery;
- write a novel;
- write your name;
- paint a beautiful landscape;
- paint a ceiling;

- write a piece of music;
- conduct an orchestra;
- play the kazoo;
- bake a cake;
- make a cup of tea;
- repair a faulty computer;
- speak to an external assessor about a school project;
- speak to a friend on the phone;
- recognise a friend in a crowded room;
- recognise your master's voice (if you are a dog!);
- play chess;
- become world chess champion;
- play noughts and crosses;
- choose a birthday present for your brother;
- explain to the shopkeeper why you are returning the item you bought;
- drive a car;
- walk along the street;
- tie your shoe laces;
- dress yourself;
- breathe;
- remember enough to pass Higher Computing;
- remember your name;
- learn to speak a foreign language;
- learn to speak.

Compare your results with someone else in your group.

Can you summarise what you mean by "intelligence"?

As a group, try to write down 5 or 6 general types of behaviour that "require intelligence".

1.2.1 Definitions of intelligence

There are many different definitions of "intelligence". According to our version of the Chambers Dictionary, intelligence means "intellectual skill or knowledge; mental brightness" and intelligent means "endowed with the faculty of reason; having or showing highly developed mental faculties; alert, bright, quick of mind; well-informed; knowing, aware". Other dictionaries will have quite different definitions.

Just as there was no agreed definition of **artificial** intelligence, there is no universally agreed definition of **natural** intelligence.

However, it is useful to make a list of some general areas that should be included in any discussion of intelligence. Here are some:

- the ability to learn;
- creativity;
- reasoning and the ability to explain;
- use of language;
- vision and making sense of the environment;
- problem solving;
- adapting to new situations;
- cognitive ability;
- memory.

If a computer system is able to do any of these things, we could say that it is showing some form of *artificial intelligence*. Notice that we are **not** saying that a computer or program is *intelligent*; we are simply saying that it *shows intelligent behaviour*. This avoids getting into fascinating (but ultimately fruitless) discussions about whether computers can think, feel emotions, or be conscious.

It is important to distinguish between *intelligence* and *intelligent behaviour.* This is true of humans as well as computer systems! After all, I have no way of knowing if anyone else in the human race can think or feel emotions or is conscious, other than myself. All I can say is that other people appear to behave (at least some of them for some of the time!) as though they can think and feel emotions. I can observe their *behaviour*, but I have no way of knowing what they are *experiencing in their own heads*. As to whether or not other people are conscious: it is possible that I am the only conscious being in the universe, and everybody else is a figment of my imagination .... The same is true of any computer system.

All this discussion about the difficulties of defining intelligence, and artificial intelligence means that AI has no definite boundaries. If you have the sort of mind that likes everything to be neat and precise, you won't like this, but it does mean that AI can flourish. New and unexpected developments can be included under the broad definitions we have considered. AI is a bit like an umbrella under which many different areas of study can shelter. During this course we will consider several of them, and trace the historical development of the subject.

1.2.2 Review questions

Q1: Why is it difficult to define "artificial intelligence"?

Q2: Why is it difficult to define "intelligence"?

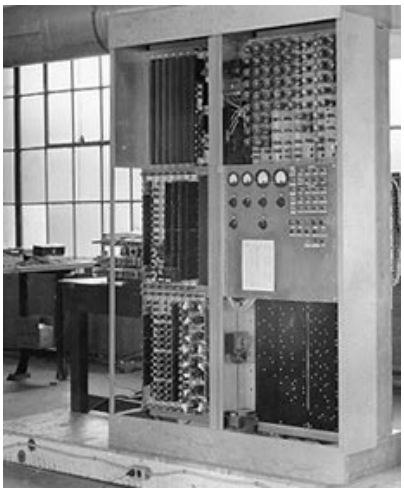
Q3: List 6 aspects of human intelligence, and give an example of each.

1.3 The Turing Test

Learning Objective

After studying this topic, you should be able to:

- describe the Turing Test;
- describe the weaknesses of the Turing Test.

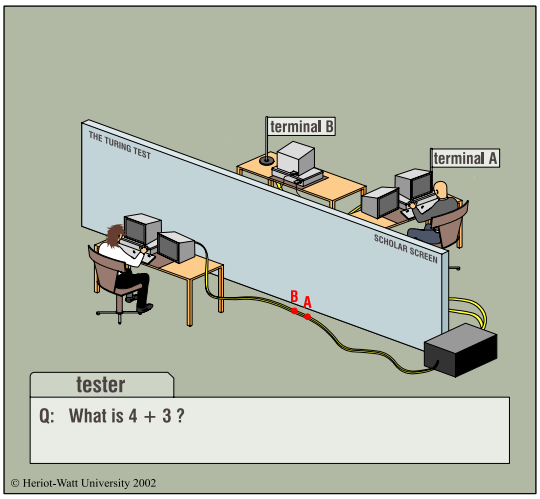


Artificial Intelligence dates back to the early 1950's. Some promotional videos from that era describe computers as "giant mechanical brains", although they could only carry out repetitive calculations, and showed none of the aspects of intelligence we have considered (creativity, language, vision, reasoning). However, some thinkers could see ahead that these early computers would develop into more "intelligent" machines.

One such thinker was **Alan Turing**, an English mathematician. In 1950, he invented a test which could be used to decide whether or not a machine (computer) could be said to be "intelligent".

His idea was very simple. The person carrying out the test sits at a **terminal** in one room. Using this terminal, the tester can communicate through the wall to two other terminals in another room. One of these terminals is being operated by another human. The other terminal is connected to the computer being tested for intelligence. The tester can communicate with both (but only through his terminal) by typing questions and receiving

responses. There is no limit to the question topics.



If the tester is unable to tell which is the human and which is the computer, the computer running the program would have passed the test and could be said to have displayed intelligence.

Turing test animation

There is an animation of the Turing Test on the course website.

Notice that passing the test would not show that the computer program **was** intelligent; it would only show that the program *displayed intelligent behaviour*. (Of course, this is also true of the person!)

There have been arguments ever since about whether or not the **Turing Test** is a valid one. However, it is regarded as an important stage in the development of Artificial Intelligence and a target which is yet to be achieved by any programmed machine.

Peter Ross of Edinburgh University sums it up like this: "AI is **not** about trying to produce artificial brains or humans, and nobody seriously works on trying to pass the infamous **Turing Test**. Instead, it is about creating smart artefacts, such as robots or computer programs, and about the scientific investigation of aspects of intelligence through modelling and hypothesis testing."

Sentence completion - Turing Test

5 min

There is a sentence completion task on the Turing Test on the course website.

Loebner prize

15 min

Although the Turing Test has serious flaws, it continues to act as a challenge to some AI researchers and programmers. One incentive is the Loebner Prize, which offers $150,000 to the first program to 'pass' the Turing Test. So far, no program has done so, although there have been many attempts. You can read about the Loebner Prize, and try out some of the unsuccessful attempts to win it, at this web address:

http://www.loebner.net/Prizef/loebner-prize.html

1.3.1 Review questions

Q4: Describe the Turing Test in your own words.

Q5: If a computer program passed the Turing Test, could it be said to be "intelligent"? Give one argument in favour of this, and one against.

1.4 Summary

- Human intelligence is very hard to define as it has so many aspects;
- Aspects of human intelligence include the ability to learn, creativity, reasoning and the ability to explain, use of language, vision, problem solving, adapting to new situations, cognitive ability, memory;
- There are many definitions of AI, because it is a rapidly changing area, it involves researchers from many backgrounds, and because human intelligence is not well defined;
- The Turing Test was first described in 1950 by Alan Turing;
- The purpose of the Turing Test is to determine whether or not a system can be described as intelligent;
- In the Turing Test, an operator uses a terminal to communicate with a system being tested by asking any questions and observing the responses. If unable to tell whether the responses are coming from a machine or a human, then the system being tested can be described as intelligent;
- No computer system has yet passed the Turing Test;
- In fact, the Turing Test can only be used to show that a system 'behaves' in an intelligent way.

1.5 End of Topic Test

An online assessment is provided to help you review this topic.

Topic 2

The Development of AI

Contents

Learning Objectives

After studying this topic, you should be able to:

- *Describe how AI began in the 1950s;*
- *Describe the development of game playing programs from simple early examples to contemporary complex examples displaying intelligence;*
- *Describe early attempts at language processing including Eliza and SHRDLU;*
- *Describe chatterbots and contemporary applications of language processing;*

- *Explain the need for knowledge representation techniques, including semantic nets and logic programming;*
- *Identify the two specialised languages used in AI: LISP (functional) and Prolog (decalrative/logic);*
- *Explain the change in emphasis from general AI towards domain-specific developments;*
- *Explain how faster processors, more memory and increased backing storage capacity have supported the development of AI;*
- *Describe the implementation and advantages of parallel processing.*

2.1 Introduction

In this topic, you will explore some of the early successes of AI, including simple game playing and language processing. These early successes led to a great deal of excitement among AI researchers, who believed that the goal of developing artificial intelligence could be achieved. However, it soon became apparent that things were not so simple, and there was a change in emphasis away from the grand picture towards setting and achieving more limited goals. At the same time, there was a realisation that new techniques were needed to deal with problems requiring the representation of large amounts of knowledge. New programming languages, including LISP and Prolog, were devised, which introduced new problem solving strategies very different from traditional algorithmic programming. Meanwhile, developments in hardware were making it possible to handle the complexity required for AI developments.

2.2 Game playing and other early developments

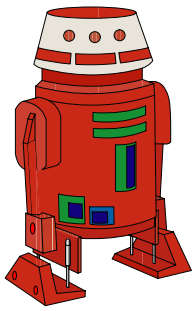
Learning Objective

After studying this topic, you should be able to:

- describe how AI began in the 1950s;
- describe the overall aim of early AI research;
- explain why game playing programs were one of the early successes of AI research.

2.2.1 How it all began

Following the end of World War II, the computer age began. There is debate about which was the first computer, but contenders include the German "Z3" (1941), the British "Colossus" (1943) and the American "Mark 1" (1944). The first electronic *programmable* computer is generally reckoned to be "ENIAC", built in 1946. ENIAC led to BINAC, and then to UNIVAC, the first commercial computer, in 1950, shown on the left.



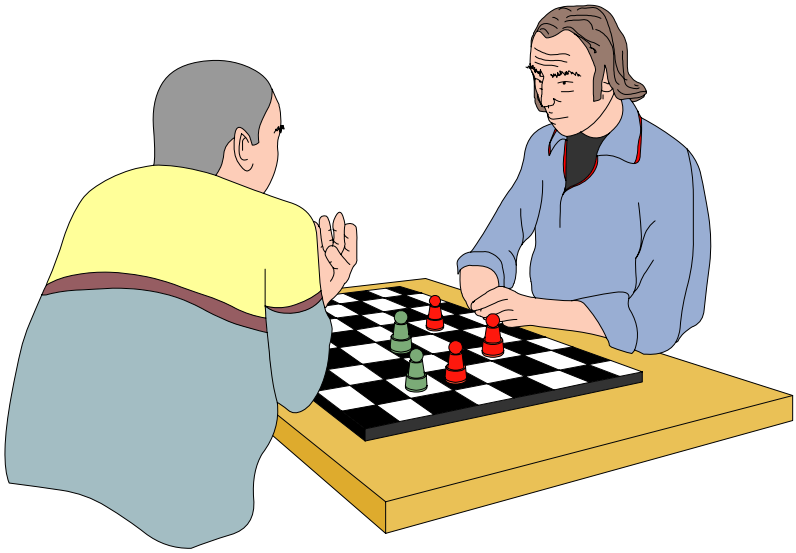
There was a great deal of excitement about these machines, and speculation about what they might be able to do. Some people described them as "electronic brains". Could they equal (or even surpass) humans in intelligence? Some believed the answer was "yes". Around this time, the scientist / writer **Isaac Asimov** wrote a series of books where the heroes were intelligent robots with "positronic brains". The first of these, called "I, Robot", was published in 1950, just when **Alan Turing** described his famous test.

Try to get hold of any of Asimov's Robot short stories or novels; apart from being entertaining to read, they raise many questions about machine intelligence which are worth considering.

In 1956, **John McCarthy** called a conference of people interested in this area of research. It was at this now famous "Dartmouth Conference" that the term Artificial Intelligence was first used. From this conference, researchers went out with the aim of developing computers and computer programs in the likeness of humans. Of course this was a huge and ambitious undertaking, so early research developed in smaller, specific domains.

2.2.2 Game playing

One area which gave some early signs of encouragement was in game playing. In 1956 a chess program called MANIAC-1 was the first to beat a human player (although it was to take another 40 years before chess programs could successfully take on the World Champion).



Deep Blue

You can read about the chess-playing program Deep Blue and its 1997 triumph over World Chess Champion Gary Kasparov at the following web address:

www.research.ibm.com/deepblue/

Apart from the results of 40 years of research into how to program games such as chess, one of the reasons for the great advances is the way that hardware has improved over time. Deep Blue runs on an RS/6000-SP supercomputer, with 6080 processors and 44Terabytes of storage! Such processing power was unheard of in the early days of AI!

Image reproduced with kind permission of Cray Inc.

2.2.3 AI in the 1960s

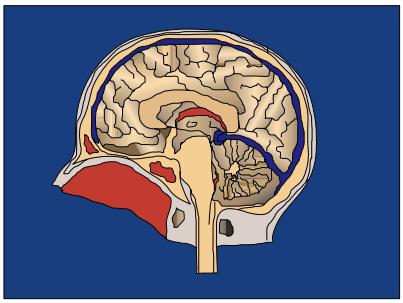
Programs were also developed to play checkers, draughts, noughts and crosses and other relatively simple and logical games. These were successful because the problem to be solved was in a limited *domain* - the problem was clearly defined, the rules were logical, and the number of possible actions at any time was limited.

These programs certainly showed some aspects of intelligence, but could not possibly pass the **Turing Test**. A chess program might be indistinguishable from a human playing chess, but if you asked it whether it was hungry or what colour the walls were painted, it would be unable to answer.

Other researchers developed programs to solve mathematical problems with some success, but once again these only operated in a limited domain. The grandly named "General Problem Solver" (GPS) written in 1957 could break down logical problems and apply logical reasoning, but would be unable to handle "real life" problems.

In the mid-1960's, Evans' Analogy program used the idea of "problem solving by analogy" to model one aspect of human reasoning. Although it was very successful in the limited domain of problems involving geometric shapes, it could not be applied to more general problem solving scenarios.

A completely different strand of research in these early days was into developing intelligence using a "**neural network**". This idea, based on the structure of the brain, had first been suggested in 1943. Research had shown that the brain was made up of millions of relatively simple **neuron**s all connected together - quite different from the design of the single processor **Von Neumann computer**.



Attempts were made to build neural networks during the 1950s, but the technology of the time was not really adequate. One example, called SNARC, used 3000 vacuum tubes to simulate a network of 40 neurons. The first real success in this area was **Frank Rosenblatt**'s Perceptron in 1959.

All of these early developments were held back by the relatively primitive technology of the day. Many ideas were being debated and developed, but few could be put into practice with much success. Compared to computer technology at the start of the 21st century, the computers of the 1950's and 60's were very slow, had much less memory, used tape storage, were very expensive, and only a small number existed in research institutions.

Programming techniques and languages were also much less sophisticated than they are now, and all input and output was text-based. Despite these hardware and software constraints, progress was made, but only in relatively limited domains such as game playing and mathematical logic. The "intelligent computer", able to pass the **Turing Test** was still a long way off!

2.2.4 Review questions

Q1: What 2 significant developments took place in 1950?

Q2: Why is 1956 considered the birth year of AI?

Q3: What was the overall aim of early AI researchers?

Q4: Why was games playing one of the first areas of successful AI development?

Q5: What aspects of intelligence are emulated by a game playing program?

Q6: Why could a chess program not pass the Turing test?

Q7: How does a neural network differ from a "von Neumann" computer?

Q8: What factors limited the success of early AI research?

2.3 Language Processing

Learning Objective

After studying this topic, you should be able to:

- state that the main fields of research during the period 1965 to 1975 were knowledge representation, machine understanding, and language;
- describe how Eliza holds a "conversation";
- name SHRDLU and Parry as 2 other language-based programs;
- state that a chatterbot is a program that simulates human conversation.

Early success in AI had been in game playing and other limited domains. There seemed to be no prospect of developing a system which demonstrated intelligent behaviour in wide enough fields to pass the **Turing Test**. To do so, a computer system would need to have access to a much wider knowledge base than simply the rules of chess. Critics were also emerging who argued that all the current systems were able to do was to follow rules mechanically. No matter how successful they were, the programs had no "understanding" of the problem.

This led to a new focus for research into the fields of knowledge representation, machine understanding and language.

To pass the **Turing Test**, a computer system would need to be able to "hold a conversation" in a natural way. That meant it would need to have a wide vocabulary, understand the (often illogical) rules of English grammar, and be able to respond to questions in a sensible way. One of the first programs which demonstrated some ability in this area was **Eliza**, written by **Joseph Weizenbaum** in 1966.

2.3.1 Eliza

Eliza was never meant to be able to hold a normal conversation. It was designed to simulate a non-directive therapist. This means that it reacts to your comments or questions in an open-ended way, and it is up to you to take the initiative in the conversation. Despite this, there are stories (maybe true, but probably exaggerated by the research students in the University Union bar on a Friday night) of Eliza having fooled Weizenbaum's company vice-president.

The vice president was trying to contact Weizenbaum using a terminal (a bit like modern day on-line chat). He dialled through to Weizenbaum's computer and stared typing his conversation; he thought he was "talking" to Weizenbaum, but it was actually Eliza at the other end. The famous conversation supposedly went like this:

V.P. Do you think I can use the computer this morning?

Eliza: Why do you ask?

V.P. I might be able to make some additional Telecomp sales.

Eliza: Why aren't you sure?

V.P. My prospective customers have never seen a demonstration of the system.

Eliza: Does his mean very much to you?

V.P. This is ridiculous.

Eliza: What do you mean, this is ridiculous?

V.P. Is there any reason why I can't use the computer now?

Eliza: Tell me more about you prospective customers..

The V.P. was getting fed up being stalled by "Weizenbaum's" evasive replies, so he phoned him up directly to give him an earful. The phone conversation began ..

V.P. Why are you being so snotty to me?

Weizenbaum: What do you mean, why am I being so snotty to you? (which is exactly the sort of response Eliza would have made..)

A conversation with Eliza

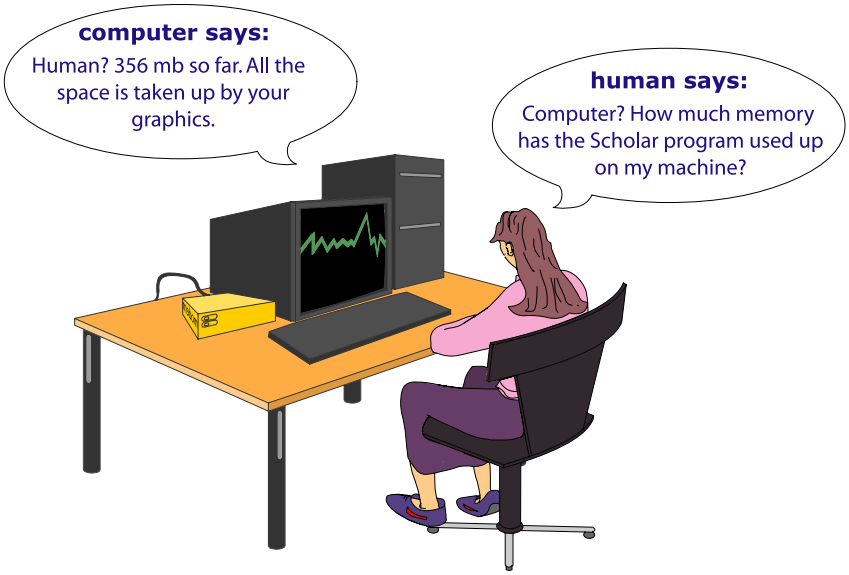
30 min

There are many examples of Eliza programs available to run on modern PCs. Your teacher may be able to let you use one; if not, a search on the internet should lead to some examples. For example, you will find a version of Eliza at:

www.uib.no/People/hhiso/eliza.html

Hold a "conversation" with Eliza.

Try to write down how Eliza seems to work.



Basically, **Eliza** scans your input for keywords, and replies to you (often with another question) using those keywords.

For example, if you say "I find Higher Computing difficult", Eliza might respond "Why do you find *Higher Computing difficult*", or "Does it worry you that you find *Higher Computing difficult*".

Eliza sometimes uses related keywords, so if you say "My *brother* is a pest", Eliza might respond "Do you have problems with your *family*?". If Eliza cannot "understand" what you say, it simply resorts to stock phrases like *"Tell me more".*

How does Eliza work?

Now try to answer these questions about Eliza. You might find it helpful to discuss the answers with another student before you look at the answers.

Q9: Does Eliza understand what you say to it? How can you tell?

Q10: Why do you think Weizenbaum chose to model a non-directive therapist rather than to model normal conversation?

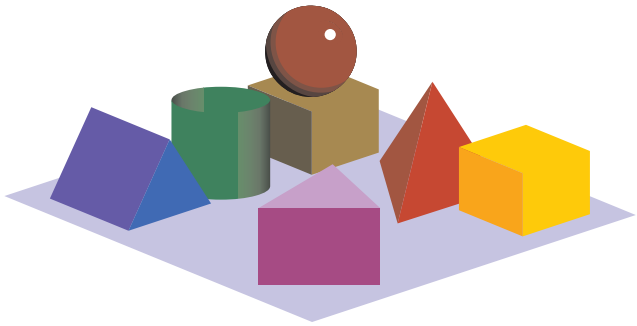
Q11: Does Eliza pass the Turing test?

2.3.2 SHRDLU and Parry

Other programs which could process language were also developed around this time. Two of the most famous ones are **SHRDLU** (1970) and **Parry** (1971).

Parry followed on from Eliza. Instead of modelling the conversation of the therapist, Parry was an attempt to model the conversations of a paranoid person. One reason for such an odd choice of domain was that AI researchers are not simply computer scientists as we know them, but include psychologists and other experts in human behaviour.

SHRDLU was developed by **Terry Winograd** and could respond well to quite complex sentences about a block world, consisting of coloured blocks or various shapes:



SHRDLU could respond to sentences like "Move the red block behind the green one". It could "understand" that "the green one" meant the green block which had been referred to in an earlier sentence. It could respond in an intelligent way to concepts like behind, on top of and in front of. However, if you asked it to move the car to the end of the drive, it would be unable to respond. SHRDLU's world only consisted of coloured blocks. It operated in a very limited domain. However, it was able to show some kind of understanding of its limited domain, and did not simple respond mechanically and without understanding as **Eliza** had done. This was because **SHRDLU** had a system for *representing the knowledge* it had about its domain.

A great deal of research has gone into language processing in the 30 or more years since SHRDLU appeared on the scene. You will study this in more detail in Topic 3.3.

Much of this is serious research, but one of the spin-offs has been a huge number of programs called 'chatterbots' which simulate human conversation. Some of them are not much more sophisticated than Eliza, except that modern computers are faster and have more memory than the computers which Weizenbaum used around 1970. As a result, these chatterbots can attempt to hold a conversation much more successfully that Eliza could. The more successful ones use knowledge representation techniques which allow a degree of 'understanding' rather than simply reacting to input.

Chatterbots

30 min

Explore some comtemporary **Chatterbots**.

Try some of the following web addresses:

www.simonlaven.com (has links to many chatterbots)

www.abenteuermedien.de/jabberwock/index.php (Jabberwock is a good example of an on-line chatterbot - in both English and German !)

www.alicebot.org (another well known 'bot')

Chatterbots like these are mainly for fun, or have been developed as research projects. However, there are potential applications in making software interfaces more user-friendly. Compare, for example, Ask Jeeves (www.ask.com) with Google (www.google.com).

2.3.3 Review questions

Q12: Name 3 programs from around 1970 which showed some language ability.

Q13: What was SHRDLU's domain?

Q14: In what way was SHRDLU more "intelligent" than Eliza or Parry?

Q15: Name one software advance and one hardware advance that allowed AI research to progress during the early 1970s.

2.4 Knowledge Representation Techniques

Learning Objective

After studying this toipc you should be able to:

- state that LISP and Prolog are two languages specially developed for use in AI applications;
- state that LISP is a functional language;
- state that Prolog is a logic/declarative language;
- describe the main differences between a declarative language (Prolog) and an imperative language (Pascal, BASIC).

Early developments, whether in games playing programs or chatterbots, were limited by the programming techniques available at the time. The only tools available to researchers were standard programming languages. These were designed for coding algorithms to solve a problem. Complex games and human conversations cannot easily be represented by an algorithm. To be successful, they have to be able to manipulate large amounts of knowledge, often in an unpredictable way.

Much early AI research was into ways of representing knowledge. Among the most successful of these were semantic nets, scripts and frames. You will use semantic nets in Topic 8.

This ability to represent knowledge was seen as the key to further success in AI. Marvin Minsky's 1974 publication 'A framework for representing knowledge' was a crucial moment for AI. Around this time, the computer language **Prolog** was invented. Along with the earlier language **LISP**, this gave researchers the tool required for knowledge representation.

2.4.1 LISP

LISP is known as a functional language. Just to give you a flavour of LISP, here is part of an ELIZA program, written in Common LISP:

```
(defun segment-match (pattern input bindings &optional (start 0))
  "Match the segment pattern ((?* var) . pat) against input."
  (let ((var (second (first pattern)))
        (pat (rest pattern)))
    (if (null pat)
        (match-variable var input bindings)
```

```
;; We assume that pat starts with a constant
;; In other words, a pattern can't have 2 consecutive vars
(let ((pos (position (first pat) input
                     :start start :test #'equal)))
  (if (null pos)
      fail
      (let ((b2 (pat-match pat (subseq input pos) bindings)))
        ;; If this match failed, try another longer one
        ;; If it worked, check that the variables match
        (if (eq b2 fail)
            (segment-match pattern input bindings (+ pos 1))
            (match-variable var (subseq input 0 pos) b2))))))))
```

2.4.2 Prolog

Prolog is a logic language.

Prolog can also be classified as a **declarative** language. In a Prolog program, the key task of the programmer is to find a way of 'declaring' precisely all the facts and relationships which describe the situation being tackled. This is in contrast to imperative languages, like BASIC and Pascal, where the programmer codes the instructions that the computer must carry out.

Here is some Prolog code:

```
/* animal.pro
  animal identification game.

    start with ?- go.     */

go :- hypothesize(Animal),
      write('I guess that the animal is: '),
      write(Animal),
      nl,
      undo.

/* hypotheses to be tested */
hypothesize(cheetah)   :- cheetah, !.
hypothesize(tiger)     :- tiger, !.
hypothesize(giraffe)   :- giraffe, !.
hypothesize(zebra)     :- zebra, !.
hypothesize(unknown).             /* no diagnosis */

/* animal identification rules */
cheetah :- mammal,
           carnivore,
           verify(has_tawny_color),
           verify(has_dark_spots).
```

You have probably studied an imperative language in the Software Development unit of this course, and you will study Prolog later in this unit, in Topics 7 and 8.

2.5 Domain Specific Developments

Learning Objective

After studying this topic, you should be able to:

- state that "domain specific developments" have been the main focus of research since 1975;
- describe some "domain specific developments";

During AI's first decade (up to 1965), there was much optimism as researchers aimed at the goal of developing computer systems which would behave intelligently. Early success were in areas like game playing, but it soon became apparent that "complete" AI was going to be a huge problem. The problem was much more complex than expected, and the necessary hardware and software tools did not yet exist.

In the next 10 years, up to 1975, research focussed on the tools needed to represent knowledge and to allow programs to understand, rather than blindly following set rules. AI became recognised as a valid area of research, with funding becoming available and research centres set up in prestigious universities such as MIT, Stanford and Edinburgh.

However, there was increasing debate about whether or not AI was philosophically possible, and some of the early optimism was lost. It was time now to show that AI could bring real-world benefits. Otherwise, research funding would dry up. Researchers therefore started to focus on limited domains where a solution might be possible rather than on the global goal of developing a generally intelligent machine.

The most successful of these was the development of **Expert Systems**. Using the result of earlier research, and the development of knowledge processing languages such as **Prolog**, it was possible to encode expert knowledge on specific and limited areas. The system could then give advice which was as good as, or better than, the advice given by human experts.

Early success included **MYCIN** (diagnosing blood infections) in 1979, **XCON** (configuring mainframe computers) in 1980 and **Prospector** (identifying sites for mineral extraction) in 1982. The 1982 Alvey report from the UK government gave a further boost in this field. Since then a huge variety of expert systems have been developed for use in medical, commercial and military applications.

Although **expert systems** can claim the greatest commercial success, many other strands of AI research have developed. Researchers have specialised into relatively small and specific areas (domains) where there is some possibility of success. Perhaps eventually, all these will be able to be fitted together to achieve the original goal of a machine "in the likeness of a human".

These domain specific developments include:

- natural language processing;
- computer vision;
- pattern matching;
- intelligent robots;

- intelligent tutoring systems;
- neural networks;
- fuzzy logic systems.

Since 1975, the development of AI has continued in this domain-specific way. As a result it has become a very diverse field of research. There have been a wide range of very successful developments in these limited domains, but we are still nowhere near developing an 'artificial human'.

Identifying AI developments

10 min

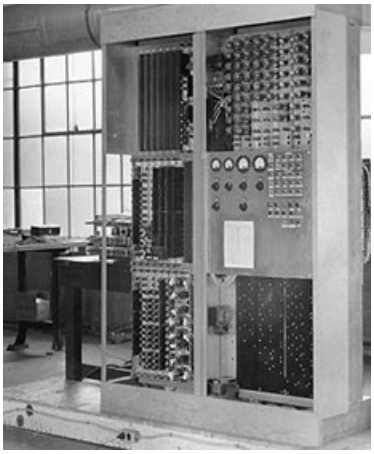
On the Web is a an activity which requires you to identify AI developments within the correct phases of the history of AI. You should now complete this activity.

2.6 Hardware Developments

Learning Objective

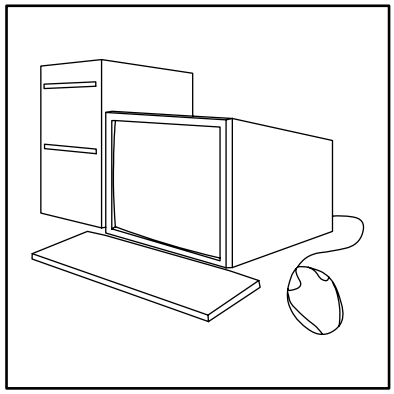
After studying this topic you will be able to:

- describe how developments in hardware and software have increased the success of AI research since 1975.



As we have already seen, the types of hardware available to researchers has changed dramatically since the early days of AI. Alan Turing could not have imagined the number and power of modern day computers.

Consider the following quote from Popular Mechanics magazine in March 1949: *"Where a computer like the ENIAC(1939) is equipped with 18,000 vacuum tubes and weighs 30 tons, computers in the future may have only 1,000 vacuum tubes and weigh only 1 1/2 tons."*



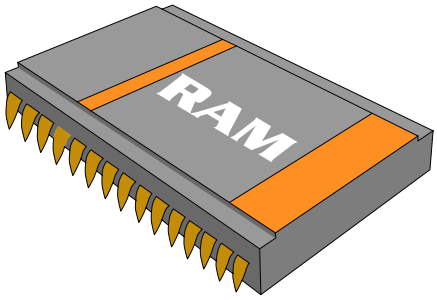
The computer you are using probably has a processor consisting of at least 42 million transistors (the modern equivalent of vacuum tubes), and certainly weights much less than 1.5 tons!

As we have already seen, faster processors and increased memory and storage capacity, allow modern AI applications to be much more sophisticated than in the early days. A modern chess-playing program can search through many times more possible moves per second than early programs. Similarly, a modern-day Eliza can store a much larger vocabulary than Weizenbaum's original.

30 min

Hardware and software issues

Write a list of major developments since 1975 which you think have contributed to the success of AI.



You may have listed some of the following:

- introduction of personal computers;
- the invention of the GUI;
- huge increases in processor speed;
- huge increases in processor capability;
- development of magnetic discs;
- development of optical discs;
- computer networking becomes possible;
- the Internet;
- improvements in memory chips means more memory at less cost;
- many new programming languages developed.

For each of the above development, can you link it to progress in AI?

For example, the introduction of PCs meant that increasingly researchers had instant access to computers rather than having to wait for access to a departmental mainframe through a **terminal**.

2.7 Parallel processing

Learning Objective

After studying this topic, you should be able to describe:

- developments in parallel processing;
- applications of parallel processing;
- hardware and software issues related to parallel processing;
- philosophical, ethical, moral and legal issues related to parallel processing.

2.7.1 What is parallel processing?

As you have discovered so far in your study of Artificial Intelligence, there are many problems which require the processing of vast amounts of data, and so which require very fast processors if they are to generate results in a reasonable length of time. One current development which helps overcome this type of problem is **parallel processing**.

In **parallel processing**, a task is divided into many sub-tasks which can then be processed simultaneously on different processors.

For example, consider a chess-playing program. At any stage in the game, there will be many different moves which could be made. The program must analyse each of the possible moves and then decide which is the best one to make. On a conventional computer, each move must be considered in turn, one after the other, until all have been investigated. Using parallel processing, each possible move can be allocated to a different processor, and all evaluated at the same time. Clearly this will speed up the program considerably.

Parallel processing can be applied to any problem which can be divided up into sub-tasks in this way. Care must be taken to ensure that the sub-tasks are independent of each other, or the benefits will be lost. Specialised programming languages (e.g. Parlog and Occam) have been invented to aid the development of parallel solutions to AI problems.

Parallel processing is particularly useful in **search-based problem solving** (see Topic 3).

2.7.2 Applications of parallel processing

Parallel processing can be applied to all the areas of AI that we have considered so far, and many more.

In **natural language processing**, as we have seen, problems arise through ambiguity and the multiple meanings of words. Parallel processing in this situation would allow the software to explore several words in a sentence at the same time, or to consider each of several meanings for a word simultaneously.

Expert systems (see Topic 2.2 and Topic 5 for more detail) are based on hundreds of rules which have been programmed into them by a **knowledge engineer**. A rule may be of the form:

IF patient has a history of heart disease AND patient is over 45 years old AND patient has high blood pressure THEN carry out the Mosolowski-Anderson test.

There are three conditions in the rule; each of these may require considering another rule with multiple conditions. In a **Von Neumann computer** system, each condition has to be evaluated in turn. Parallel processing would allow the expert system's **inference engine** to process all three (or more) conditions at the same time.

Computer vision depends on the processing of information extracted from images consisting of thousands of pixels, represented by digital codes. Clearly any processing of this data can be speeded up by processing many pixels, or groups of pixels, at the same time using parallel processing. This can be extended to any type of **pattern matching**, whether it is finding patterns in visual data, sound or simply complex statistical or financial data. Parallel processing is also used in large scale simulations, complex realtime calculations and virtual reality systems.

Perhaps one of the most distinctive features of human intelligence is our ability to do several thing simultaneously. Parallel processing provides computers with the ability to do this. It is clear that even though processor speeds and available memory is still increasing rapidly on all computer systems, the vast amounts of data required in most real-world AI problems can only be tackled successfully by multiple processors.

The most extreme example of parallel processing is the **neural network** (see Topic 2.6), where hundreds or thousands of very simple processors work together to process fuzzy data in an intelligent way.

2.7.3 To find out more about parallel processing

Web sites

- www.epcc.ed.ac.uk (Edinburgh University's parallel computing site)
- www.cray.com/supercomputing (web site for Cray, manufacturers of parallel computers)

Image reproduced with kind permission of Cray Inc.

20 min

Your own summary notes on parallel processing

Create your own brief summary notes on parallel processing.

Make sure you can describe:

- what parallel processing means;
- why it is useful in AI developments;
- some successful applications.

2.8 Summary

- The first computers were developed in the 1940s;
- Alan Turing described his famous test for AI in 1950;
- The development of AI began with the 'Dartmouth Conference' in 1956;
- Games playing programs (checkers, draughts, chess) were an early success of AI research;
- Early games playing programs were simple rule-based systems, and were not really 'intelligent';
- The development of faster computers, larger memory capacities and new programming techniques have led to the development of much more effective and 'intelligent' games playing programs;
- Early attempts at language processing including Eliza and SHRDLU;
- Eliza was not intelligent - it simply reflected back the user's input;
- Chatterbots have developed from Eliza, and have much larger vocabularies, and mechanisms for 'understanding' user input;
- Chatterbots can be used to give programs a much more user-friendly interface;
- Knowledge representation techniques, including semantic nets and logic programming, were developed to make it possible to develop programs which incorporated large amounts of knowledge;
- Two specialised languages are used in AI: LISP (functional) and Prolog (declarative/logic);
- AI research has made great progress by focussing on domain-specific developments rather than on attempting to create 'general' intelligence;
- Domain specific developments include natural language processing, computer vision, intelligent robots, neural networks and expert systems;
- Faster processors, more memory and increased backing storage capacity have supported the development of AI;
- Parallel processing (using many simple interconnected processors) has many applications in AI.

2.9 End of Topic Test

An online assessment is provided to help you review this topic.

Topic 3

Vision, language and movement

Contents

Prerequisite knowledge

Before studying this topic, you should be able to:

- *Describe some applications of computer vision systems (including industrial and military uses and interpreting satellite photos);*
- *Describe some applications of speech recognition (including speech driven word processors, disabled users, military uses, controlling cars and mobile phones);*
- *Describe some applications of dumb and intelligent robots.*

Learning Objectives

After studying this topic, you should be able to:

- *State some arguments for and against "strong AI";*
- *Describe the main stages of a computer vision system;*
- *Describe the problems of interpreting 2D images of 3D objects;*
- *Identify the main stages of natural language processing (NLP);*
- *Explain some of the difficulties involved in NLP;*
- *Describe some applications of NLP;*
- *Explain the difference between dumb and intelligent robots;*
- *Describe some of the practical difficulties of implementing intelligent robots and strategies to overcome these;*
- *Describe some contemporary research and developments in the field of intelligent robotics;*
- *Describe some possible social and legal implications of intelligent robots;*
- *Describe some uses of intelligent software in embedded control systems.*

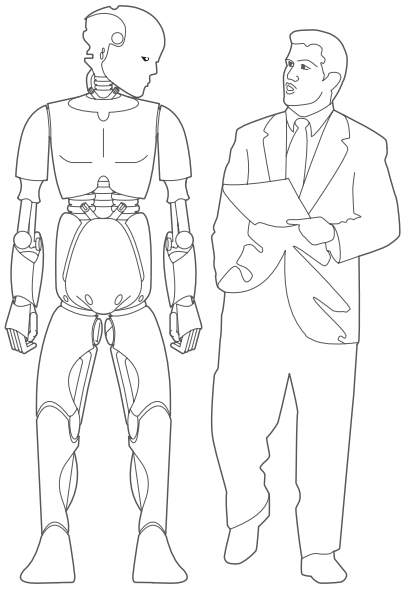
3.1 Introduction

In the first two Topics, we have explored the philosophical ideas that lie behind Artificial Intelligence, and then discovered how AI has developed from small (but ambitious) beginnings some 50 years ago into a diverse (but realistic) area of research and devlopment at the start of the 21st century.

We have seen how progress has been made possible by the massive changes in computer hardware over the last 50 years, in terms of faster processors, more memory and storage capacity, and simply the increased availability and affordability of computing power to researchers.

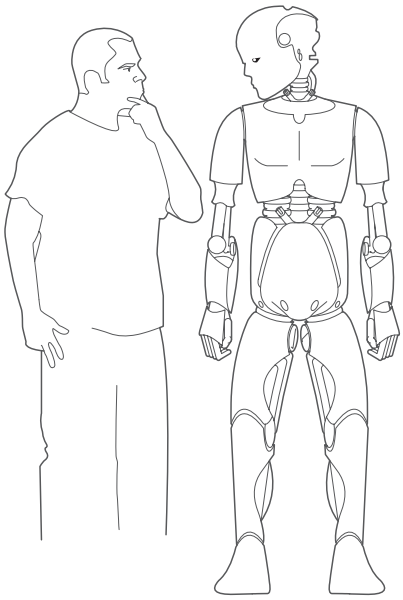
Perhaps we should now ask the question, "Given the huge progress which has been made in a wide range of limited domains, and the seemingly never-ending increase in computing power, will it ever be possible for the ambitious early vision of AI to become reality?"

This question is really a philosophical one rather than a technological one, and argument can be put on both sides.



Some would say that there is no limit to what can be achieved. If hardware continues to advance as it has done over the last 4 decades, and researchers continue to develop ever more sophisticated systems, then there is no reason why it should not be possible to develop a system that displays complete artificial intelligence. The humanoid robots of science fiction could become a reality within our lifetimes. It should be possible to develop machines which would pass the Turing test. These devices could then bring great benefit to the human race. This view is known as "Strong AI".

Others argue that no matter how much progress is made in hardware or software techniques, artificial intelligence is philosophically impossible and undesirable. Even if it proved possible to develop a humanoid robot, it would still only be a robot. It would only appear to be intelligent. It could not be conscious, self-aware or self-motivated, as humans are. Rather than pursuing a meaningless dream, these researchers would argue that it is more sensible to continue to develop within existing limited domains.



No doubt you have your own views.

20 min

Optional: exploring strong AI

If you want to explore these issue further, there are plenty of resources on the web, for example:

- www.compapp.dcu.ie/~humphrys/newsci.html

  (an article entitled "AI is possible .. but AI won't happen: The future of Artificial Intelligence" by Mark Humphrys of Dublin City University, School of Computing - however, note that it was written in 1997, and things have moved on a bit from there)

- www.comp.glam.ac.uk/pages/staff/efurse/Theology-of-Robots/A-Theology-of-Robots.html

  (an article by Edmund Furse of the University of Glamorgan, giving detailed arguments for and against "Strong AI")

- www.alanturing.net/turing_archive/pages/Reference%20Articles/what_is_AI/What%20is%20AI13.ht

  (a more recent article entitled "Is Strong AI possible?")

- and finally, www.iep.utm.edu/a/artintel.htm

  (a 1 page article form "The Internet Encyclopedia of Philosophy").

15 min

Strong AI

If possible, work with a partner on this challenge.

Write down

a) 2 arguments in favour of "Strong AI", the belief that artificial intelligent machines will one day be possible;

b) 2 arguments against "Strong AI", explaining why artificial intelligent machines will never be possible.

The verdict remains open!

In this rest of this Topic, we will examine what progress has been made in 4 important domains within AI. After you have studied these sub-topics, your views about "strong AI" may (or may not) have changed.

3.2 Vision systems

Learning Objective

After studying this topic, you should be able to describe:

- developments in computer vision;
- applications of computer vision;
- hardware and software issues related to computer vision;
- philosophical, ethical, moral and legal issues related to computer vision.

3.2.1 What is computer vision?

Computer vision is one of the hardest areas of AI. The aim is to develop systems which can see, make sense of what they see, and react accordingly. This is something which we human beings are very good at, but which is extraordinarily difficult to achieve in a computer system. A light sensor attached to a robot arm to detect the presence of a piece of metal which it is to paint could be described as computer vision, but such systems have been around for many years in industry, and don't really count. By vision systems we mean the attempt to develop the same level of sophistication and flexibility as the human eye and brain.

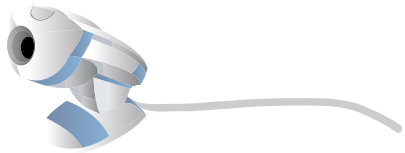
Essentially, computer vision breaks down into a 5 stage process:

1. image acquisition (digitisation)
2. signal processing
3. edge detection
4. object recognition (pattern matching)
5. image understanding

Sequencing the activities involved in computer vision

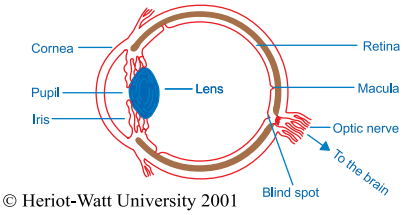
5 min

On the Web is a an activity which requires you to order correctly the activities involved in computer vision. You should now complete this activity.



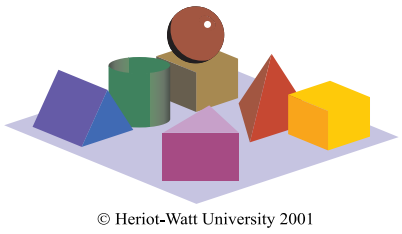
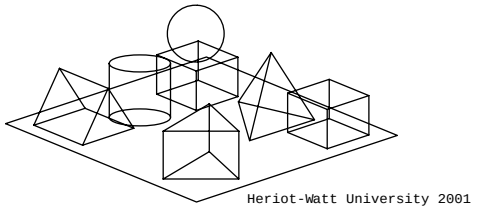
The first stage is **image acquisition**. This is a technical problem which has been solved, and so would not count as part of AI. Digital cameras (both still and video) and scanners are now available which can take any image and convert it into a stream of digital data. However, these systems have only recently developed to the level of accuracy which is required, and become available at affordable prices.

The second stage is **image processing**. Again, techniques are well developed for enhancing digital images in various ways, including improving the resolution, removing distortion caused by the optical system in the camera, removing dust and scratch marks, and improving contrast or changing overall brightness of an image. These techniques are not AI, but the development of the software required was necessary before real vision systems could be achieved.

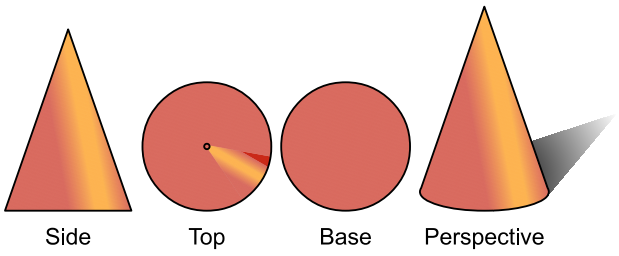


The third stage is where the AI involvement begins. When you look at a scene (either real life or a photograph), an image is formed on the retina at the back of your eye. Your brain has to make sense of this image. It looks for features which help to identify what is in the scene. The most important features are **edges**. An edge is where there is a sudden change of colour in an image, and this is usually (although not always) where one object finishes and another begins. By identifying edges, the system can start to make sense of the raw digital data.

There are a number of methods which have been developed to transfer a photographic image into a **primal sketch** as shown below. One of these is called the Waltz algorithm.

photograph	primal sketch
	

The next task is to recognise the shapes as objects. This can be done by **pattern matching**, i.e. comparing the objects which have been identified with known objects. In the example above, an artificially simple image has been chosen. It is a block world, like the one which **SHRDLU** was able to describe. Pattern matching to determine which block shapes are in the image is relatively simple, although even here, any given shape can look quite different when viewed from different angles. Look how different the cone shape looks when viewed in different ways:



In a real world image, like this photograph of Montmartre from inside the Modern Art Museum in Paris, the identification of shapes and objects is much more complex. However, your eyes and brain can do this task very easily. It must be a soluble problem!

The final, and most difficult stage is **image understanding**. How do all the objects which have been identified relate to each other? What do they mean? What is happening out there? Are some of the objects moving? Are some of the objects more important than others? What action needs to be taken?

3.2.2 Applications of computer vision

Computer vision is being used increasingly is security systems and industrial processes.

Security systems using computer vision can recognise individual faces, and allow or deny access.

Vision systems are now available for industrial robots. This makes them much more flexible as they can identify objects that they are working on, and be programmed to respond accordingly.

3.2.3 Is computer vision really AI?

Why is Computer Vision considered to be a branch of AI? Which aspects of human intelligence does it model?

- **ability to learn :** YES - some systems are able to improve their recognition capability with time - the more "trees" they see examples of, the more likely they are to recognise another "tree" in the future;
- **use of language :** NO - although pattern recognition is used in OCR applications to recognise printer letters and symbols;

- **vision :** obviously YES!

- **problem solving :** NO - although the purpose of vision systems is to make it possible for computer systems to operate in problem-solving environments, independently of humans;

- **adapting to new situations :** YES - the ideal vision system would be able to cope with completely unfamiliar scenes, just as the human eye and brain can.

3.2.4 Hardware requirements of Computer Vision

Computer vision systems are very demanding of computer resources. As you know from your study of Computer Systems, digital images can take up vast amounts of memory and backing store. The processing of a digital image requires the system to manipulate millions of bits (representing pixels), using complex algorithms, and matching sections of the image to banks of stored images. This obviously requires a great deal of processing power. Until recently, this was only possible using large institutional mainframe computers.

However, traditional **Von Neumann computer** systems are not the ideal for implementing artificial vision. Human vision works using eyeballs containing thousands of light sensitive cells, which send signals **in parallel** to thousands of nerve cells called **neuron**s in the brain. Most successful vision systems use a similar model, with many processors being used simultaneously to process the image. This approach is called **parallel processing** and is dealt with in more detail in topic 2.7. One particular form of parallel processing which is being very successful applied to pattern matching and vision systems is called **neural network**s which are covered in more detail in topic 4.

3.2.5 To find out more about computer vision

Web sites

- www-2.cs.cmu.edu/afs/cs/project/cil/ftp/html/vision.html

  (The "vision home page" with links to dozens of vision-related sites - some of which are useful and some not so useful!)

- cvs.anu.edu.au/bioroboticvision/

  (various examples of vision systems used in robotics, including a helicopter guidance system)

- http://mi.eng.cam.ac.uk/research/speech/

  (vision research projects at Cambridge University)

20 min

Your own summary notes on computer vision

Create your own brief summary notes on computer vision.

Make sure you can describe:

- what computer vision aims to achieve
- some of the problems

- some successful applications
- any issues arising from the use of computer vision

3.3 Natural language processing

Learning Objective

After studying this topic, you should be able to describe:

- the 4 main stages of NLP;
- developments in NLP;
- applications of NLP;
- hardware and software issues related to NLP;
- philosophical, ethical, moral and legal issues related to NLP.

3.3.1 What is NLP?

Computer systems are designed to use "formal languages"; these include high level programming languages (e.g. Pascal, C++, BASIC) and low level machine languages. **NLP** is concerned with developing systems which can cope with "natural languages", like English, French or Serbo-Croat.

Formal languages are much more straightforward - they have a small vocabulary, the meanings of words are unambiguous, and the rules (syntax) of the language are defined very precisely. Natural languages are much harder to work with - they have a huge vocabulary (maybe 1 million words in English), the vocabulary is always changing, there are regional variations, words can have multiple meanings, and the rules of grammar are often illogical, imprecise, and often broken!

Questions

Q1: Copy and complete this simple table highlighting the main differences between formal languages and natural languages:

	formal language	natural language
example		English
vocabulary		millions of words
meanings of word	clearly defined and unique	
grammar / syntax rules		illogical, imprecise, often broken
development	does not change	

Q2: Write down an example of each of these problems associated with "natural languages":

1. an English word which didn't exist 10 years ago
2. a word from your local dialect which would be unknown somewhere else
3. an English word that can have several meanings
4. a rule of grammar than can be broken
5. two English words that sound the same

3.3.2 The stages of NLP

Natural Language Processing covers many different areas of research. If the aim is to develop a computer system that can take spoken input, and give spoken output, there are several stages, each of which is a major problem to be solved. The system will need to be able to

- input sound, and use sampling techniques to convert the sound into digital data
- analyse this data, and split it up into recognisable sounds (**phonemes**)
- figure out which of these sounds go together to form words
- identify how these words fit together into meaningful phrases and sentences
- "understand" the meaning of these sentences
- generate an appropriate response (a statement, a question, or an action to be carried out)
- convert the response into intelligible sound output

5 min

Ordering the stages of natural language processing

On the Web is a an activity which requires you to order correctly the stages involved in natural language processing. You should now complete this activity.

Stage 1: converting sound into digital data

The first of these steps is a manageable technical problem. Using a microphone and **digital sampling**, it is straightforward to convert sound into digital data using current technology.

Stage 2: splitting sound into phonemes

The next stage is not so easy. If you listen to someone speaking, they don't separate words from each other; sounds and words run together into a continuous stream of sound. The software has to be able to break this continuous stream into a series of recognisable sound patterns, called phonemes. English (and most other western languages) can be broken down into around 50 different **phonemes**.

Stage 3: identifying words

The software must next recognise how the phonemes can be grouped together into words. This is not easy. For example, the phrase "new display" and "nudist play" are almost impossible to distinguish by their sound. If we heard these sounds in a conversation, we would either use the context to decide which was the appropriate meaning, or ask the speaker to clarify which they meant.

Stage 4: extracting meaning

To proceed further, the software must have some method of extracting meaning from the list of words. This is the real crux of NLP, and applies to systems designed for handling text input as well as those which use speech input. In this area of research, AI overlaps with **linguistics** and **psychology**. Many different techniques are used to identify the structure and meaning of the input, and to overcome the ambiguous nature of natural language.

Q3:

Here are some examples of sentences which are ambiguous in meaning:

1. *"I saw a man on the hill with binoculars".*
2. *"The teacher shouted at Gary. He was often very aggressive".*
3. *"I know more politicians than Tony Blair".*

Can you suggest at least 2 meaning for each example?

Q4:

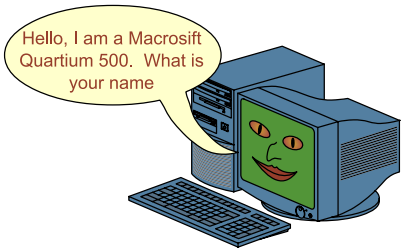
And here are some examples of sentences which seem to be wrong grammar when you read them first, but can make sense if you analyse them differently:

"Fat people eat accumulates".

"The man who hunts ducks out on weekends".

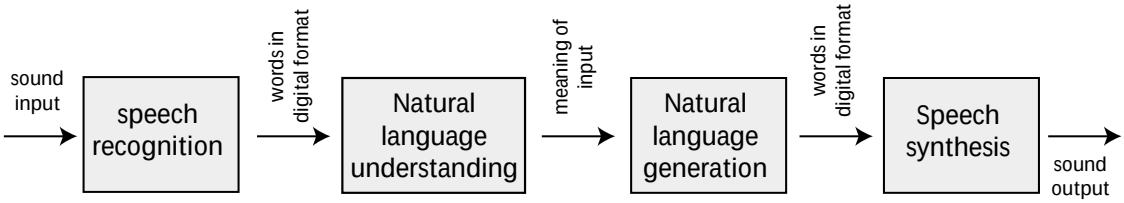
Can you make sense of each example?

Stage 5: generating a response



Finally, having made sense of the input in some way, an **NLP** system must make an appropriate response. This might be activating some output device, or it might involve a spoken response to the user. A spoken response requires at least 2 stages of processing. Firstly the system must decide on an appropriate and meaningful response (this is the difficult part), and then it must convert that response into intelligible and audible speech. Early systems sounded very artificial, but the quality of speech output is improving rapidly. The system must not only "pronounce" words correctly, but run them together in a way which sounds natural and uses intonation which is appropriate to the context.

Summary of Stages of NLP



3.3.3 Applications of NLP

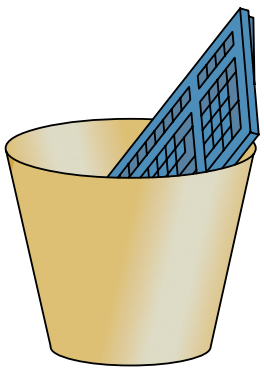
- **Machine translation**

  Systems have been developed to translate written text from one natural language to another. Some of these simply do a word for word translation which is not particularly helpful, but others extract meaning from the input in one language, and then build up equivalent phrases in the target language. The greatest successes have been in restricted domains such as the Meteo system which can translate weather forecasts from English into French.

- **Database query systems**

  Using the correct syntax to query a database can be complex and slow. Software is being developed which allows queries to be made in "natural language".

- **Speech-driven word processors**



IBM ViaVoice and Dragon NaturallySpeaking are two current systems which allow the user to dictate speech directly into a word processor (or other software). Early systems required the user to speak very clearly and slowly, but modern systems can be trained to recognise a user's voice, and can cope with fairly useful speeds (over 100 words per minute).

- **Command and control systems**

  ViaVoice and Dragon NaturallySpeaking can also be used to give commands to a computer, such as saving a document, switching between applications and so on. Another application, called Game Commander, is designed to provide voice control to many computer games. CommandTalk is a voice command system for military simulation software. Current improvements in systems like these suggest that voice input may replace the keyboard as the standard within the foreseeable future.

- **Natural Language Search Engines**

  Standard Internet search engines (like Google or Altavista) require the user to input search criteria in a structured Boolean format, like +perth -australia. This way of specifying the search parameters is very precise, but difficult for the average Internet user. Search engines like askJeeves (www.ask.com) allow the user simply to type in a question in natural language, like "I want to know about Perth, but not the one in Australia". The search engine uses NLP to identify the key words in the question, and presents the user with an appropriate list of web sites.

3.3.4 Is NLP really AI?

Why is NLP considered to be a branch of AI? Which aspects of human intelligence does it model?

- **ability to learn** : YES - some systems can improve their ability to recognise speech input as they are used
- **creativity** : YES - constructing sentences and responses to user input;
- **use of language** : YES - both "understanding" input, and generating output;
- **adapting to new situations** : POSSIBLY - a system may be able to adapt to a new speaker.

3.3.5 Hardware issues of NLP

Much early research was into the theoretical aspects of natural language systems, but practical applications were very limited by the hardware available (e.g. Eliza, SHRDLU and Parry). NLP requires

- fast processors - to be able to handle the huge amounts of input data at useful speeds
- large amounts of RAM - to be able to store the input data, and the vast amounts of rules and meanings of words required to interpret speech input.

Both processor speeds and amount of RAM are increasing rapidly, so it should be possible for the improvements in NLP to continue.

3.3.6 Philosophical, moral, ethical and legal issues of NLP

As NLP is so closely related to **linguistics** and **psychology**, research in this area certainly raises some interesting philosophical issues. The relationship between

language and intelligence is a topic of debate amongst researchers - is it our ability to use language that separates us from the rest of the animal kingdom?

NLP developments have certainly brought increased access to computers and their benefits for people who are blind, or who are unable to use a keyboard. On the other hand, in military applications, it could be argued that voice command systems could lead to more dangerous weapon systems.

As reliance increases on machine translation, legal issues may arise where mistranslations lead to problems of a commercial nature.

3.3.7 Where to find out more about NLP

Paper-based resources

- Advanced Higher Information Systems Natural Language Processing support materials, published by Learning and Teaching Scotland (LTS), 2001
- Higher Computing Artificial Intelligence support materials, published by Learning and Teaching Scotland (LTS), 1998

Web sites

- www.stanford.edu/group/SHR/4-2/text/dialogues.html
  (amusing conversations involving Eliza, Parry and Racter, three early NLP programs)
- www.cs.bham.ac.uk/~pxc/nlpa/nlpgloss.html
  (a glossary of NLP terminology)
- http://mi.eng.cam.ac.uk/research/speech/
  (some speech research projects at Cambridge University)
- http://babelfish.altavista.com/tr
  www.freetranslation.com/
  (websites offering a machine translation service)
- www.lhsl.com/naturallyspeaking/
  www-306.ibm.com/software/voice/viavoice
  www.gamecommander.com
  (information about ViaVoice, Dragon and Game Commander)
- www.uib.no/People/hhiso/eliza.html
  www-cgi.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/classics/eliza/0.html
  (some versions of Eliza)
- www.lhsl.com/realspeak/demo
  (a speech synthesis demo)
- www.research.microsoft.com/research/nlp/

www.cogs.susx.ac.uk/lab/nlp
(two general sites covering NLP issues)

- www.simonlaven.com
(a site devoted to "chatterbots")

Your own summary notes on NLP


20 min

Create your own brief summary notes on NLP.

Make sure you can describe:

- what NLP means;
- some of the problems;
- some successful applications;
- any issues arising from the use of NLP.

3.4 Intelligent robots

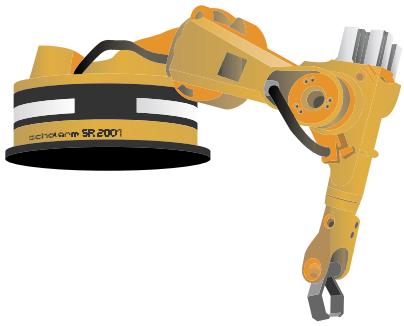
Learning Objective

After studying this topic, you should be able to describe:

- developments in intelligent robots;
- applications of intelligent robots;
- hardware and software issues related to intelligent robots;
- philosophical, ethical, moral and legal issues related to intelligent robots.

3.4.1 What are intelligent robots?

The idea of an intelligent **robot** has always been a dream that has been developed by science fiction writers. The most famous of these include the Czech playwright **Carel Capek**, who invented the word robot, and **Isaac Asimov** who wrote many short stories and novels based on the interactions between humans and robots. Film writers have also taken up the theme, and many sci-fi films are populated with robots and androids (robots in human form). So much for fiction! What about the real world?



It is now more than 25 years since the car industry started making use of robots instead of people in the highly labour-intensive manufacture of motor vehicles. Early robots were simply robotic arms with tools attached to the end. These were ideal for carrying out repetitive tasks on a car assembly line - tightening nuts, spraying paint, welding joints, moving heavy parts. Once programmed, they could repeat a task 24 hours a day with complete precision and reliability.

However, these were really only computer-controlled machines. There was no intelligence involved. Such a machine might spray paint empty space if a car was missing from the assembly line. Clearly some improvement was required!

The first improvement was to provide **sensor**s to the robot arm. A light sensor could be used to detect a bar code attached to the vehicle. This prevented the type of accident described above. A further benefit was that different car models could have different bar codes so that they could be identified by the robot arms. The sensor provided feedback to the robot controller, which then activated the appropriate program to paint that particular car body. This freed car manufacture from the concept of an assembly line dedicated to one particular model, and made the manufacturing process much more flexible. However, this is still not intelligence - the robot is simply following one of several pre-programmed sequences.

All of the above can be described as "dumb robots". Even with sensors and feedback, they are simply following programmed instructions, so cannot be considered as "intelligent robots".

Robots can be considered intelligent when they go beyond this idea of simple sensors and feedback, and display some further aspect of human-like behaviour. Examples might include:

- the use of **vision systems** and **pattern matching** rather than simple sensors, allowing the robot to recognise real objects, even when viewed at different angles
- the **ability to learn**, so that the robot can improve its performance at a task beyond the level at which it was originally programmed
- mobile robots which can work out the best route through a factory rather than following a pre-programmed route
- robots which can "walk" on uneven surfaces rather than using simple wheels or rollers

There is a great deal of research going on at all levels into ways of making robots more intelligent. Robots are being developed which can make decisions using **fuzzy logic** (that is, with incomplete information about a situation). Other robots are being provided with **neural network**s rather than traditional processors as their controllers. This allows them to learn and improve their performance, and works well with vision systems. The goal is to develop robots which can operate autonomously (that is, without human control). This is particularly useful in dangerous or inaccessible environments like the sea bed, underground or in space.

Image reproduced with kind permission of Honda Motor Co., Ltd

The Honda car manufacturing company are leading the way in researching **humanoid** robots. Their latest version, called ASIMO, is very highly developed. It can walk and move its arms. It is equipped with vision systems, and can respond to around 50 different spoken commands (in Japanese). It is well worth having a look at ASIMO's website to get up to date details of what it can do. However, don't forget that there are lots of robots in use, which show many features of intelligence, but look nothing like ASIMO or any human!

Identifying characteristics of intelligent robots

5 min

On the Web is a an activity which requires you to identify correctly the characteristics of intelligent robots. You should now complete this activity.

3.4.2 Practical Problems

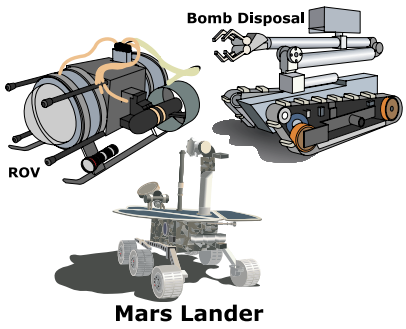
There are many practical problems to be overcome in developing intelligent or autonomous robots. Some of these problems are specific to intelligent robots, and some are relevant to all robots, whether intelligent or dumb. The problems to be overcome include:

- Providing a power supply
- Implementing mobility
- Implementing effective 3D vision systems
- Providing adequate processing power
- Navigation and path planning
- Manipulating objects (pick and place)

The web sites at the end of this topic will let you see how some of these problems are being addressed by current research.

3.4.3 Applications of intelligent robots

Intelligent robots are being used increasingly in many different areas. These include:



- manufacturing industry;
- bomb disposal;
- medical operations;
- deep sea exploration and rescue;
- space exploration.

There is research into developing **humanoid** robots - that is, one which look and move like humans. However, most intelligent robots look nothing like humans or sci-fi robots!

3.4.4 Are intelligent robots really AI?

Why is **robotics** considered to be a branch of AI? Which aspects of human intelligence does it model?

- **ability to learn:** yes, neural networks allow robots to learn about their environment;
- **creativity:** yes, intelligent robots may be able to invent new courses of action rather than following pre-programmed paths;
- **reasoning:** a robot controlled by an expert system should be able to justify its actions;
- **use of language:** robots can be equipped to understand spoken commands and to respond in synthetic speech.
- **vision:** many robots are being equipped with vision systems so that they can respond flexibly to new situations
- **problem solving:** intelligent control systems allow robots to make decisions
- **adapting to new situations:** the main idea of an intelligent robot is that it can operate in any situation rather than in a carefully designed environment such as the early car manufacturing assembly lines

3.4.5 Hardware requirements of Intelligent Robots

Intelligent robots developments are probably more demanding on hardware than any other area of AI. There are two aspects to this - the actual physical hardware of the robot (an engineering problem) and the control hardware (a computer science problem). In terms of the second of these, intelligent robots using environmental sensors such as vision systems, must be able to respond in real time to a large volume of complex data, accepting the data, processing it, and issuing instructions to the motors and joints of which the robot is built.

To be able to do this, many robots have multiple processors, each dealing with their own area. Each processor must be fast, and may also have to communicate with the other processors in the system.

Because of the hostile environments in which they often operate, robotic hardware has to be designed to withstand the intense pressures on the sea bed, the cold of outer space, bombardments by cosmic rays, or simply the dust, noise and vibrations in a manufacturing plant.

3.4.6 Philosophical, moral, ethical and legal issues of intelligent robots

Intelligent robotics raises a whole variety of **philosophical issues**, which are the basis of the stories which have been written by authors such as Asimov.

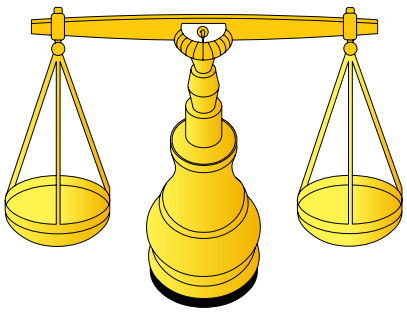
Some of these are very practical and relevant - the use of intelligent robots makes factories safer and more efficient in economic terms, but what happens to those who lose their jobs as a result.

There are also less practical issues to be considered:

- where and when should research stop?
- what if research eventually leads to machines which are "more intelligent" than humans?
- could we possibly consider that machines controlled by neural networks are in some sense "conscious"?
- if so, what rights do they have?

If such questions seem to you to be very fanciful, just think about the rate of progress in computing over the last 30 years. No-one can predict where we will be in 30 years time. All we can say is that current systems will look very primitive when we look back on them from the future.

On a more mundane level, there are **legal issues** arising when a robotic device causes damage or injury - whether in a car factory or a modern hospital operating theatre. Who is to blame, and accept responsibility - the software designer, the hardware designer, the user, the company which bought the machine, the company which sold the machine .....



3.4.7 To find out more about intelligent robots

Web sites

Note: to reach some of these resources you will need to navigate from the home page of the web site given.

- asimo.honda.com

  (web site describing ASIMO, Honda's humanoid robot, complete with movies to

watch and downloadable resources)

- www.dai.ed.ac.uk/groups/mrg/robots.html
  (some examples of research robots at Edinburgh University, including Robat and Gillespie)
- www.computermotion.com/clinicalapplicatins/roboticsprocedures/generalsurgery/
  (a surgical robotic system)
- www1.iff/fhg/de/eng/aut/index_rs.html
  (a German company which produces intelligent robots for cleaning windows on skyscrapers)
- marsrovers.jpl.nasa.gov/home/index.html
  (the Mars Rover independent mobile robot for exploring the surface of Mars)
- www-robotics.cs.umass.edu/research_frames.html
  (some interesting current projects, with video clips)
- cvs.anu.edu.au/bioroboticvision/
  (various examples of the use of vision systems in robotics, including a helicopter guidance system)
- www.frc.ri.cmu.edu/~nivek/faq/TOC.html
  (a robotics frequently asked questions archive)
- mi.eng.cam.ac.uk/research/vision/research.html
  (research into visually guided robots at Cambridge university)

20 min

Your own summary notes on intelligent robots

Create your own brief summary notes on intelligent robots.

Make sure you can describe:

- what makes a robot "intelligent";
- some of the problems;
- some successful applications;
- any issues arising from the use of intelligent robots.

3.5 Smart devices

Learning Objective

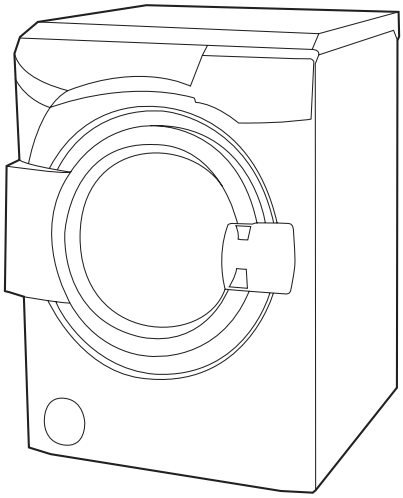
After studying this topic, you should be able to:

- describe examples of smart devices;
- explain what is meant by fuzzy logic.

At one time, computers and machinery were two completely different technologies. As we have seen in the last section, these two technologies have merged into the science of robotics. The first robots were completely "dumb", then sensors and feedback were added to make them more flexible, and most recently, intelligent robots have been developed, using the latest AI techniques to make robots even more flexible, to the stage where some robots can act autonomously.

A similar development has taken place where computing, then AI, have been applied to a wide range of household and industrial products.

3.5.1 Intelligent washing machines?



The earliest washing machines simply had an on-off switch, and possibly a dial for setting the water temperature. Gradually, these became more sophisticated, with a range of adjustable settings. Then someone had the bright idea of replacing all the complex mechanical or electrical switching devices with a microprocessor chip. This could be pre-programmed with a selection of different washing programs, which the user could select by simply selecting a number (1 for a hot "whites" wash, 2 for a cooler "coloureds" wash, and so on). The microprocessor (in effect a small dedicated computer) was embedded into the machine. Sensors in the machine interacted with the programs in the microprocessor to control all aspects of the running of the system.

Similar embedded control chips can now be found in many household devices.

Examples of this include central heating control systems, microwave cookers, CD players, electronic keyboards, burglar alarms, and many others.

The Orange Future Home project illustrates some advanced examples of embedded systems, with additional built-in communication. You can find out more about this at www.orange.co.uk/about/community/future_home_resource.html.

Just like the computer-controlled robots, most of these are not really "intelligent" devices, as they follow pre-programmed instructions in response to user input and feedback from sensors.

Recently, some appliance manufacturers are starting to advertise their products as "intelligent" or "smart". Can this claim be justified?

Take a look, for example, at Philips "intelligent washing machine" at www.semiconductors.philips.com/markets/mms/applications/intelligent_washing-machine

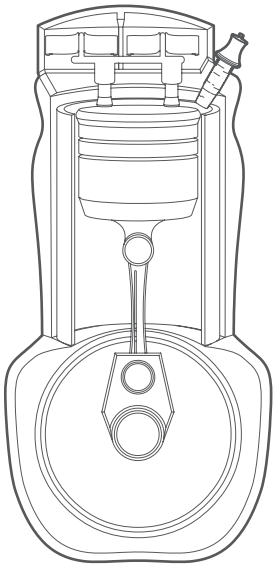
This is certainly an example of an embedded control system, with a wide range of sensors, and using an LCD panel to communicate with the user. The microprocessor runs a very complex program to interact with the user, take account of feedback from sensors, and make decisions to control heaters, pumps and so on. It is certainly complex, but is it intelligent?

We need to evaluate the system against our definitions of AI in topic 1.

Would you consider this washing machine to be intelligent, or is this simply an advertising slogan? There is no clear correct answer to the question!

There is an interesting article about this question which you can download from http://iieg.essex.ac.uk/papers/cognitive_disappearance_summary.pdf

3.5.2 Car Engine Management Systems



A similar story can be told about car engines.

Until the 1980s, car engines were basically mechanical devices, with everything controlled directly by the driver.

Since then, electronic control systems have been developed, with microprocessors taking feedback from sensors, and adjusting the fine tuning of the engine accordingly. As with the washing machine example, these have become increasingly complex (and car engines have become more responsive and efficient as a result), but it is difficult to justify the label "intelligent".

One development which is being increasingly used within embedded devices which can justify the label "intelligent" is the use of "fuzzy logic".

3.5.3 Fuzzy logic

Conventional computing is based on **Boolean logic**. In Boolean logic, everything is either true or false. For example, a whole number is either odd or even; there is no other possibility. This maps well on to the binary number system, with its 2 clear states - 0 and 1. Boolean logic states that if something is true, its opposite is false. Nothing can be true and not-true at the same time. So, obviously, a whole number cannot be both even and not-even (odd).

This theory works extremely well for many applications of computing. Most programs are based on this true-false logic. We can write code which is based on this idea; for example:

IF mark>50 THEN Print "Pass" ELSE Print "Fail"

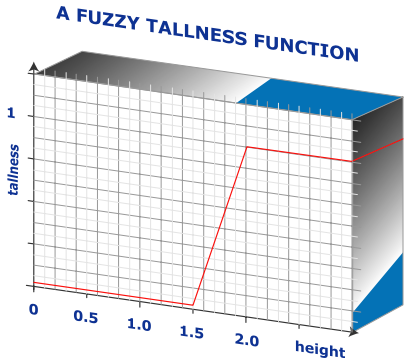
However, not everything works this way. Lots of "real world" problems contain information which is less precise. Bill's height is 2m - I would say that he is tall. Andrea is 1.5m - she is definitely short. What about someone who is 1.9m? Tall or short? Or 1.8m? Or 1.75m? You could argue that the cut-off should be exactly 1.853m, or some other height. But that would mean a person who is 1mm below the threshold is short, and another person 1mm taller is tall. This is unsatisfactory.

Fuzzy logic was invented in the 1960's by Dr. Lofti Zadeh of the University of California at Berkeley, to deal with real-world situations which were not well described by **Boolean logic**.

For example, in **fuzzy logic**, we could define a tallness function, like this:

- tallness = 1 (if a person is over 2m in height)
- tallness = 0 (if a person is under 1.5m)
- tallness = 2 $\times$(height-1.5) (if a person is between 1.5m and 2m)

Using this definition, a person is definitely tall (tallness =1) if they are over 2m, definitely not tall (tallness=0) if they are under 1.5m. A person who is 1.8m has a tallness of 0.6; a person who is 1.95m has a tallness of 0.9.

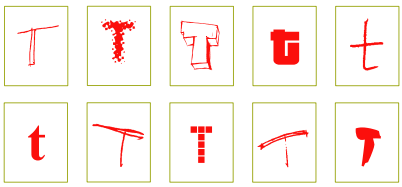


This does not seem to be a very revolutionary idea. However, in real situations, there may be many factors involved, all of which can best be described using a fuzzy function like the one above. For example, a climate control system in an office block has to respond to the temperature in many rooms, the humidity, whether these are already rising or falling, the pollen count inside and outside, and so on. Each of these can be described by a fuzzy function. **Fuzzy logic** is the mathematical theory which describes how these functions should be combined to give sensible outputs. Using **Boolean logic**, the control system might end up having to continually switch heaters and fans on and off. Using **Fuzzy logic**, it can operate much more smoothly, adjusting controls gradually in response to gradual changes in the environment.

3.5.4 Applications of fuzzy logic

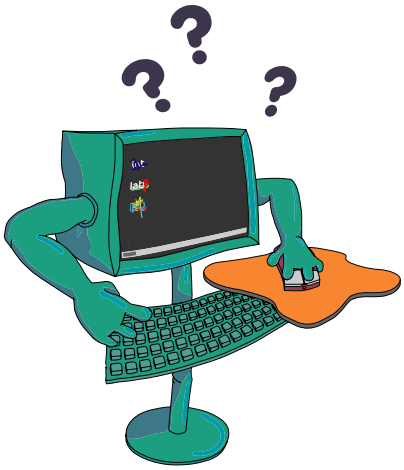
Fuzzy logic systems first appeared on the scene around 1990, especially in Japan. Since then, they have spread to many applications. These include control systems such as the one described above. Fuzzy logic control systems are being used in washing machines, cameras, microwave ovens, cars, traffic lights, lifts ... and more! To begin with, there were many sceptics, but the success of fuzzy logic control systems is so widespread that they are gaining universal application throughout the world.

Fuzzy logic can also be applied to **Expert System**s (see Topic 2.2 and Topic 5) by using **certainty factor**s. Expert Systems often base their advice on inputs which are not clearly true or false, and must use fuzzy logic to provide answers from fuzzy data.



Fuzzy logic has also proved very useful in hand-writing recognition systems. For example, a letter T could be defined as a vertical line with a horizontal line above it. But in a handwritten T, the lines are unlikely to be absolutely straight or vertical or horizontal. A recognition system has to be able to draw conclusions from the fuzzy data it is given.

3.5.5 Is fuzzy logic really AI?



Fuzzy logic can be considered to be a branch of AI, because it closely resembles the way the human brain appears to operate. All the time, our brains are receiving fuzzy information - information that is not clearly true or false - or information that cannot easily be categorised. Is the colour I see blue or green? There is no sharp distinction. We take in this fuzzy information, and are able to come to conclusions. Often these are also fuzzy! It seems best to go for option A, but there is some merit also in options, B, C and D. Finally we make a decision and act upon it. If we waited for all information to be sharp rather than fuzzy, we would be completely paralysed! Fuzzy logic systems behave in a similar way.

There have been philosophical arguments among mathematicians about whether or not **fuzzy logic** theory is required. Some argue that **Boolean logic** is adequate - once all the terms have been properly defined. However, the success of the applications of fuzzy logic make these arguments seem purely academic. It works, and it make system design easier, and the resulting systems more successful.

Although the logic may be called "fuzzy", this does not mean that it is mathematically imprecise. The outputs of fuzzy logic control systems can be calculated exactly, and the reasoning of a fuzzy expert systems can be justified. This means that the ethical and legal issues associated with neural networks are not such a problem as might at first be imagined.

To find out more about fuzzy logic

- www.fuzzy-logic.com/
  (a good introductory description of fuzzy logic with examples)

20 min

Your own summary notes on fuzzy logic

Create your own brief summary notes on fuzzy logic.

Make sure you can describe:

- what fuzzy logic means;
- some successful applications;
- any issues arising from the use of fuzzy logic.

3.5.6 Other intelligent control systems

Fuzzy logic is only one aspect of AI which is starting to be used widely in control systems. Other examples include:

- Neural networks
- Vision systems replacing simple sensors
- Adaptive systems (which can learn and improve their own performance)

You can read about one example of the use of these techniques in an industrial control system at http://icsl.marc.gatech.edu/Projects/braze/furnace.html

3.6 Summary

- "Strong AI" is the belief that AI research will eventually lead to the development of an autonomous intelligent machine;
- some researchers believe that "strong AI" will be possible, because of ever-increasing hardware and software advances;
- some believe that it is philosophically impossible to achieve "strong AI";
- major advances have been made in computer vision systems and natural language processing;
- the main stages in computer vision are image acquisition, signal processing, edge detection, object recognition and image understanding;
- a major difficulty is interpreting 2D images as 3D objects;
- the main stages of NLP are speech recognition, natural language understanding, natural language generation and speech synthesis;
- difficulties in NLP include ambiguity of meaning of words, similar sounding words, grammar inconsistencies and the changing nature of human language;
- NLP can be used for automatic translation, speech driven software, search engines and database interfaces;
- "dumb" robots, some with sensors, have been used in a range of industrial applications;
- robots may be described as "intelligent" if they have additional features such as vision systems, adaptive programs, neural nets or the ability to walk;
- current robotic research is working to overcome a wide range of practical problems associated with both dumb and intelligent robots;
- robotics raises a number of interesting philosophical, social and legal issues;
- intelligent software, including fuzzy logic, is being used in control systems for a range of embedded devices including domestic appliances, car engine control system and industrial processes.

3.7 End of Topic Test

An online assessment is provided to help you review this topic.

Topic 4

Artificial Neural Systems

Contents

Learning Objectives

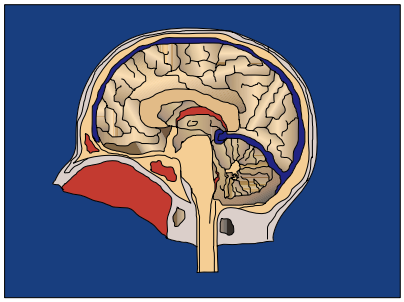
After studying this topic, you should be able to:

- *compare a human neuron with an artificial neuron;*
- *compare a neural net with a human brain;*
- *describe the structure of a neural net (artificial neurons, links, weights, layers);*
- *describe how a neural net learns through an iterative process;*
- *describe some applications of neural nets;*
- *describe some of the philosophical and legal issues associated with neural nets;*
- *explain how a neural net can be either modelled in software or hard-wired.*

4.1 Neural Nets

4.1.1 What are neural networks?

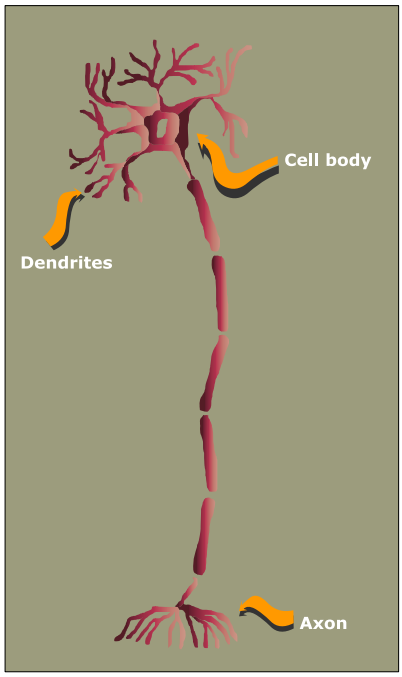
Many AI problems, as we have seen, are very difficult to solve using standard computing methods involving writing programs to run on conventional **Von Neumann processors** . This has led some researchers to approach such problems from a very different direction. The starting point for this approach to AI is the human brain itself. The human brain can show many complex aspects of intelligence, so why not try to model systems on the human brain?



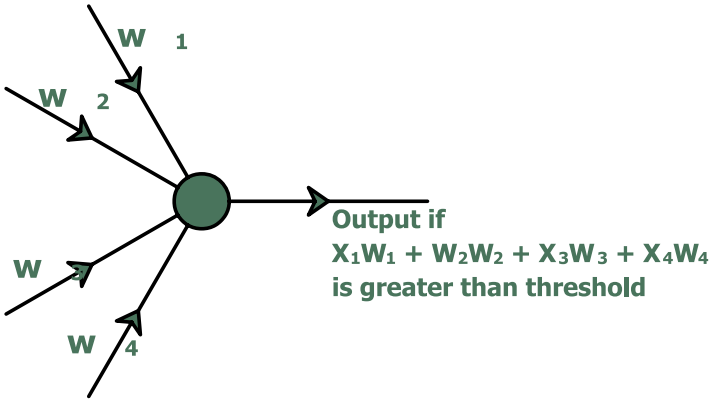
Research has shown that the human brain is made up of millions of simple nerve cells called **neurons**. Current estimates suggest that a human brain may have around 3 x 10^{10} of these neurons. This is a daunting number to build into a model. However, even simple creatures like insects can behave in intelligent ways (they can see, they can learn), and they do this with a much smaller brain of around 1 million neurons.

If we zoom in, we discover that each **neuron** is a relatively simple structure, consisting of a cell body, with many long tentacle like extensions called **dendrite**s. It also has a long tube-like extension called an **axon**.

Neurobiologists have studied the way neurons work, and have found that they operate in a fairly simple manner. The **dendrite**s are the cell's input devices. They receive signals from other neurons. If sufficient inputs are received, this triggers an electrical output signal along the **axon**. The signal can cross the small gap (called a synapse) between the **axon** of one cell and the **dendrite**s of the next by a chemical process. Each neuron may have up to 80,000 dendrites, so it receives inputs from a huge number of other neurons. The axon may divide into perhaps a hundred branches, so it can pass on signals to 100 other neurons. The human brain consists of millions of these relatively simple neurons communicating with each other in a highly interconnected network.

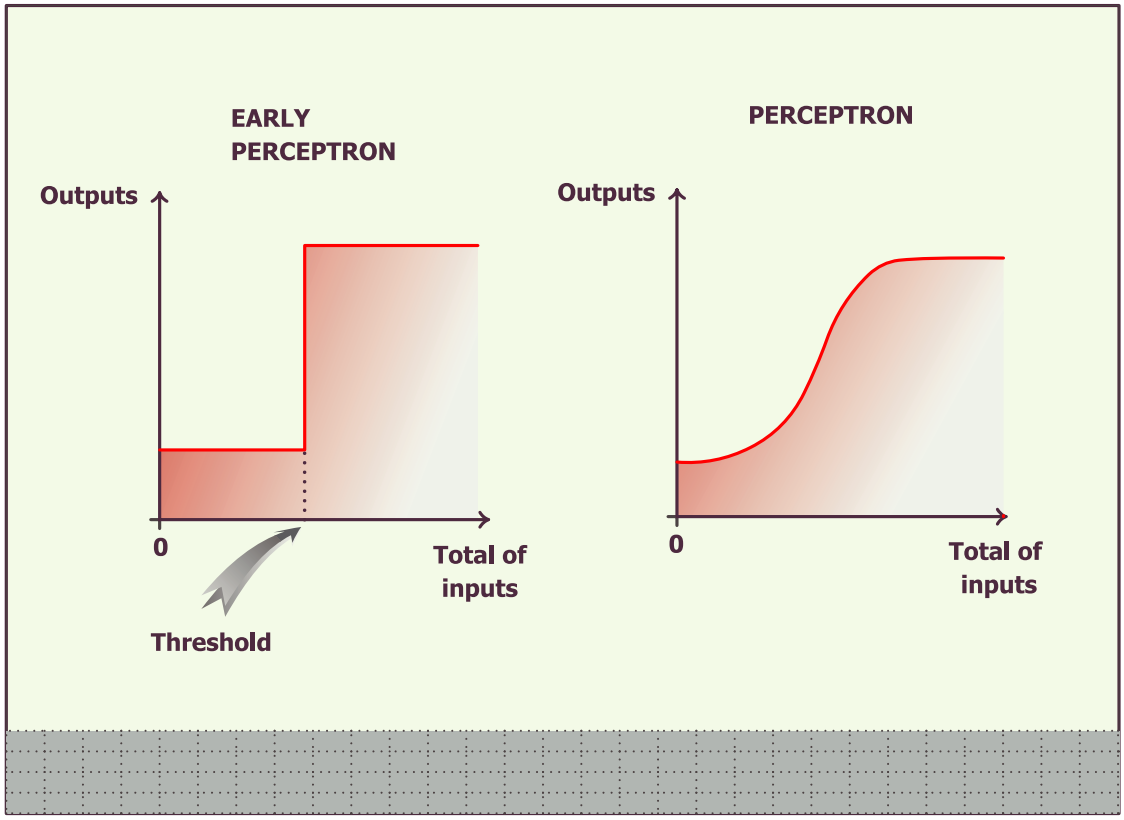


Early researchers realised it would be possible to model these simple neurons. In the 1960s, Rosenblatt developed the **perceptron**, based on a simplified neuron.



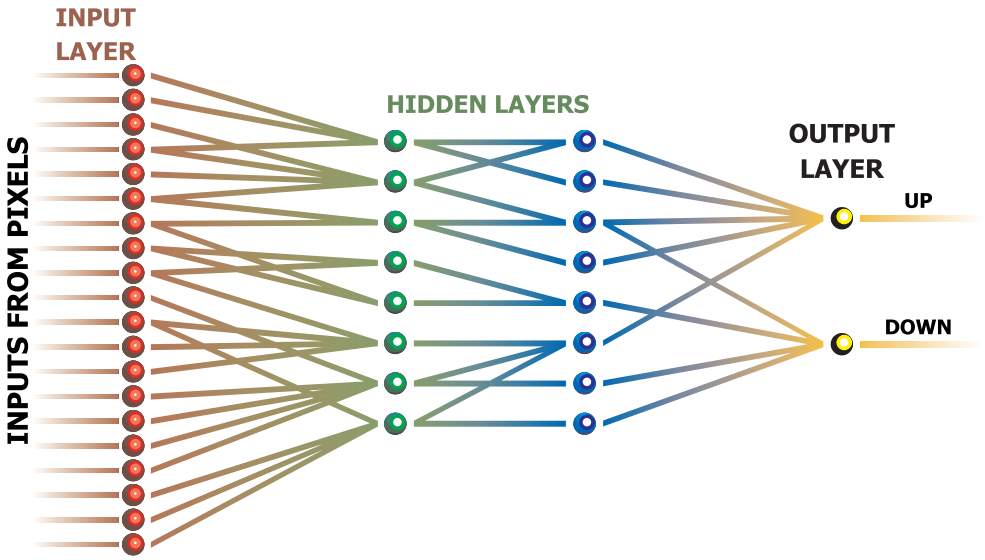
His **perceptron** had a small number of inputs (like **dendrite**s), and a single output (**axon**). It multiplied each input by a number (called a weight) which could be adjusted, added up all the inputs, and gave an output if the total came to more than a threshold value.

Unfortunately, a research paper by Minsky and Papert in 1969 put a damper on further research until the mid 1980's. The problems they identified were theoretical rather than practical. Their objections could be overcome by making **multi-layer network**s of these perceptrons, and by modifying the way they worked slightly. Instead of having a purely digital on/off output, they needed to have a continuously varying output curve:



With these two important modifications, **neural network**s developed as a hot area of research, and were surprisingly successful, even when the number of artificial neurons was small (anything from 20 up to a few hundred).

Neural networks have proved very successful at **pattern matching**, where they can be used to classify and generalise from data that is inexact or incomplete. Here, for example (in a simplified form) is how a neural network could be developed to distinguish between hand-drawn images of an up arrow or a down arrow.



First the images would be digitised into (say) 20 pixels on a 4 by 5 grid. Each of these pixels is either black (1) or white (0). This information is fed into 20 artificial neurons. These form the input layer of a neural network, with perhaps 2 **hidden layers** of 8 neurons, passing output to a final layer of 2 neurons. One of the final neurons should give an output for an up arrow, and the other one for a down arrow.

How would the system work?

Firstly, it would be trained by being "shown" a training set of inputs. To begin with the "weights" connecting the artificial neurons would be random numbers. Suppose the first image used as an input is an up arrow. The network accepts this input, and some of the neurons in the first layer are triggered. This in turn may trigger some of the neurons in the **hidden layers**. Finally one or other (or both) of the two neurons in the output layer may be triggered, giving an output voltage. If, by chance, it is the wrong one which gives the higher output, a process known as **back-propagation** is used to make minor adjustments to the weights connecting the neurons. The network is then tested on another image; once again the connecting weights are adjusted slightly. The process is repeated many times using the training images, until the weights have been adjusted so that system gives "correct" results most of the time (ideally 100%, but 80% or 90% may be good enough).

Now the system must be tested on a set of unseen images. Hopefully the adjusted weights will lead to it recognising all the test images correctly. If so the system can be put to use.

The design and training of neural networks seems to be as much a "black art" as a "science". Designers have to make decisions about how many artificial neurons to include, how many hidden layers, the initial values of the weights, the number of training examples and the algorithm used to adjust the weights.

Note that there are two ways of constructing a neural network. The network can be simulated as a **software model**. This is the normal method during the development phase, as it is easy to adjust the weights or the number of neurons in each layer until an optimal arrangement is found. Then the network can actually be constructed out of **hardware neurons**, so that, for example, it can be built into a robot.

The most surprising aspect of neural networks is that they actually work! Even networks similar to the one described above, with only 38 neurons, seem to be able to carry out tasks that could not easily be achieved using a traditional programming model. Systems with a few hundred neurons (far less than the human brain) can be extremely successful in image and voice recognition, and predicting stock movements or weather forecasting.

4.2 Applications of neural nets

Generally speaking, artificial neural networks are suited to **pattern recognition** problems where the data is plentiful and inexact, and there are no experts who can explain what the patterns are. The main weakness of a neural network is that it cannot explain how it reaches its conclusion. If your system need to do this, then you have to find a way of writing rules so that you can use an **expert system**. Problems may also be solved using a combination of expert systems and neural networks.

Neural networks have been applied successfully in the following fields:

- vision systems;
- voice recognition;
- handwriting recognition;
- data analysis;
- financial forecasting;
- weather forecasting;

and many others.

4.2.1 Are neural networks really AI?

Why are **neural network**s considered to be a branch of AI? Which aspects of human intelligence do they model?

- **ability to learn:** yes, algorithms have been developed which allow a neural network to adjust its own weights, so that its performance can improve;
- **creativity:** neural networks can be used to solve problems which have defeated an algorithmic approach; however, uses are more in pattern recognition rather than creative areas at present;
- **reasoning:** a neural network can arrive at an answer, but is unable to show the reasoning which it used to justify its answer.
- **use of language:** neural networks have been very successful in speech recognition and in speech production;
- **vision:** neural networks are very effective in extracting useful information from vision systems;

- **problem solving:** neural networks are effective in a wide range of problem-solving situations;
- **adapting to new situations:** most neural networks are designed and trained for a particular situation; they are flexible enough to cope with new input data, but could not be transferred from one type of problem (e.g. speech recognition) to a completely different problem (e.g. playing chess) without being redesigned.

4.2.2 Hardware requirements of neural networks

Hardware-based neural networks require completely different hardware from conventional computer science. A "normal" computer has one processor which is extremely complex, accessing a program stored in memory. It executes the steps of the program sequentially, following the order intended by the programmer. Its execution and output are entirely predictable.

A hardware-based neural network is almost exactly opposite in design. It has many (possibly hundreds or thousands) of processors. Each of these processors is extremely simple. It does not execute a program at all. Its output is not predictable (although if it has been well trained, the output should be "correct" a statistically acceptable percentage of times).

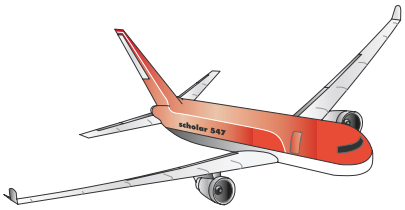
However, many neural networks are in fact **software models**, rather than actually constructed from hardware. Neural network simulators can be run on conventional computers. They do require significant processing power and memory as they must simulate all the artificial neurons in real time.

4.2.3 Philosophical, moral, ethical and legal issues of neural networks

The use of neural networks probably raises more tricky issues than any other area of Artificial Intelligence.

The **philosophical issues** arise from the whole concept of modelling the human brain (although it must be remembered that current neural networks come nowhere near the complexity of the human brain, with its millions of millions of neurons). We do not really understand what it is that makes us humans self-aware or conscious. However, many psychologists would argue that there is nothing mystical about it; it simply results from the physical nature of the brain and nervous system. If that is so, then living creatures with simpler brains may also be conscious or self-aware (perhaps to a lesser extent), and so there is no reason why a sufficiently complex artificial neural network should not also be self-aware or "conscious". At the moment, this is speculative - fun to discuss or think about, but of no practical relevance! However, if more and more complex networks are constructed, it is possible to imagine a situation where an artificial neural network has to be treated as having rights (analogous to human rights or animal rights). How far into the future this may be is anyone's guess, but it may be the most difficult philosophical issue that the human race will have to face ...

More **practical issues** arise from the application of neural networks, especially in safety-critical applications. The underlying issue here is the fact that neural networks are non-deterministic. That is, unlike conventional algorithmic computing solutions, the output from a neural network is not entirely predictable, and so no guarantees can be given that it will give the "correct" answer. Consider an extreme example of this.



Suppose a neural network had been designed and trained to fly a passenger aircraft. It would use vision systems, inputs from sensors and instructions from flight controllers as its inputs, and would then activate the various engines, flaps and machinery of the aircraft. After extensive training and testing, the designers might be able to say that the neural network made safe and correct decisions in 99.999999% of situations. However, it is impossible to predict how it will react to a completely unexpected and unforeseen situation. How would passengers feel about travelling in the aircraft? Probably "not too happy"!

However, the same is also true about a human pilot. The pilot will have been trained on a simulator and on real planes and have considered how to respond to every imaginable emergency, but it is not possible to give a 100% guarantee that the pilot will always make the correct decision. This is no real surprise - after all, the pilot's brain is basically a trained neural network too!

In reality, neural networks are not yet used in such extreme situations, so the issues which arise are less critical. A neural network may be used in a **pattern recognition** system to read postcodes on letters, so that they can be sorted into the correct mailbags. It sorts 99% of letters correctly. The manufacturer cannot give more than statistical guarantees about its operation. However, it is better and faster than human sorters used to be. There is no great disaster if the occasional letter goes astray. But what if it sends a vital letter on a wild goose chase to the wrong address, and as a result a multi-million dollar contract is lost, leading to a company going out of business, hundreds of people losing their jobs ....

4.2.4 To find out more about neural networks

Web sites

- www.20q.net

  (on-line 20 questions game, powered by a neural net. The 20Q neural net contains over 10,000,000 connections. The game uses the answers you have given it to determine the most probable objects, then it uses the most probable objects to determine the best question to ask. At the end of the game, once it has determined the answer, it adjusts the weightings. The game has been completely self-taught by people playing the game. It started with one question and one object, everything else was added by players.)

- www.tropheus.demon.co.uk

  (some neural network software to download)

- http://neuron.eng.wayne.edu/software.html

  (some neural net demo java applets)

- www.cormactech.com/neunet/whatis.html

  (the website for neural net software for Windows; the menu selection "uses" has a list of applications of neural nets; there are also some java applets)

20 min

Your own summary notes on neural networks

Create your own brief summary notes on neural networks.

Make sure you can describe:

- what a neural network is;
- its similarities and differences from a human brain;
- some successful applications;
- any issues arising from the use of neural networks.

4.3 Summary

- Neural nets are an attempt to model the human brain to solve problems;
- The human brain is made up of special nerve cells called neurons;
- Each neuron receives inputs (form its dendrites) and gives outputs (through its axon);
- It is possible to create artificial neurons, with several inputs and an output;
- An output is triggered if the sum of the inputs (each multiplied by its "weight") exceeds a threshhold value;
- The human brain has millions of highly interconnected neurons;
- In a neural net, the artificial neurons are arranged in several layers (an input layer, some hidden layers, and an output layer);
- A neural net has to be trained; in the training process, the weight of the links are adjusted until the net gives correct responses;
- The training of the net is an iterative process;
- A neural net is non-deterministic (unlike a normal algorithmic program whose output can be predicted with certainty);
- Neural nets have proved very successful in pattern matching applications;
- Neural nets can be software models, or hard-wired.

4.4 End of Topic Test

An online assessment is provided to help you review this topic.

Topic 5

Expert Systems

Contents

Prerequisite knowledge

There is no prior knowledge assumed for this topic. However, you may have some background knowledge of the purpose, advantages and applications of Expert Systems from the Intermediate 2 unit.

Learning Objectives

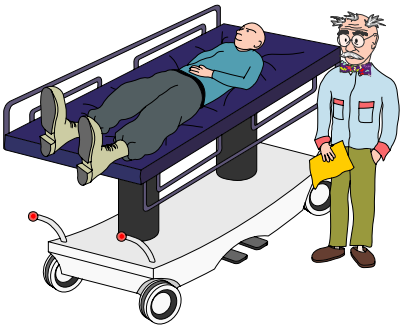
After studying this topic, you should be able to:

- *Describe the purpose of an expert system;*

- *Describe the components (knowledge base, inference engine, user interface and working memory) of an Expert System;*
- *Distinguish between an Expert System and an Expert System Shell;*
- *Describe some contemporary applications of Expert Systems;*
- *Describe some advantages and disadvantages of Expert Systems;*
- *Describe some moral and legal issues relating to Expert Systems.*

5.1 Introduction

5.1.1 What are expert systems?



One day, you are not feeling well, so you arrange to visit your doctor. When you are called into the consulting room, instead of your doctor sitting there, you are faced with a computer terminal. On the screen is a prompt for your name. You enter your name, and the computer asks you why you have come. It prompts you to enter your symptoms, how long you have been feeling like this, whether you have had these symptoms before, and many other questions.

After some time, the computer says that you are suffering from bubonic plague. It advises you to avoid contact with other people and to stay in bed for the next 2 weeks. The printer spits out a prescription for you to take to the chemist, and the computer screen advises you to return for a further consultation next month ....

Instead of consulting your doctor (a human expert), you have just used a computer "expert system". Fiction? At the moment, yes, but perhaps not too far away into the future.

An **Expert System** is a computer program which can do the job of a human expert. That is, it can give reliable advice on a limited area of expertise, and it can explain its reasoning. It is able to interact with any user in the same way that you might consult a human expert such as a doctor or a lawyer or a specialist in the flora of Northern Canada!

An expert system can be defined as "a computer system which can draw reasoned conclusions from a body of knowledge in a restricted domain"

5.1.2 Advantages and disadvantages of expert systems

So why bother developing systems to replace human experts?

There are a number of reasons.

Firstly, human experts are not 100% reliable; they may disagree with each other, or forget to take a crucial factor into account before making a decision.

Secondly, they may not always be available! In a small company, the only expert on some area may be ill or on holiday just when some advice is required. Your doctor may not be able to see you until next week. Even worse, an expert may move to another job (taking his expertise with him) or may retire. A new employee may take years to build up the same level of expertise that his predecessor had.

Thirdly, an expert may have unsurpassed knowledge in the field, but be hopeless at explaining that to someone else.

An expert system can overcome all these weaknesses of human experts.

An expert system (properly programmed) should be 100% reliable; that is, it will never forget to take something into account, and will never give advice which it cannot justify from the information it has been given.

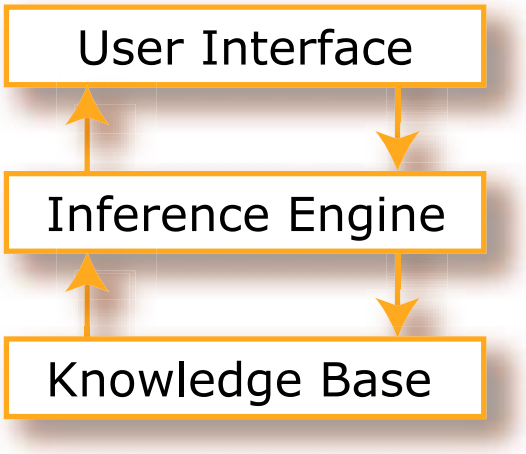
An expert system is always available; you can even take it with you if you have a portable computer, or you could consult an expert system over the internet.

An expert system should be programmed to explain and justify any advice it gives.

5.2 Expert systems and shells

5.2.1 Anatomy of an expert system

An expert system consists of 3 parts:

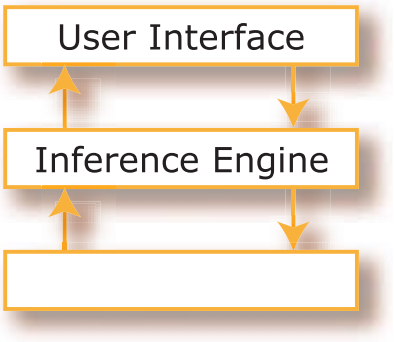


- the **user interface**, allowing the user to input information in response to questions generated by the system. The user interface will also be able to give advice and (very importantly) explain why it is giving that advice.
- the **knowledge base**, consisting of all the facts and information about the topic, and all the rules and relationships between these facts. This information will have been provided by one or more **domain expert**s, and converted into an appropriate form by a **knowledge engineer**.
- the **inference engine**, which searches the knowledge base, generates questions to the user, and uses the rules and relationships which have been programmed into the knowledge base to come to conclusions which can be reported to the user.

When an expert system is being used, the **inference engine** will generate new facts and rules from the **knowledge base** and from the user's responses. These are all held in **working memory**.

5.2.2 Expert system shells

An **expert system shell** is an expert system with no knowledge base. It consists only of a user interface and an inference engine. Expert system shells are used to create expert systems. The programmer only has to code the knowledge base, as the user interface and inference engine already exist.



It would be possible for a programmer to develop an expert system using a standard software development environment. The programmer would need to develop a suitable **user interface**, providing a means of dialogue between the system and the user. This could be done using standard tools and libraries.

It would also be necessary somehow to code all the knowledge (facts, rules, relationships) and include them in the software as a **knowledge base**. This might be done using a knowledge representation language like Prolog (see Topic 4).

Finally, it would be necessary to develop software which could interact with both the user and the knowledge base to draw conclusions from the knowledge base - an **inference engine**. Developing all 3 modules of this system would be a major and complex software development task.

The breakthrough in the development of expert systems was the realisation that these three components could be developed as independent modules, and that the same user interface and inference engine could be used to develop many expert systems. A user interface and inference engine without a knowledge base is called an **expert system shell**.

Using an expert system shell to develop an expert system may be less flexible than developing an expert system from scratch - for example, you have to use the user interface provided - but it certainly reduces the time taken and the skills required. It also reduces the chance of errors, as the user interface and inference engine will have been tested.

Sentence completion - expert systems

5 min

On the Web is a sentence completion task on Expert Systems. You should now complete this task.

5.2.3 the development of an expert system

The development of an expert system from an expert system shell involves a **knowledge engineer** and one or more **domain expert**s. The knowledge engineer has an understanding of how expert systems work, and how to represent knowledge in an appropriate format for an expert system. The domain expert needs no Computing knowledge; he or she is an expert in the subject - a medical expert, a lawyer, a professor of ancient Chinese literature, a computer technician, a sports coach ....

The task of the knowledge engineer is to extract the knowledge from the domain

expert(s) and convert it into facts and rules which can be written in an appropriate knowledge representation language. This is not an easy task, and may take a long time. The knowledge engineer will need to interview the domain expert, consult written sources (books, lecture notes ...) and gradually try to structure the knowledge in a logical way. This may involve many techniques including semantic nets (as in Topic 8) or tables. It will be an iterative process, with the knowledge engineer returning to the domain expert to check that the facts and rules are correct before moving on to the next section of the knowledge base. Particular difficulties may arise if 2 experts do not agree about some aspect of the domain, or where knowledge is uncertain.

Eventually, the *analysis* and *design* stages will be complete, and the knowledge engineer can move into implementation. The knowledge can be constructed into a knowledge base and inserted into an expert system shell. Having completed this stage, the domain expert will be called back to test the completed expert system. This is likely to lead to the need to add more facts and rules, correct any misunderstanding, and refine details. The knowledge engineer can then implement these corrections, and further testing will be carried out by the domain expert. This process continues iteratively until the domain expert is satisfied that the expert system works correctly.

15 min

Review questions

Q1: What is the purpose of the User Interface in an Expert System?

Q2: What is the purpose of the Knowledge Base in an Expert System?

Q3: What is the purpose of the Inference Engine in an Expert System?

Q4: Name a suitable domain expert for the following expert systems:

a) Scottish politics 2001-2002;
b) Managing a football team;
c) How to pass Higher Computing;
d) How to become a millionaire.

Q5: Distinguish clearly between an expert system and an expert system shell.

5.3 Applications of expert systems

5.3.1 Some examples

Expert systems were (and perhaps still are) the most successful commercial spin-off from AI. By focussing on a restricted domain, researchers could by-pass all the difficult issues of developing intelligent systems which might pass the **Turing Test**. The benefits of computer experts over human experts were also readily understood by business and industry, so money was made available to support research. As a result, there are many examples of expert systems in use. Here are a few areas of success, but there are many more.

- **Medical Expert Systems:** The scenario described above is still science fiction, but there are many expert systems in specialised branches of medicine. Normally

these would be used by doctors to support their own knowledge, rather than by complete non-experts. Examples include MYCIN (to diagnose blood disorders), PUFF (lung infections) and BABY (for monitoring babies in intensive care)

- **Geological Exploration:** One of the earliest successful expert systems was called Prospector, which was able to analyse echo soundings, and suggest places where it would be worthwhile drilling to find new mineral deposits.
- **Computer Configuration:** Another early success was XCON, which allowed salesmen with little expert technical expertise to select the correct component parts of a mainframe computer to meet a customer's requirements.
- **Chemical Analysis:** DENDRAL, as far back as 1965, was designed to identify molecular structure in unknown substances. This was partly funded by NASA, with a view to using a system like this on a spacecraft to analyse the Martian soil.
- **Legal Advice:** One system called AREST was developed by the University of Oklahoma to assist the police in robbery enquiries. Another system is used by the Department of Social Security to check claims by clients
- **Finance:** A system called FOLIO has been developed to advise on stock exchange investments.
- **Manufacturing:** One system is used to identify faults in circuit boards, and advise on how to correct them.

These are simply a selection of the hundreds of experts systems in use; there are many more.

5.3.2 Where to find out more about expert systems

Paper-based resources

- Advanced Higher Computing Artificial Intelligence support materials, published by Learning and Teaching Scotland (LTS), 2001
- Higher Computing Artificial Intelligence support materials, published by Learning and Teaching Scotland (LTS), 1998

Web sites

- www.bcsnsg.org.uk/itin08/darling.htm

  (a good general overview of expert systems. To find this resource you may need to start at the bcsnsg homepage, then select ITIN and the archive for issue 8.)
- www.cee.hw.ac.uk/~alison/ai3notes/chapter2_5.html

  (course notes on expert systems - and other aspects of AI - from Heriot-Watt University)
- www.computing.surrey.ac.uk/research/ai/PROFILE/mycin.html

  (a very detailed description of MYCIN with an example consultation and related expert systems)

- smi-web.stanford.edu/

  (notes on many of the expert systems developed, including MYCIN and DENDRAL can be found by selecting 'research projects' and then 'projects of historical interest'.)

- www.aaai.org/AITopics/html/expert.html

  (a gateway to many article and web sites about Expert Systems)

- www.aiinc.ca/demos/index.html

  (links to several demo expert systems for you to try, including whale identification, student selection and spa pool fault diagnosis)

- www.expertise2go.com/webesie/tutorials/ESIntro/

  (an easy introduction illustrated by a car fault diagnosis expert system)

5.4 Evaluating an expert system

60 min

Evaluating an expert system

Your tutor will provide you with a number of simple expert systems to consult.

- Describe the domain of each expert system (the area of knowledge that it covers), and note its limitations - what it can't do.
- Make some brief notes about the user interface: you should comment on its use of menus and graphics, how it provides answers, and how you provide it with the information it needs.
- Describe any improvements which could be made, or any alternative solutions to the problem.
- Consider any legal, moral or ethical implications which might arise from the use of any of these expert systems.

Note: evaluating an expert system is one of the practical skills required to pass this unit, so your tutor may wish to see your report.

5.5 A practical example

In this topic we will consider some extremely simple expert systems. Real useful expert systems are much more complex!

Like a Prolog program, the knowledge base of an expert system consists of facts and rules. These rules are written in a knowledge representation language.

5.5.1 Example: the coins of Upper Scholaria

As a travel agent, you are often asked by travellers about how to identify the coins of Upper Scholaria. You decide it would be useful to develop an expert system which customers could consult directly. You phone the local school to ask if any pupils studying Higher Computing would like a temporary post as a knowledge engineer. Shortly afterwards, Lucinda arrives. She explains that she has been studying Expert Systems at school and will be pleased to develop the expert system for you.

Lucinda begins by analysing the problem. She searches the internet for websites about Scholaria and finds some information about its monetary system. She makes notes about the different types of coins. She is not clear about the difference between a 20H coin and a 50H coin, so she phones a domain expert - the receptionist at La Hotel Granda in the capital city of Scholaria. This soon clarifies the difference.

Next, as the design stage of her project, she constructs a table of the information she has discovered:

value	**colour**	**size(mm)**	**shape**	**symbol**
1H	bronze	18	round	fish
6H	bronze	18	round	bird
10H	silver	18	round	flower
20H	silver	22	7 sided	fish
50H	silver	22	round	fish
1J	silver	25	triangular	mountain
6J	two tone	30	7 sided	bird

The next stage is to convert this knowledge into rules.

5.5.2 Backward and forward rules

There are 2 basic types of rules used in the knowledge representation languages of Expert System Shells. These are known as:

- **Forward Rules** (of the form *IF condition(s) THEN conclusion*)
- **Backward Rules** (of the form *conclusion IF condition(s)*)

The rules might look like this (the exact syntax depends on the expert system shell chosen):

Forward Rules	Backward Rules
IF colour = bronze AND size (mm) = 18 AND shape = round AND symbol = fish THEN value = 1H.	value = 1H IF colour = bronze AND size(mm) = 18 AND shape = round AND symbol = fish.
IF colour = bronze AND size (mm) = 18 AND shape = round AND symbol = bird THEN value = 6H.	value = 6H IF colour = bronze AND size(mm) = 18 AND shape = round AND symbol = bird.
IF colour = silver AND size (mm) = 18 AND shape = round AND symbol = flower THEN value = 10H.	value = 10H IF colour = silver AND size(mm) = 18 AND shape = round AND symbol = flower.
IF colour = silver AND size (mm) = 22 AND shape = 7 sided AND symbol = fish THEN value = 20H.	value = 20H IF colour = silver AND size(mm) = 22 AND shape = 7 sided AND symbol = fish.
IF colour = silver AND size (mm) = 22 AND shape = round AND symbol = fish THEN value = 50H.	value = 50H IF colour = silver AND size(mm) = 22 AND shape = round AND symbol = fish.
IF colour = silver AND size (mm) = 25 AND shape = triangular AND symbol = mountain THEN value = 1J.	value = 1J IF colour = two tone AND size(mm) = 25 AND shape = triangular AND symbol = mountain.
IF colour = two tone AND size (mm) = 30 AND shape = 7 sided AND symbol = bird THEN value = 6J.	value = 6J IF colour = two tone AND size(mm) = 30 AND shape = 7 sided AND symbol = bird.

5.5.3 Implementation and testing

The Coins of Upper Scholaria


30 min

Using an expert system shell available to you, create a "Coins of Upper Scholaria" knowledge base. Check whether your chosen expert system shell uses forward or backward rules (some allow you to use either). You may need to alter the syntax for the shell that you have chosen.

Compile the knowledge base, and test it by consulting its user interface. Make sure that:

- it correctly identifies each of the coins;
- it reports "no solution" if you describe a coin which does not exist (e.g. a silver coin, 22mm in size, which is seven-sided and has a flower on it.)

5.5.4 Justification and explanation

A human expert can not only give good advice, but can explain or justify his reasoning. This is important for several reasons:

- it helps the enquirer to understand the subject better;
- it means that the enquirer can learn how the expert thinks;
- it gives the enquirer more confidence in the advice given.

The last of these is particularly important. Imagine an oil tycoon considering where to spend £1 billion on a new drilling exploration project. He asks his expert advisor, who answers "Try drilling at Deadman's Gulch in Arizona". The tycoon might ask the expert to justify this suggestion. "No reason - just a hunch", says the expert. The tycoon would want more reassurance than that! If the expert could show him geological maps, the results of test drills and so on, the tycoon would be more inclined to take the advice.

The same is true of Expert Systems. The user requires some justification to give confidence that the advice given by the system is based on logical reasoning, and is not just the computer's "hunch". All expert systems provide this. The user interface has built-in functions which can explain the reasoning of the inference engine. For this reason, the user interface is often called an **explanatory interface**.

The explanatory interface can provide two types of justification - these are *why* and *how*.

Why?

The *why* justification feature allows the user to ask the system **"Why are you asking me that question?"**. This is useful when using an expert system to learn how the expert thinks. Why does the doctor ask you how often you drink alcohol? Why does the technician ask you if you have recently downloaded something from the internet? Why does the antique dealer ask if the furniture has been stored in a humid atmosphere? By revealing why a question is being asked, the user can gain an understanding of the line of reasoning which the system is trying to follow.

How?

The *how* justification feature allows the use to ask the system **"How did you come to that conclusion?"** This is useful at the end of the consultation, as it shows the reasoning which led to a particular conclusion. It is also useful during testing of the system, especially if the system provides an unexpected response. Was it a mistake in the programming or was it a new conclusion which the domain expert hadn't thought of before, but was perfectly justified.

15 min

Exploring Justification Features

Load up your Scholarian Coins expert system. Consult the system.

At each stage where the system asks you for information, it should give you an option to choose "Why?"

Try this option at each stage of the consultation. You should be shown the fact or rule which the system is currently trying to establish.

At the end of the consultation, the system should give you the option to choose "How?"

Try this option. You should be shown the series of logical steps by which the system came to its conclusion.

5.6 Philosophical, practical, legal and ethical issues

5.6.1 Are expert systems really AI?

Why are Expert Systems considered to be a branch of AI? Which aspects of human intelligence do they model?

- **ability to learn**: NO - the system can only improve if extra knowledge is put into the knowledge base by a knowledge engineer
- **creativity**: NO - predictable output from facts and rules
- **reasoning**: YES - an expert system can justify any advice it gives, explaining in a logical way how it reached that particular conclusion
- **use of language**: NO;
- **vision**: NO;
- **problem solving**: YES - the whole purpose of the expert system is to help humans to solve problems, but only in a limited domain;
- **adapting to new situations**: YES - a system may combine the expertise of several experts, and so come to new conclusions.

5.6.2 Hardware requirements of Expert Systems

Expert Systems require:

- fast processors - to be able to handle complex knowledge bases at useful speeds;

- large amounts of RAM - to hold the knowledge base, and allow the inference engine to apply search techniques.

Both processor speeds and amount of RAM are increasing rapidly, so it should be possible for more and more complex expert systems to be developed, into wider and less restricted domains.

5.6.3 Ethical and legal issues of expert systems

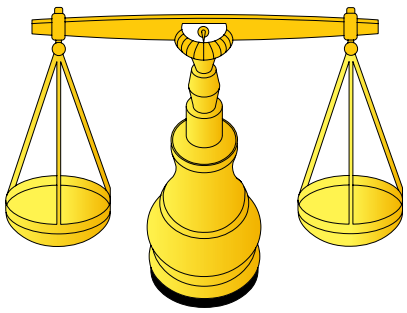
The main issues relating to expert systems are to do with the legal implications which could arise if the advice given by an expert system leads to damage to equipment, loss of earnings or even loss of life.

Consider, for example, the use of an expert system giving weather advice to round-the-world-yachtsmen (or women). What if, following the advice to sail, the boat is wrecked and the crew are lost at sea. Who is to blame, and who accepts responsibility (which may lead to massive financial implications)?

Who is to blame?

5 min

Either on your own, or with another student, brainstorm a list of people who might be responsible when an expert system gives advice which leads to loss of life or damage to property.



You might argue that any of the domain expert, the knowledge engineer, the programmer of the expert system shell or the software company selling the product could be held responsible.

However, generally speaking, the **user** is considered to accept responsibility by acting on the advice given by an expert system, just as the reader of a book would be responsible if taking advice from a book. No doubt, the software company's lawyers will have carefully inserted no-responsibility clauses in the small print of the software documentation.

There are also ethical issues about replacing human experts with expert systems, leading to de-skilling and loss of jobs. On the other hand, expert systems are still being developed, so there are new jobs for **knowledge engineer**s too.

How would you feel about consulting an expert system rather than a doctor? You might prefer the anonymity if you were embarrassed about the nature of your illness. Besides, recent research has shown that medical diagnosis systems can give correct advice in around 70% of cases. Are you happy with that level of good advice? No? Other research shows that general practitioners only make a correct diagnosis in under 40% of cases!

5.7 Summary

- The purpose of an expert system is to draw reasoned conclusions from a body of knowledge in a limited domain;

- Expert systems have advantages over human experts - they are permanent, cost effective, give consistent advice and are portable;
- Expert systems have some disadvantage compared to human experts - they only cover a limited domain, lack common sense, cannot retain new knowledge, are inflexible, and take time and expertise to develop;
- An expert system consists of a user interface, a knowledge base, an inference engine and working memory;
- An expert system shell is an expert system without a knowledge base;
- Expert system shells are use for constructing expert systems;
- Expert systems have been used in a huge and ever-increasing variety of different knowledge domains.

5.8 End of Topic Test

An online assessment is provided to help you review this topic.

Topic 6

Search-based Problem Solving

Contents

Prerequisite knowledge

There is no prior knowledge assumed for this topic. However, you may have some background knowledge of simple search trees and search strategies from the Intermediate 2 unit.

Learning Objectives

After studying this topic, you should be able to:

- *Describe some applications of search-based problem solving;*
- *Describe depth-first and breadth-first search strategies;*
- *Exemplify search strategies on a search tree;*
- *Compare breadth-first and depth-first strategies in terms of order of visiting nodes, memory implications, advantages and disadvantages;*
- *Describe and exemplify combinatorial explosion;*
- *Describe (with examples) the purpose and benefits of heuristics.*

6.1 Introduction to search

Many of the types of problem being researched within AI can be tackled by using the power of a computer to search for a solution among many possibilities.

For example, a **chess-playing computer** must look at the possible legal moves which it can make, and search for the move which is most likely to lead to success in winning the game.

A program designed to hold an intelligent **conversation** has to be able to consider what it might say next, and choose an appropriate response from the many possibilities.

An **intelligent mobile robot** carrying goods from one part of a factory to another must be able to search for a suitable route through the building to get to its intended destination.

An **expert system** must be able to search its knowledge base to find suitable advice for the person who is interrogating it.

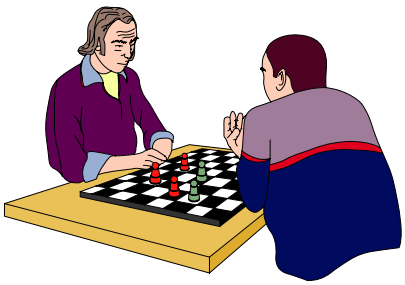
In all of these situations, the problem can be described in terms of:

- a starting state (e.g. the current position of the pieces in a chess game);
- a goal state (e.g. checkmating the other player to win the game);
- a set of operations (e.g. the list of all possible legal chess moves).

5 min

Identifying start and goal states

Consider the intelligent robot described above. Write down its start state, goal state and set of operations.



In most situations, there are many possible actions that can be taken. For example, at the start of a chess game, there are 20 possible legal moves. In response to these moves, the opponent may make any one of 20 legal responses. This means that a chess computer which looks only one move ahead must consider 20 x 20 possible positions.

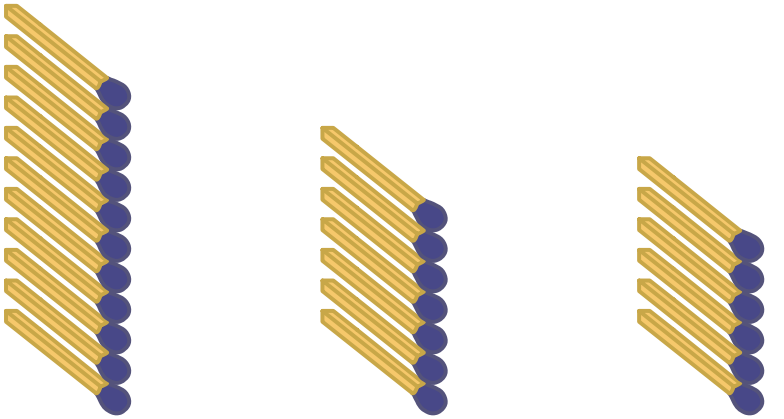
A program could be written to select one of these moves at random, but an intelligent program must have some systematic way of searching through the possibilities and choosing the best one. There are two basic strategies for searching through all the possible states in a situation. These are called **depth-first** searching and **breadth-first** searching.

To understand these two search strategies we are going to consider a simple problem called the matches problem. We could have considered some real life situation or even the game of chess, but it is easier to understand if we take a very simple problem like this one.

6.2 The matches problem (using breadth first search)

6.2.1 Introducing the problem

You are given 24 matches, arranged in 3 piles as shown below:



This is the **start state**. It can be written as (11,7,6).

The aim of the game is to move matches to form 3 equal piles. We can write that the **goal state** is (8,8,8).

The rules are as follows:

- any move must double the number of matches in the destination pile;
- no move may result in a pile ending up with 0 matches.

In principle, there are 6 possible operations at any time:

1. move matches from the left pile to the middle pile (**LtoM**);
2. move matches from the left pile to the right pile (**LtoR**);
3. move matches from the middle pile to the left pile (**MtoL**);
4. move matches from the middle pile to the right pile (**MtoR**);
5. move matches from the right pile to the left pile (**RtoL**);
6. move matches from the right pile to the middle pile (**RtoM**).

However, at any time, only some of these moves are possible, depending on the number of matches in each pile.

Q1: What are the 3 possible moves from the start state?

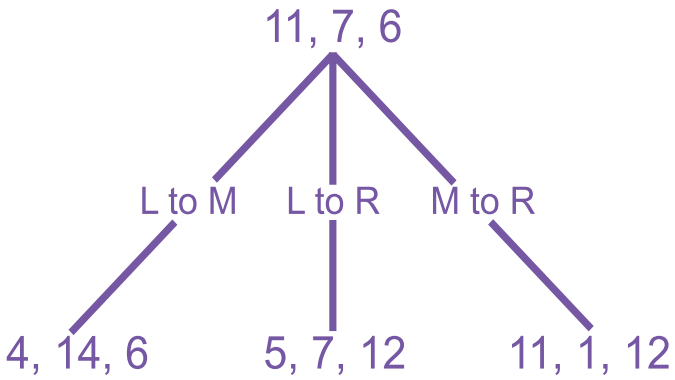
Q2: Why are RtoM, RtoL and MtoL not legal moves from this starting state?

6.2.2 Creating a search tree

The 3 legal moves could be written down like this:

1. (11,7,6) using LtoM becomes (4,14,6)
2. (11,7,6) using LtoR becomes (5,7,12)
3. (11,7,6) using MtoR becomes (11,1,12)

These moves can be shown on a diagram called a **search tree**, as follows:



None of the states we have reached is the goal state (8,8,8), so we need to apply the operators again to each of the three states reached.

20 min

Expanding the search tree for the matches problem to the second level

Get a large sheet of paper (at least A4 size). Turn it sideways. Copy the search tree above. Next, consider the possible legal operations which can be applied to (4,14,6). Add these to your search tree.

Do the same to (5,7,12), and finally to (11,1,12). To be systematic, always try the operators in the order LtoM, LtoR, MtoL, MtoR, RtoL, RtoM.

If possible, work with a partner, as it is very easy to make mistakes if no-one is checking what you are doing.

Check your answer is correct before you continue.

20 min

Expanding the search tree for the matches problem to the third level

Consider the possible legal operations which can be applied to each of the 9 possible states at the foot of your search tree. Add these to your search tree (you can see why a large sheet of paper was recommended). Stop when you reach the goal state.

Keep this search tree. You will need it again in Topic 6.5.2.

Simulation of the matches problem - breadth first

On the Web is a simulation of the matches program being played and the search tree being developed to the solution you found above. You should now look at this simulation.

6.2.3 Definition of breadth-first search

Working out all the possible operations at each stage, and adding them to your search tree is a very tedious task. However, it is the type of task which a computer can be programmed to do very accurately and quickly, and it is the basis for many AI

applications including games playing programs, natural language processing, intelligent robots and expert systems.

The method of searching the search tree that you have used is called **breadth first** searching. This is because you search for the goal state or solution to the problem by working across the **search tree** at level 1, then searching across the tree at level 2, and so on until you find a solution. Your search tree becomes wide before it becomes deep!

Matches problem (2) (optional)

30 min

As extra practice, you could construct another breadth-first search tree for the matches problem. This time make the start state (13,9,2), and expand as far as the 3rd level.

This time, you will discover that even after applying 3 operations, there is still no goal state in the search tree. To find a goal state, you would need to expand the tree one further level, but this is too time-consuming for a human!

6.3 The matches problem using depth first search

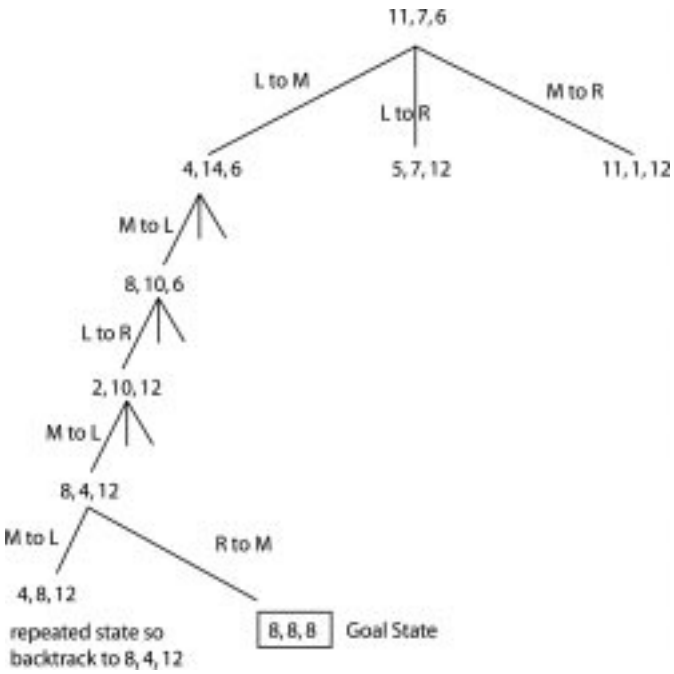
In the previous topic, you studied the matches problem using **breadth first search**. This is not the only way to find a solution to a search-based problem. It is also possible to use a technique called **depth first search**. Instead of searching all the options at each level before moving to the next level, depth first searching continues down a single branch of the tree until it finds a solution, or comes to a dead end. To see how this works, we will again consider the matches problem with (11,7,6) as the start state.

Matches problem search tree (depth first)

20 min

Take a fresh sheet of paper, and put the start state (11,7,6) at the top of the page. As before, add in the 3 first level "nodes", (4,14,6), (5,7,12) and (11,1,12). This time, keep applying operators to the left branch of the tree until you find a solution (or until a state is repeated, or turns out to be a dead end). Again, to be systematic, always try operators in the order LtoM, LtoR, MtoL, MtoR, RtoL, RtoM. If a state is repeated or is a dead end, then back track to the previous node and try the next operator from there.

If you followed the instructions above, you should have a diagram like this:



Simulation of the matches problem - depth first

On the Web is a simulation of the matches program which shows the search tree being developed by depth first search to the solution you found above. You should now look at this simulation.

6.4 Comparing depth-first and breadth-first searches

Both breadth-first and depth first searches lead to a solution to the matches problem. However, they lead to different solutions. Which is the best method?

The answer to this question depends on what you mean by the best solution. Consider the solutions we found.

Breadth first led to the following solution:

- (11,7,6) $>$ (4,14,6) $>$ (4,8,12) $>$ (8,8,8);
- the solution requires 3 operations (LtoM, MtoR, RtoL);
- we know that there is no solution using fewer operations, since we have checked all of level 2;
- but to find it, we had to check out 16 possible moves.

Depth first led to the following solution:

- (11,7,6) $>$ (4,14,6) $>$ (8,10,6) $>$ (2,10,12) $>$ (4,8,12) $>$ (8,4,12) $>$ (8,8,8);
- the solution requires 6 operations (LtoM, MtoL, LtoR, MtoL, MtoL, RtoM);

- it is not the best solution to the problem if best means the one requiring least moves;
- but to find it, we only had to check out 7 possible moves.

6.4.1 Advantages and disadvantages

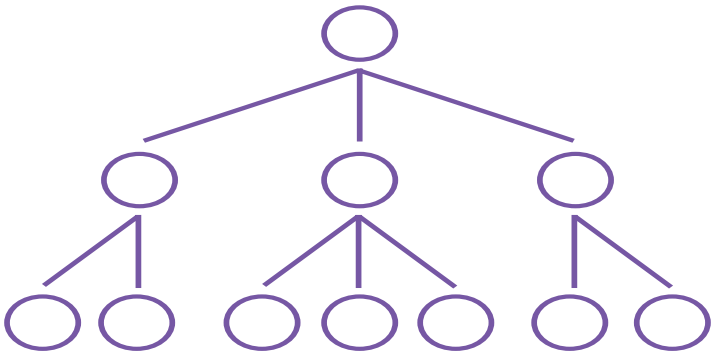
These results are typical of the difference between depth-first and breadth-first searching. Differences include:

- breadth-first search finds the "best" solution;
- depth-first search requires less memory, as it does not need to store all the possible branches at each level;
- depth-first search may be "lucky" and find a solution quickly, but it may follow a branch which has no solution, or where the solution is a long way down the branch;
- depth-first searching requires the programmer to prevent the search getting into an infinite loop by instructing the search to backtrack if it finds a state it has already found;
- breadth-first search is guaranteed to find a solution if one exists.

The speed of either search method can be increased by using **parallel processing** (Topic 2.7). We have assumed that the nodes of the search tree must be investigated in a particular order. This is only true if the processing is being carried out by a conventional "von Neumann" computer system with a single processor. Parallel processing computer architectures allow many nodes to be considered simultaneously. At any level in the search tree, different processors may be allocated to the evaluation of different branches, so that the whole process is speeded up. Under these conditions, the distinction between depth-first and breadth-first searching becomes blurred.

Review questions

Q3: This diagram represents the search tree for a simple problem:



Copy this search tree, number the nodes in the order they would be visited in a breadth first search, and draw a continuous arrow showing the "route" taken by the search method

Q4: Copy this search tree again, but this time number the nodes in the order they would be visited in a **depth** first search. Again, draw a continuous arrow showing the "route" taken by the search method

Q5: Which search method will always find a solution if one exists?

a) breadth-first
b) depth-first

Q6: Which search method normally uses less memory?

a) breadth-first
b) depth-first

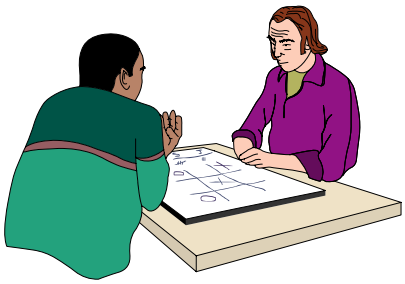
Q7: Which search method may get into an infinite loop, and never find a solution?

a) breadth-first
b) depth-first

6.5 Exhaustive and heuristic search

6.5.1 Problems with exhaustive search

So far we have looked at searching for a solution by constructing a search tree, and using either breadth-first or depth-first searching to find a solution. Both methods have their own relative advantages and disadvantages. These methods are know as **exhaustive search** methods (not because they are tiring to use, but because they consider every possible option). Because of this, both methods suffer from a serious problem if they are applied to any more complex situation.

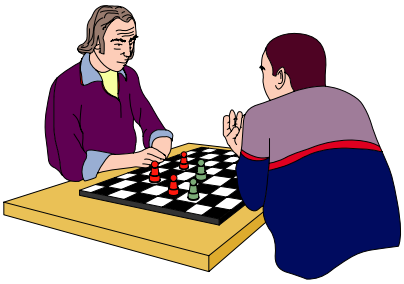


Consider a program to play a (relatively) simple game like **noughts and crosses**. For the first move, there are 9 places you can place your X. Your opponent places his O in any one of the remaining 8 places. This means that there are 72 (9 x 8) possible playing positions after 1 move each.

Imagine trying to draw out the search tree for a game of noughts and crosses. It would be very time consuming, and require a very large piece of paper. If you continue to the end of the game (i.e. all places filled), there are 9 x 8 x 7 x 6 x 5 x 4 x 3 x 2 x 1 nodes in total. That comes to 362880 possible games. (In fact there are fewer than this, because some games will be complete before the whole grid is filled in.)

It would take you or me a long time to work out all the options, but an 800MHz processor with 64Mb of RAM should manage it without too much trouble. (In the early days of AI, with slow processors and small amounts of memory, it might have been a bit more of a problem).

So, a noughts and crosses programs should be possible using one of these search techniques.



What about **chess**? We have already seen that there are 20 possible first moves, followed by 20 responses from your opponent. As the game progresses, the number of possible moves actually increases rather than decreasing. So after one move each, there are 400 possible positions; after 2 moves each, there are more than 160,000 possible positions, and after 3 moves each, there are over 64 million positions!

A good chess player will normally look at least 3 moves ahead when deciding which move to make, and a world champion may look up to 6 moves ahead. That means a **search tree** with over 4000 trillion (million million) possibilities. Even the world's most powerful processors could not manage to search such a large tree in a sensible length of time.

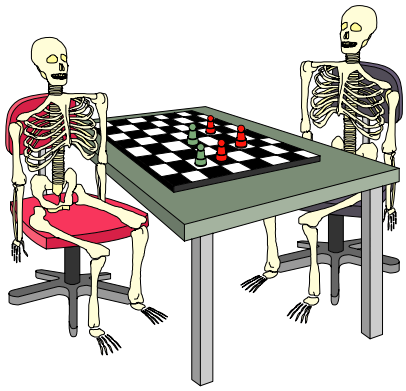
This problem is called **combinatorial explosion**.

Combinatorial explosion occurs in almost all real-life problems. Just think about how many decisions you have made already today, and how may options there were for each decision! The same problem occurs when considering programs to simulate conversation. There are hundreds of thousands of words in the English language, which can be combined in an endless variety of ways.

The solution to combinatorial explosion is to use **heuristics**.

6.5.2 Heuristics

Heuristics



The world chess champion may look ahead 6 moves, but he doesn't look at every possible move - otherwise his opponent would probable be dead by the time he had considered all 4000 trillion possibilities. Instead, he knows, from what he has been taught, and from his experience, that some moves are more likely to be helpful than others. He has "rules of thumb" which quickly reduce the number of moves he needs to consider:

- he knows that there is not much point looking at moves which would allow his major pieces to be captured;
- he also knows that it is best to build up a strong group of pieces towards the centre of the board;
- he knows that some pieces are more valuable than others.

Using "rules of thumb" like these, he can quickly narrow down the possible options to a much smaller number, and he can then search through these (in his imagination) in an exhaustive way probably using a combination of breadth-first and depth-first searching.

For example, he may be in a situation where there are 43 legal moves, but applying his "rules of thumb" he knows that only 4 of them are likely to very useful. He may consider the likely situation 1 move beyond each of these 4 possibilities (breadth first search). One of them looks particularly promising, so he tries to think ahead what are the most likely series of moves from that position (depth-first search). This may lead to a dead end, so he backtracks, and looks in depth at another possibility, and so on.

Similarly, we use **heuristics** in everyday conversation. When someone asks you a question, you don't consider every possible word or sentence which you might use as a reply. You choose from a restricted set of options of "sensible" responses. How do you know what responses are sensible? You have built up experience over the years, and you know that certain responses are more likely to lead to a successful outcome than others.

However, **heuristics** are not always helpful. Sometimes you can win a game of chess by setting a trap for the other player. For example, you might put your queen (a very powerful piece) into a place where it can be taken by your opponent. This does not look like a promising move, and a heuristic might suggest this move is not worth considering. However, if your opponent does take your queen, this might make a surprise checkmate possible, and you win the game. A good chess player has to be able to spot moves like that which break the usual "rules of thumb". The whole idea of a "rule of thumb" is that it **usually** works, but **not always**.

Similarly, in conversation. You might have a rule of thumb which says "never shout at a teacher". There might be a situation when you would need to overrule this rule, for example if the teacher was about to step out in front of a fast-moving vehicle. (You would, wouldn't you?)

20 min

Heuristics and the matches problem

Let's return to the matches problem. If you experiment, you will find that states with 4 matches in any of the piles are more likely to lead to a solution. States with 8 matches in any pile are even better. These are "rules of thumb" or heuristics.

This could be programmed by giving each node a value using the following formula:

Value = (number of 8s x 2) + (number of 4s).

So, for example:

- value(11,7,6)=0;
- value(8,14,2)=2;
- value(4,8,12)=3.

Go back to your matches problem breadth-first search tree, and write the value beside each node. You will then see how choosing the node of the highest value will lead

directly to the solution.

6.5.3 Intelligent searching

There are a number of factors which make a search-based system really appear intelligent:

- the ability to use **heuristics** rather than simple blind **exhaustive search**. There is nothing particularly intelligent about methodically working through every possible option. Computers are good at doing so, but it only makes them into "very fast idiots".
- the system's ability to **develop its own heuristics**, rather than just use those which have been programmed in. For example, an intelligent chess program would learn from its successes and failures, and adapt its rules about what moves are likely to be successful. This means it will learn from experience and improve its ability to play as time progresses.
- the system should be **able to override its own heuristics** when there is an option which breaks these rules, but would lead to success.

Matching word meanings using glossary terms

10 min

On the Web is an activity that asks you to match correctly a list of word meanings and terms. You should now carry out this activity. You may wish to take a printout when you have completed this task and retain it in your folder of work.

6.6 Summary

- Many AI problems can be described in terms of a start state, a goal state and a set of operations;
- These problems can be solved by searching through all possible states to find a solution;
- Two search strategies are breath-first and depth-first searching;
- These strategies can be illustrated using a search tree, with a root, branches and nodes;
- Breadth first search tries all nodes at the first level before moving on to the second level;
- Breadth first search will always find the best solution, but it requires more memory that depth-first search;
- Depth first search completes the search of one branch before backtracking and trying other branches;
- Depth first search uses less memory, but it may not find the best solution, or it may get stuck in an infinite loop;

- Breadth-first and depth-first are both exhaustive search strategies (they try every possibility);
- In real problems, there are often many branches from each node, leading to combinatorial explosion (too many possibilities for the system to search in a reasonable time);
- To reduce the search space, heuristics (rules of thumb) can be used to select the most likely branches;
- A system which can develop its own heuristics could be described as intelligent.

6.7 End of Topic Test

An online assessment is provided to help you review this topic.

Topic 7

Prolog (facts, rules and queries)

Contents

Prerequisite knowledge

There is no prior knowledge assumed for this topic. However, you will find it useful to have used Prolog at Intermediate 2 level to create and query simple Prolog programs

Learning Objectives

After studying this topic, you should be able to:

- *Describe the characteristics of a declarative language (Prolog);*
- *State that a Prolog program is a knowledge base of facts and rules;*

- *Describe how the knowledge base can be searched or queried to find a goal;*
- *Explain the meaning of the terms goal, sub-goal, instantiation, matching, back-tracking point;*
- *Develop simple Prolog programs, including multi-argument clauses and rules;*
- *Use negation in Prolog;*
- *Query a Prolog knowledg base using complex queries.*

7.1 Introduction

Traditional programming languages work by the programmer designing then coding an **algorithm** (set of logical actions) which will solve a specified problem. The "intelligent" part of this process is carried out by the program designer(s). Whatever language is being used, it is the program designer who has to figure out how to solve the problem, then work out a sequence of steps which can be programmed. The computer then blindly follows the program steps from beginning to end. The computer is not showing any intelligence at all.

This approach to programming has been incredibly successful in many branches of Computing Science, but less so in the areas associated with Artificial Intelligence. There are a number of reasons for this. For example, many AI problems cannot be reduced to a set of logical steps. From a philosophical view, it could also be argued that if a program can be written in the traditional way, the program can hardly be described as "intelligent".

We have also seen that many AI problems are difficult to solve, not because of the complex logic involved, but due to the vast amount of information which must be incorporated. For example, in **natural language processing**, a major difficulty is simply the vast number of words, their meanings and the virtually infinite way they can be combined to form sentences. Declarative languages (such as Prolog) were developed as a tool for dealing with this type of situation.

Using a "traditional" **procedural language**, the task of the programmer is to develop and code an algorithm to solve the problem. Using a **declarative language**, the task of the programmer is to code an exact description of the problem, including all the relevant facts and the relationships between them.

As well as being classified as a declarative language, **Prolog** can also be described as a **knowledge processing language** or a logic language; it has been designed for problem solving which requires the handling of large bodies of knowledge. This knowledge has to be structured by the programmer in a logical way, then a suitable query has to be entered which will lead to the answer to the problem. Hence the name **Prolog** (Programming in Logic).

In this topic, you will explore some fairly simple Prolog programs which will illustrate how this works.

7.2 A simple Prolog knowledge base

Consider a visitor going on holiday with a large extended family. In order to behave intelligently, the visitor must get to know everyone's name, how they are related to each other, and as many facts as possible about each person and about their relationships. Once the visitor has established the basic facts and relationships, she can figure out all sorts of other things. She can then use this information to decided on appropriate actions to take (or avoid!). Fortunately, she has been sent some information about the family in advance...

7.2.1 Analysis and design

In the family group, there are 5 females - Catherine, Shiona, Tanya, Sam and Susan - and 5 males - Bill, Lee, Greg, Andy and Pete. Bill and Shiona are the parents of Tanya and Pete. Shiona and Sam's parents are Catherine and Andy. Sam and Lee have two children, Susan and Greg. Finally, Haggis is Susan's pet dog!

Before the visitor can make sense of this information, it has to be organised into some logical order - this is the **analysis** phase of the development process.

10 min

Analysis

Take the paragraph of information from above, and try to write it out in a systematic way as a list of simple facts like:

- Catherine is female;
- Catherine is parent of Shiona;

and so on ...

Be careful not to add any information which isn't actually there!

Check your answer before continuing.

Design

You have completed the **analysis** phase, so next we move on to the **design** phase (just as in any software development process). In fact, you have already made some design decisions. For example, choosing to write all the parent-child relationships as "Lee is the parent of Susan" rather than "Susan is the child of Lee" was a design decision.

Notice that if you wrote "Lee is the **father** of Susan" you have jumped ahead. It doesn't say so explicitly in the information given. You have deduced that from two facts - the fact that Susan if Lee's child, and the fact that Lee is male!

7.2.2 Implementation

We have completed analysis and design, so we can move on to **implementation**.

Each of the facts above can be coded into a single Prolog fact. The exact syntax will depend on which version of Prolog you are using, but it might well look like this:

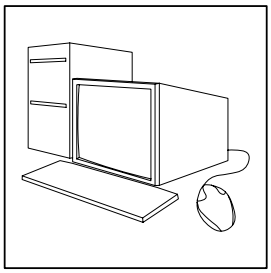
fact in English	**fact in Prolog**
Catherine is female	female(catherine).
Shiona is female	female(shiona).
Tanya is female	female(tanya).
Sam is female	female(sam).
Susan is female	female(susan).
Bill is male	male(bill).
Lee is male	male(lee).
Greg is male	male(greg).
Andy is male	male(andy).
Pete is male	male(pete).
Haggis is a dog	dog(haggis).
Bill is parent of Tanya	parent(bill,tanya).
Bill is parent of Pete	parent(bill,pete).
Shiona is parent of Tanya	parent(shiona,tanya).
Shiona is parent of Pete	parent(shiona,pete).
Catherine is parent of Shiona	parent(catherine,shiona).
Catherine is parent of Sam	parent(catherine,sam).
Andy is parent of Shiona	parent(andy,shiona).
Andy is parent of Sam	parent(andy,sam).
Sam is parent of Susan	parent(sam,susan).
Sam is parent of Greg	parent(sam,greg).
Lee is parent of Susan	parent(lee,susan).
Lee is parent of Greg	parent(lee,greg).
Susan is the owner of Haggis	owns(susan,haggis)

Notes about Prolog syntax (check with your tutor that the following information is correct for the version of Prolog which you will be using)

- each Prolog fact ends with a full stop (.)
- a fact consists of a **predicate** (e.g. male) followed by 1 or more **arguments** (in brackets)
- the arguments must not begin in upper case (capital letters) even when they are names
- if there is more than 1 argument, they are separated by commas (some versions use spaces rather than commas - check with your tutor)

20 min

Your first Prolog knowledge base



- load your Prolog system
- key in the knowledge base as above
- find out how to add internal commentary, and add your name and today's date at the top of the knowledge base
- compile it
- if it doesn't compile, correct any syntax errors, then compile again
- save it as Family1
- print out the knowledge base

7.3 Querying a knowledge base

Now that the information has been coded as a Prolog knowledge base, we can use it to answer some questions about the family. In Prolog, we do this by writing **queries**.

7.3.1 Simple queries

Suppose we wanted to check if Lee was male. We would enter the query:

?male(lee)

(the answer should be yes)

If we wanted to check if Sam was male, we would enter the query:

?male(sam)

(the answer should be no)

Suppose we wanted to find out the names of all the males. This time, we would enter the query:

?male(X) or **?male(Who)**

In this query, we use a **variable** called X or Who. In Prolog, a variable name begins with a capital letter (upper case).

The answer should be : X=bill, X=lee, X=greg, X=andy, X=pete

Is Bill anyone's parent? We could enter this query:

?parent(bill,X)

The answer should be: X=tanya, X=pete

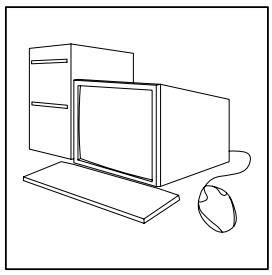
And, finally, who owns Haggis? The Prolog query would be:

?owns(X,haggis)

and the answer should be: X=susan

Querying the knowledge base (1)


20 min



1. load your family knowledge base;
2. compile it;
3. find out how to enter a query;
4. Enter the following queries, and write down the results:

1. **?male(lee)**
2. **?male(sam)**
3. **?male(X)**
4. **?parent(sam,greg)**
5. **?parent(sam,tanya)**
6. **?parent(sam,Child)**
7. **?parent(bill,X)**
8. **?parent(P,pete)**
9. **?owns(X,haggis)**
10. **?child(greg,sam)**

Check your answers

Now try out a few queries of your own.

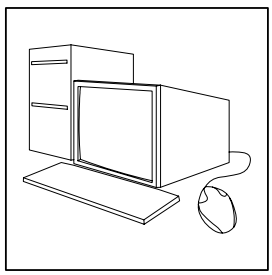
7.3.2 Complex queries

Suppose the visitor wants to know who is Tanya's dad ... the knowledge base cannot give an answer, because it doesn't know what a "dad" is! We can easily solve this by using complex queries. Complex queries require 2 or more conditions to be true. The conditions are usually separated by a comma (,) in the query.

Querying the knowledge base (2)


10 min

Load up your Family1 knowledge base again. Enter the following query:



?parent(X,tanya),male(X)

This should provide the correct answer, since it will check for an X who is the parent of Tanya, and who is male. Try it yourself.

Write and test a query to find who is Tanya's mother.

7.4 Adding rules to a knowledge base

The family knowledge base is fine as far as it goes. Using queries, you can check any facts about who is who, and their basic relationships, but it doesn't let you deduce any further facts, like who is Susan' brother, or even who are Sam's children. You could probably work these out for yourself. To do so, you would use **rules** that you had learned. Some of the rules seem obvious (if Sam is Greg's parent, then Greg is Sam's child), and some are a bit more complicated (if two people share a parent, then they must be brother or sister to each other).

We are going to add some simple **rules** to the family knowledge base. The first rule will define who is someone else's **child**. The second will define who is **brother** to someone.

The first one is easy. If X is the parent of Y, then Y must be the child of X. It's obvious to you, but not to the computer. We need to write a Prolog rule to represent this information. The rule could be written as:

child(X,Y) if parent(Y,X).

Note: the word "if" is often represented in Prolog by the symbol :- so this rule could be written as:

child(X,Y) :- parent(Y,X).

What about brother and sister?

X is the brother of Y if

- X has a parent Z
- Y also has the same parent Z, and
- X is male.

In Prolog, this could be written as:

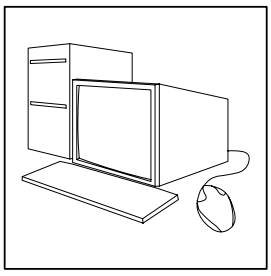
brother(X,Y) if parent(Z,X) and parent(Z,Y) and male(X)

or (depending on the syntax of your system)

brother(X,Y) :- parent(Z,X), parent(Z,Y), male(X).

30 min

Adding rules



Load your Family1 knowledge base.

1.

- add the **child** rule (child(X,Y) :- parent(Y,X).)

- compile it, and run some queries to check it works

2.

- add the **brother** rule ((brother(X,Y) :- parent(Z,X), parent(Z,Y), male(X).)
- compile it, and run some queries to check it works

3.

- design and implement a **sister** rule
- compile it, and run some queries to check it works

4.

- design and implement a **grandparent** rule (hint: a grandparent is the parent of a parent)
- compile it, and run some queries to check it works

5. design, implement and test rules for:

   - **granny** (hint: a granny is a female grandparent!)
   - **grandpa** (hint: a male grandparent)
   - **grandchild**

6. design, implement and test rules for:

   - **cousin** (hint: you are my cousin if we share the same grandparent!)
   - **aunt**
   - **uncle**

Make sure you check your answers, and have a look at the important notes following the expected answers.

Print out your extended knowledge base, and save it as Family2.

7.5 How Prolog solves a query

Before you start this topic, you will find it very useful to have a hard copy of the Prolog file Family2, which should be as follows: (the clauses have been numbered for ease of reference)

```
1.  female(catherine).
2.  female(shiona).
3.  female(tanya).
4.  female(sam).
5.  female(susan).

6.  male(bill).
7.  male(lee).
```

```
8.  male(greg).
9.  male(andy).
10. male(pete).

11. dog(haggis).

12. parent(bill,tanya).
13. parent(bill,pete).
14. parent(shiona,tanya).
15. parent(shiona,pete).

16. parent(catherine,shiona).
17. parent(catherine,sam).
18. parent(andy,shiona).
19. parent(andy,sam).

20. parent(sam,susan).
21. parent(sam,greg).
22. parent(lee,susan).
23. parent(lee,greg).

24. owns(susan,haggis).

25. child(X,Y):-parent(Y,X).

26. brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),not(X=Y).

27. sister(X,Y):-parent(Z,X),parent(Z,Y),female(X),not(X=Y).

28. grandparent(G,Y):-parent(G,Z),parent(Z,Y).

29. granny(G,Y):-grandparent(G,Y),female(G).

30. grandpa(G,Y):-grandparent(G,Y),male(G).

31. grandchild(A,B):-grandparent(B,A).

32. cousin(X,Y):-grandparent(G,X),grandparent(G,Y),not(X=Y).

33. aunt(A,X):-parent(P,X),sister(A,P).

34. uncle(U,X):-parent(P,X),brother(U,P).
```

It is important to understand Prolog's method of solving queries for 2 reasons.

Firstly, it will help you to design and debug any Prolog programs you write if you understand the way the system works. This is especially true as you develop more and more complex programs.

Secondly, it is a requirement of the Higher Computing syllabus!

Fortunately, it is not too difficult. Let's look at some examples.

7.5.1 Simple queries

Solving ?male(bill)

The Prolog system searches the knowledge base from top to bottom, until it finds a fact which matches the **goal** which is **male(bill)**. In this case it finds an exact match at clause 6, as both the predicate (male) and the argument(bill) match the goal. The system therefore reports the answer "yes" or "true". No other clauses match the goal, so it only reports the one solution.

Solving ?male(X)

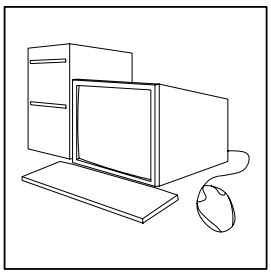
Again the Prolog system searches the knowledge base from top to bottom. At clause 6, it finds a match for the predicate male. This clause becomes an exact match, if the variable X in the goal takes the value "bill" from clause 6. We say that X is **instantiated** to bill. The system therefore reports X=bill as a solution to the goal.

If we request further solutions, the system **unbinds** the variable X, puts a marker (called a **backtracking point**) at clause 6, and continues to search down the knowledge base. It finds a match for the predicate male at clause 7, instantiates X to lee, and reports the solution X=lee.

This process is repeated 3 more times, finding the solutions X=greg, X=andy and X=pete.

If more solutions are requested, Prolog will continue to search down the knowledge base, but will find no more clauses which match the goal, so will report "no more solutions".

Tracing a simple query




10 min

Load the Prolog knowledge base Family2. Compile it. Set the trace facility on.

Run the 2 queries above, i.e. **?male(bill)** and **?male(X)**. Check that they give the expected answers. Use the trace log to observe Prolog's method of solving the queries.

Repeat this using a few other simple queries, like **female(bill)**, **female(X)**, **parent(bill,pete)**.

Prolog search strategy simulation - male(X)


20 min

On the Web is a simulation of the Prolog search strategy. You should now look at this simulation.

7.5.2 A complex query

Solving ?child(X,bill)

This query is slightly more complex. As before, the Prolog system searches through the knowledge base from top to bottom. This time, the first clause whose predicate matches the goal is clause 25: **child(X,Y) :- parent(Y,X)**. This is a **rule** rather than a **fact**. This matches the goal, as Y is instantiated to bill.

Prolog marks this clause (25) as a **backtracking point**, and uses the rule to generate a **sub-goal**, **parent(bill,X)**.

Prolog returns to the top of the knowledge base, and searches for a match for the new sub-goal. It finds its first match at clause 12: **parent(bill,tanya)**, so instantiates X to tanya, and marks another backtracking point.

Returning to the first backtracking point, and using the **instantiation**s X=tanya and Y=bill, clause 25 is now a match for the goal, so Prolog can report the solution X=tanya, Y=bill.

If we request further solutions, Prolog unbinds the variable X, and searches for further solutions to the sub-goal **parent(bill,X)** starting from the backtracking point at clause 12. It finds another match at clause 13: **parent(bill, pete)**, so marks this as its new backtracking point, and **instantiates** X to pete. Using X=pete and Y=bill, clause 25 is now a match for the goal, so Prolog can report the solution X=pete, Y=bill.

If we request more solutions, Prolog will continue to search for another match to the sub-goal **parent(bill,X)** from the backtracking point at clause 13, but finds no further matches. It then returns to the original backtracking point at clause 25, and searches on down the knowledge base for matches for **child(X,bill)**. Again it finds no more matches, so the system reports "No further solutions".

20 min

Prolog search strategy simulation - child(X,bill)

On the Web is a simulation of the Prolog search strategy. You should now look at this simulation.

7.5.3 Query with 2 sub-goals

Solving **?grandparent(catherine,Who)**

Here is one more query to trace. Don't panic if it seems complicated. As you look at more examples, it should start to make more sense.

As always, the Prolog systems searches the knowledge base from top to bottom searching for a clause to match the query. In this case, the first possible match is at clause 28, so Prolog marks this as a backtracking point, and instantiates G to catherine.

This time the rule **grandparent(catherine,Y)** generates 2 sub-goals:

1. **parent(catherine,Z)** and
2. **parent(Z,Y)**

Prolog attempts to solve the first sub-goal: **parent(catherine,Z)**. Searching the knowledge base from top to bottom, it finds the first possible match at clause 16: **parent(catherine,shiona)**. It marks this as a backtracking point, and instantiates Z to shiona.

Returning to clause 28 with Z=shiona, it can now attempt to solve the 2nd sub-goal, which is now **parent(shiona,Y)**. Again searching from the top of the knowledge base, it finds a match at clause 14 by instatiating Y to tanya.

Both sub-goals are satisfied, so the rule **grandparent(catherine,Y)** is true (succeeds) with Y=tanya, so Prolog can report this as its first solution.

If we request more solutions, Prolog will return to the backtracking point at clause 14, and attempt to find another match for the sub-goal parent(shiona,Y). It finds this at clause 15, with Y=pete.

Again, both sub-goals are satisfied, so the system can report Y=pete as a second solution.

If further solutions are requested, Prolog will continue its search from clause 15 down, but finds no further solutions to the second sub-goal. It therefore returns to clause 28, and begins searching for further solutions to the first sub-goal parent(catherine,Z) from its backtracking point at clause 16. It finds a match at clause 17, instantiating Z to sam. It can then attempt to solve the 2nd sub-goal, using Z=sam, and so on .....

The whole method seems cumbersome, but it is completely methodical, and should find all possible solutions to the query. In fact, you should recognise that the Prolog system is using a **depth-first** exhaustive search of the knowledge base (see topic 3).

Prolog search strategy simulation - grandparent(catherine,Who)

20 min

On the Web is a simulation of the Prolog search strategy. You should now look at this simulation.

7.5.4 Order of facts and rules

You should recall that depth-first searching may get into difficulties if a branch of the search tree contains a loop, or the search begins down the "wrong" branch, and so finds a solution which is not the "best" solution. The same problems can arise in Prolog, but can be prevented by careful design of the knowledge base. One design tip is to start with facts, then to follow these with any rules (as we have done in our example). This ensures that, if there is a simple solution to be found among the facts, Prolog will find this before it starts looking for more complex solutions using rules.

7.6 Negation

In Topic 7.4, we came across a problem where the query ?brother(X,Y) gave, as one of its answers, that Pete is the brother of Pete. To avoid this happening, we need to make it clear that X and Y must not be the same person. We do this using **negation**.

7.6.1 Using negation

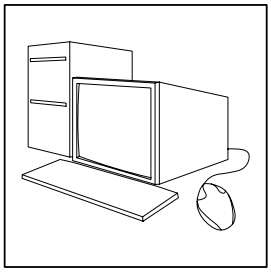
There are two ways of applying negation to this problem. We can either:

- change the rule in the knowledge base, or
- change the query

The next activity will show you how to do both of these.

20 min

Using negation in Prolog



Load the Prolog file Family2 (the one with the complete knowledge base of facts and rules).

Changing the Query

The original simple query was: ?brother(X,Y)

Instead, enter the following complex query: ?brother(X,Y),**not(X = Y)**

This should list all the brothers, without Pete and Pete, or Greg and Greg.

Notice that the system would not work if you entered the 2 clauses the other way around, i.e. not(X = Y),brother(X,Y). This is because when it tries to evaluate the first sub-goal, there are no values for X or Y.

Changing the Rule

The second (better) method is to amend the rule in the knowledge base. Change the brother rule from:

brother(X,Y):-parent(Z,X),parent(Z,Y),male(X).

to

brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),**not(X=Y)**.

Test the new rule by entering the query ?brother(X,Y). Don't forget to re-compile your knowledge base first!

There are 2 other rules which need the same treatment. Can you identify them? If so, make the changes and test that they work.

Check your answers before continuing.

Save your knowledge base as Family3

7.6.2 How Prolog evaluates a Negation

Prolog evaluates a negation in two stages. Consider the negation not(X=Y), where X=pete and Y=pete.

First it evaluates the part in brackets, i.e. pete=pete. Prolog finds this to be true. Hence it finds the negation not(pete=pete) to be false.

Consider the evaluation of not(X=Y) when X=pete and Y=bill.

In this case, Prolog finds pete=bill to be not true, so the negation not(pete=bill) is true. Obvious!

Warning note! The order of the clauses is important. When negation is used in a rule or complex query, the negation has to be towards the end of the rule. This is because Prolog cannot evaluate not(X=Y) before it has found possible values for X and Y.

7.7 Numbers in Prolog

In our example so far, all the facts have been represented as words. Prolog can also handle numerical data.

For example, lets add everyone's age to the knowledge base. There are (at least) 2 ways of doing this. Here's the information:

Catherine is 75, Andy is 72, Bill is 39, Shiona is 37, Sam is 36, Lee is 38, Tanya is 15, Pete is 12, Susan is 6 and Greg is 4. No-one knows how old the dog is!

The first method would be to add a set of facts of the form:

- age(catherine,75).
- age(andy,72).

and so on ...

This would mean adding 10 new clauses.

An alternative is to add this information to the existing male and female clauses, like this:

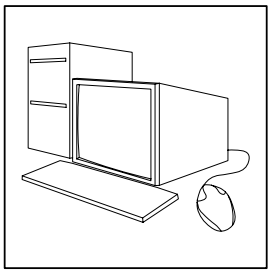
- female(catherine,75).
- male(andy,72).

and so on ...

This second method is certainly more compact, and also more readable. However, it means we have to change rules 26,27,29 and 30, as these rules use the male and female clauses.

20 min

Adding age information to the family knowledge base



Load the latest version of the family knowledge base (Family3).

Edit clauses 1 to 10 to include the ages of the family members, as above i.e. clause 1 becomes female(catherine,75) and so on.

Edit clauses 26,27,29 and 30 as follows:

- brother(X,Y) :- parent(Z,X),parent(Z,Y),**male(X,A)**,not(X=Y).
- sister(X,Y) :- parent(Z,X),parent(Z,Y),**female(X,A)**,not(X=Y).
- granny(G,Y) :-grandparent(G.Y),**female(G,A)**.
- grandpa(G,Y) :-grandparent(G.Y),**male(G,A)**.

Note the use of another variable (A) for the age.

Re-compile the knowledge base, and save it as Family4.

Now try the following queries:

1. to find how old Tanya is:
   ?female(tanya,Age)
2. to find out which males are over 65:
   ?male(Who,Age),Age>65
3. to find out which females are between 10 and 20:
   ?female(Who,Age),Age>10,Age<20

Notice that there is no simple query to answer the question "Who is over 50?". We would need 2 queries: **?male(Who,Age),Age>50** and **?female(Who,Age),Age>50**

If we had used the first suggested representation, this would have been easier. A simple query **?age(Who,Age),Age>50** would then have been possible.

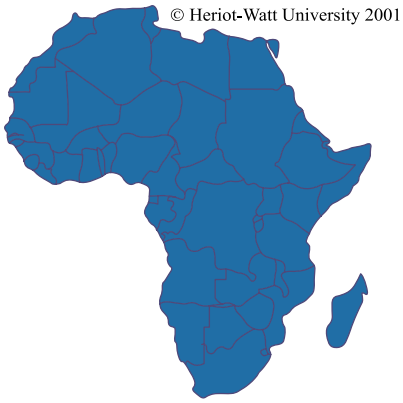
A third possible representation which would have the advantages of both would be to replace all the male and female clauses with "familymember" clauses, each with 3 arguments, like this:

- familymember(catherine,female,75).
- familymember(shiona,female,37).
- familymember(bill,male,39).

and so on ...

There are always many ways of representing the same information in a Prolog knowledge base. Just as in any other form of programming, it is important to take time over **analysis** and design before launching into the **implementation**. The programmer has to consider not only what information is to be represented, but what the most likely queries will be. Taking this into account will help to decide the form of the clauses and their order within the knowledge base.

7.8 Another example (Countries of Africa)



You are probably pretty fed up with the family example we have been working on so far, so here is another example of a Prolog knowledge base, on a completely different topic.

We are going to design and implement a knowledge base containing information about countries in Africa. For each country, we are going to include its name, its capital city, the main language spoken and the population.

Here is some of the information we will include:

name	**capital**	**language**	**population**
Algeria	Algiers	Arabic	22.2 million
Angola	Luanda	Portuguese	7.9 million
Benin	Porto Novo	French	4 million
Botswana	Gaborone	English	1.1 million
Burkina Faso	Ouagadougou	French	6.9 million
Burundi	Bujumbura	French	4.6 million
Cameroon	Yaounde	English	9.7 million

7.8.1 Analysis and design

There are several ways this information could be coded in Prolog. For example, we could list the facts as simple Prolog clauses:

```
country(algeria).
country(angola).
.....

capital(algeria,algiers).
capital(angola,luanda).
.....

language(algeria,arabic).
language(angola,portuguese).
```

.....

population(algeria,22.2).
population(angola,7.9).
.....

This method would work, but it would be very lengthy, especially if the knowledge base was extended to include all 52 countries in Africa. If we decided to include further information (e.g. the main export of each country), we would have to add 52 extra clauses to the knowledge base.

A better method would be to use **multi-argument clauses**, like this:

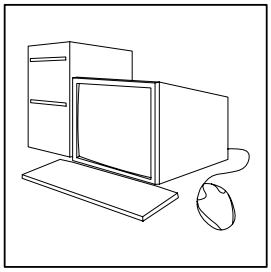
country(algeria,algiers,arabic,22.2).
country(angola,luanda,portuguese,7.9).
country(benin,porto_novo,french,4).
and so on ....

This time, there is one clause for each country, but with 4 arguments. This makes a much more compact knowledge base. The relationship between the items of information is also clearer. However, it could be argued that the knowledge base is less readable, since (for example) it is not clear what each argument represents. The 22.2 for Algeria could represent its population, its area, the average age of its population, or even the number of crocodiles per square kilometre! The programmer would need to include some internal commentary to make the meaning clear.

7.8.2 Implementation

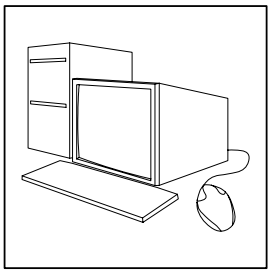
15 min

African countries knowledge base



- load your Prolog system
- key in the knowledge base as above, and extend it to include all 6 countries in the table
- add your name, today's date and an explanation of the format of the country clauses as internal commentary
- compile it
- if it doesn't compile, correct any syntax errors, then compile again
- save it as Africa1
- print out the knowledge base

Querying the african countries knowledge base



20 min

Try each of these queries. Check that it gives the expected answer. Describe how Prolog solves (or evaluates) the query. Use the trace facility to observe Prolog's strategy.

1. You want to find which country has Luanda as its capital, so enter the following query:

?country(X,luanda,_,_)

Notes:

- X is a variable which will contain the answer to your query
- "luanda" is the name of the capital city, not a variable, so no capital letter!
- You aren't interested in the language or population, so use the **unbound variable** (_) for the last 2 arguments

2.You want to find which countries have over 8 million inhabitants, so enter the query:

?country(Which,_,_,Pop),Pop>8

Notes:

- This time we have chosen to call the name of country variable Which rather than X - it makes no difference, but the results of the query are more readable
- We aren't interested in the capital city or language, so use the unbound variable for these
- We need to know if the population is greater than 8, so need a variable (which we have called Pop) to store the population
- the second part of the query will only be true if the value of Pop is greater than 8

Adding rules to the African countries knowledge base

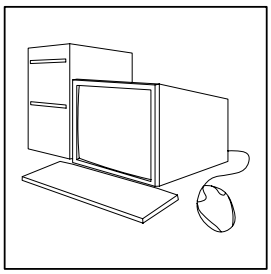
15 min

1. A sales rep is deciding which countries to target as potential markets for his company's products. He decides that only countries with over 5 million inhabitants, and which speak English, are worth considering.

He could get Prolog to find answers by using a complex query, like:

?country(X,_,english,Pop),Pop>5

However, he decides it would be better to write a rule to define countries which meet his criteria.



Load Africa2, and add the following rule:

english_target(E) :- country(E,_,english,Pop),Pop>5.

Re-compile, and query the knowledge base using the simple query **english_target(X)**. Check the answer is what you expected, and use the trace facility to observe Prolog's behaviour.

2. Write a similar rule to define a french_target country as one which speaks French and has a population of over 6 million.

Re-compile, and query the knowledge base using the simple query **french_target(X)**. Check the answer is what you expected, and use the trace facility to observe Prolog's behaviour.

7.9 Summary

- Prolog is a declarative language that is useful in a wide range of AI problems;
- A Prolog program (or knowledge base) is made up of facts and rules;
- These facts and rules are called clauses;
- A fact consists of a predicate and one or more arguments e.g. parent(tom, fred);
- A rule has a goal and one or more sub-goals e.g. brother(X,Y):-parent(Z,X), parent(Z,Y), male(X);
- An argument beginning with a capital letter is a variable;
- To get information out of a Prolog program, the user enters a query e.g. ?parent(X,bill);
- Prolog uses depth-first search to solve queries;
- Prolog searches the knowledge base from top to bottom searching for matching clauses;
- If the goal is matched by a rule, then Prolog attempts to solve each of the sub-goals in turn;
- The process of giving a value to a variable is called instantiation;
- The predicate 'not' can be used in a rule or a query;
- Prolog evaluates a negation by first evaluating the part in brackets, then negating the answer;
- Prolog facts, rules and queries can include numbers e.g. population(algeria,22.2);

- The underscore character (_) can be used as an unbound variable.

7.10 End of Topic Test

An online assessment is provided to help you review this topic.

Topic 8

Prolog (recursion and inheritance)

Contents

Prerequisite knowledge

Before starting this topic, you should know how to write facts, rules and queries in Prolog and understand how Prolog evaluates a query.

Learning Objectives

After studying this topic, you should be able to:

- *Create recursive rules in Prolog;*
- *Explain how recursive rules work;*
- *Draw a semantic net from given information;*
- *Create Prolog rules which involve inheritance;*
- *Explain how inheritance works.*

8.1 Introduction

In Topic 7, you examined some Prolog knowledge bases involving facts and rules, and used queries to extract results from the knowledge base.

In this topic, we will study how rules can be applied recursively. That means that a rule has a sub-goal which can only be solved by using the same rule again. It sounds a little complicated, but it is a very powerful technique that can be used to shorten lengthy programs - rather like using loops in an imperative language.

One of the most important uses of recursion is to implement **inheritance**. Inheritance is a neat idea which can also simplify and shorten otherwise lengthy programs. To help in the design of a knowledge base involving inheritance, we can use a type of diagram called a semantic net (which you may have already met if you studied Intermediate 2 Artificial Intelligence).

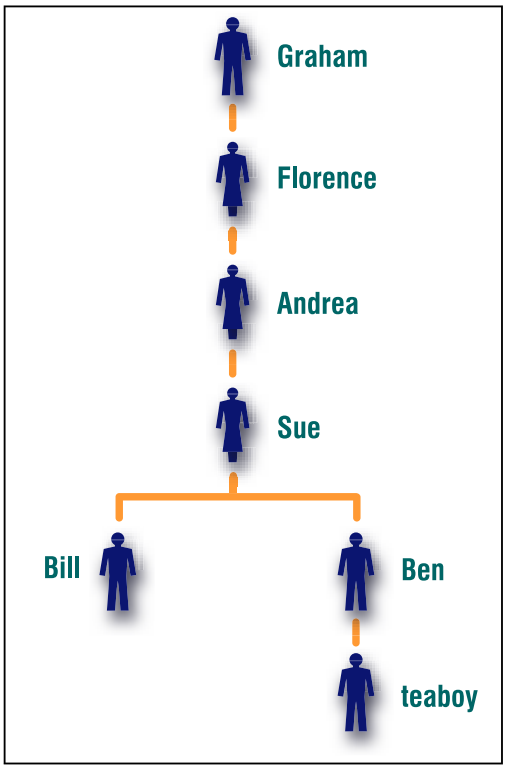
Finally, you can try a practical task, designing, implementing and testing a knowledge base using all the techniques you have learned in these last 2 Topics.

8.2 Recursion

Recursion is a very powerful and useful tool in Prolog programming. Normally, rules call other rules or facts. A recursive rule is a rule which calls itself. The next example will show you how this can be used.

8.2.1 An example using recursion

The situation is a small business, with a hierarchical management structure. Bill and Ben work for Sue. Sue works for Andrea. Andrea's boss is Florence, and Florence works for Graham. The teaboy works for Ben.



We could write a simple Prolog program to represent this information:

```
boss(sue,bill).
boss(sue,ben).
boss(andrea,sue).
boss(florence,andrea).
boss(graham,florence).
boss(ben,teaboy).
```

Suppose we wanted to find out who is above Ben in the office hierarchy. Obviously, his immediate boss is above him, so we might start by writing this rule:

above(X,Y) :- boss(X,Y).

The query **?above(Who,ben)** would give Sue as the answer, but not anyone else, although we know that Andrea, Florence and Graham are also above Ben.

We could add a second rule:

above(X,Y) :- boss(X,A),boss(A,Y).

This time, the query **?above(Who,ben)** would give Sue and Andrea as answers, but would still not give Florence and Graham, as they are more than 2 levels above him.

We would need to add 2 more rules:

above(X,Y) :- boss(X,A),boss(A,B),boss(B,Y).
above(X,Y) :- boss(X,A),boss(A,B),boss(B,C),boss(C,Y).

Obviously, this is a very clumsy method, and needs more and more rules to cover all the possibilities. This is exactly the situation where we can use **recursion**.

We can reduce all these rules into 2 rules, one of which is recursive (i.e. it calls itself).

Here are the 2 rules we could use:

1. **above(X,Y) :- boss(X,Y).**
   my immediate boss is above me
2. **above(X,Y) :- boss(X,A), above(A,Y).**
   the immediate boss of anyone who is above me, must also be above me!

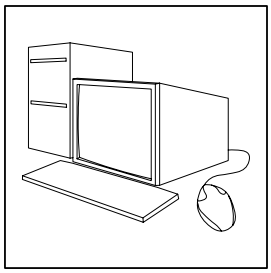
The **first rule** says that X is above Y if X is the direct boss of Y.

The **second (recursive) rule** says that X is above Y if X is above some other person A, who is in turn above Y. This should work no matter how many levels there are between X and Y.

Can you see why?

20 min

Using recursive rules



1. Load Prolog and start a new file.
2. Key in the 6 facts: **boss(sue,bill)** etc.
3. Key in the 2 rules:
   - **above(X,Y) :- boss(X,Y).**
   - **above(X,Y) :- boss(X,A), above(A,Y).**
4. Compile the knowledge base and save it as office1.
5. Run the following queries. For each one, check that it gives the expected answer, and use the trace facility to observe Prolog's behaviour.
   - ?boss(sue,bill)
   - ?boss(bill,sue)
   - ?boss(graham,X)
   - ?above(sue,bill)
   - ?above(bill,X)
   - ?above(graham,X)

Check that your answers are correct before continuing.

8.2.2 Order of clauses in recursion

When using recursive rules, it is very important to get the order of the two rules correct. The simple rule must come first, otherwise the Prolog system will get into an infinite loop and never find a solution.

```
above (X,Y) :- boss (X,Y).
above (X,Y) :- boss (X,A), above (A,Y).
```
correct

```
above (X,Y) :- boss (X,A), above (A,Y).
above (X,Y) :- boss (X,Y).
```
incorrect

Similarly, the 2 clauses within the recursive rule must also be in the correct order: the one which refers to facts must come before the one which refers to itself!

```
above (X,Y) :- boss (X,Y).
above (X,Y) :- boss (X,A), above (A,Y).
```
correct

```
above (X,Y) :- boss (X,Y).
above (X,Y) :- above (A,Y), boss (X,A).
```
incorrect

This is because Prolog evaluates from top to bottom down the knowledge base, and from left to right across a rule. If you find a problem when using recursive rules, check these two points very carefully!

8.3 Inheritance and semantic nets

In two of the examples we have considered so far, the family tree and the office hierarchy, we have used diagrams to represent the information. These diagrams help us to visualise the information, so that we can design a Prolog representation of the relationships.

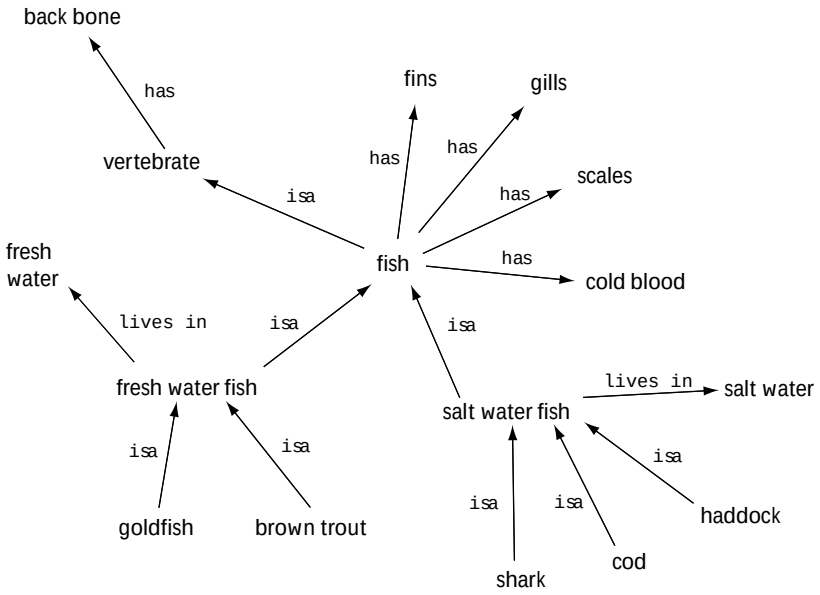
One type of diagram which can be very useful when designing Prolog programs is called a **semantic net**. In fact the two diagrams we have used are actually examples of semantic nets.

8.3.1 Representing information using a semantic net

In this example, we will use a semantic net to represent information about fish. A group of S1 pupils have been asked to write down any facts about fish. Here is what they wrote:

A fish has fins, scales, is cold blooded and has gills. There are lots of kinds of fish, including cod, haddock, shark, goldfish, and trout. A fish is a vertebrate - that means it has a backbone. I have a goldfish called Sammy. Cod, haddock and sharks are salt-water fish (they live in salt water). Goldfish and trout are fresh-water fish (they live in fresh water).

We can represent all this information in a diagram:



8.3.2 Turning a semantic net into Prolog

We can use this semantic net to write a Prolog knowledge base. Each arrow on the semantic net becomes a Prolog clause.

There are 3 different relationships shown in the diagram. These are:

- is a (e.g. Sammy is a goldfish)
- has (e.g. a fish has fins)
- lives in (e.g. a salt water fish lives in salt water)

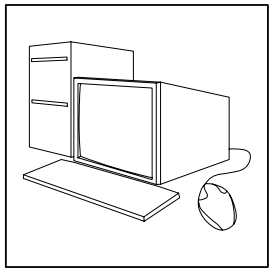
We can use these as Prolog predicates, so the knowledge base would look like this:

```
1. is_a(sammy,goldfish).
2. is_a(goldfish,freshwater_fish).
3. is_a(trout,freshwater_fish).
4. is_a(cod,saltwater_fish).
5. is_a(haddock,saltwater_fish).
6. is_a(shark,saltwater_fish).
7. is_a(saltwater_fish,fish).
8. is_a(freshwater_fish,fish).
9. is_a(fish,vertebrate).

10. has(fish,fins).
11. has(fish,scales).
12. has(fish,cold_blood).
13. has(fish,gills).
14. has(vertebrate,backbone).

15. lives_in(freshwater_fish,freshwater).
16. lives_in(saltwater_fish,saltwater).
```

A fishy knowledge base (part 1)



20 min

- load your Prolog system;
- key in the knowledge base as above;
- compile it;
- if it doesn't compile, correct any syntax errors, then compile again;
- save it as Fish1;
- print out the knowledge base.

Now try the following queries:

1. **?has(fish,X)**
2. **?is_a(sammy,X)**
3. **?lives_in(sammy,freshwater)**
4. **?is_a(sammy,fish)**
5. **?has(sammy,fins)**

Check your answers before continuing.

The answers you get to these queries are technically correct, but they show no signs of intelligence!

The first query gives the answer that fish have fins,scales,cold blood and gills. This is correct, but it doesn't give the answer that fish have a backbone. We can see from the diagram that a fish is a vertebrate. A vertebrate has a backbone, so we (humans) can come to the conclusion that a fish has a backbone.

The second query gives the answer that Sammy is a goldfish. Correct, but we can also see that Sammy is a freshwater fish, that Sammy is a fish, and that Sammy is a vertebrate.

The third query gives the answer that Sammy does not live in fresh water. Prolog only "knows" that fresh water fish live in fresh water, but it cannot jump to the conclusion that Sammy (a goldfish and therefore a fresh water fish) lives in fresh water

The fourth query gives the answer that Sammy is not a fish! Prolog knows that Sammy is a goldfish. It also knows that a goldfish is a freshwater fish, and that a freshwater fish is a fish. Any intelligent system should be able to reach the conclusion that Sammy is therefore a fish, but our knowledge base fails to do so!

Finally, Prolog does not come to the conclusion that Sammy has fins. It "knows" that fish have fins, but it does not "know" that Sammy is a fish (see above!)

We need to add some rules to our knowledge base so that Sammy will **inherit** all the

properties of fish (and all the other groups he belongs to). We do this using some **recursive rules**.

8.3.3 Inheritance rules

We could extend the knowledge base by adding extra facts, like:

has(sammy,fins).
has(sammy,gills).
lives_in(sammy,freshwater)
is_a(sammy,fish).
and so on ..

However, this would make our knowledge base very long and repetitive. What we need is a rule to say that if Sammy is in a particular group, then he should have all the properties of that group. In fact, not just Sammy, but any sub-group should inherit the properties of any group of which it is a member.

All we need to do is to add the following recursive rules:

has(X,Y) :- is_a(X,Group), has(Group,Y)

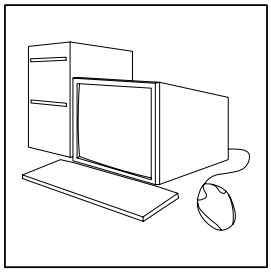
This means X has Y if X is in group G, and G has Y. For example, Sammy has fins if Sammy is a fish and fish have fins!

lives_in(X,Y) :- is_a(X,Group), lives_in(Group,Y)

This means X lives in Y if X is in group G, and G lives in Y. For example, Sammy lives in fresh water if Sammy is a fresh water fish and fresh-water fish live in fresh water!

20 min

A fishy knowledge base (part 2)



- load the Fish1 knowledge base;
- add the 2 inheritance rules (above);
- compile it;
- if it doesn't compile, correct any syntax errors, then compile again;
- save it as Fish2;
- print out the knowledge base.

Now repeat the following queries:

1. **?has(fish,X)**
2. **?is_a(sammy,X)**
3. **?lives_in(sammy,freshwater)**
4. **?is_a(sammy,fish)**
5. **?has(sammy,fins)**

Check your answers before continuing.

The inheritance rules are now working correctly, and Sammy correctly inherits all the properties of freshwater fish, fish and vertebrates.

Queries 1, 3 and 5 now give intelligent answers, but 2 and 4 do not.

However, the 2 queries using the relation **is_a** still don't work as we would want them to. To correct this we need to add one more rule:

The obvious format for this rule is:

is_a(X,Y) :- is_a(X,Group),is_a(Group,Y).

Add this rule to your knowledge base, recompile it, and run the queries again. You should now get all the answers you expect. Unfortunately, Prolog now gets into an infinite loop!

To avoid this problem, we need to define a new predicate (which we could call is_in, which is defined recursively using 2 clauses:

1. **is_in(X,Y):-is_a(X,Y).**
2. **is_in(X,Y):-is_a(X,Group),is_in(Group,Y).**

Try replacing the last rule with these two rules, and test the queries once again!

8.4 Practical task

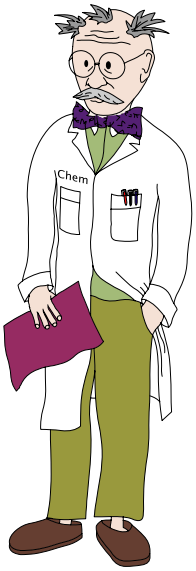
The practical assessment for this Unit requires you to develop a Prolog knowledge base. As in any other type of software development, you need to provide documentation as evidence that you have completed the task. You should provide the following evidence:

- a description of the problem;
- representation of the domain knowledge as a semantic net;
- representation of the domain knowledge in Prolog;
- implementation of the knowledge base;
- evidence of testing the solution;
- evaluation.

Here is a worked example to show you what is required.

80 min

The noble gases



A chemistry teacher wants a knowledge base to represent information about some chemical elements, so that pupils can use it to find out information about these chemicals.

The "noble gases" are a group of elements. They are all very unreactive, and (as their name suggests) they are all gases at room temperature. The noble gases are Argon, Xenon, Neon, Krypton and Radon.

Argon (symbol Ar) is colourless, has atomic number 18, and is used to fill ordinary light bulbs. Xenon (Xe) is also colourless and has atomic number 54. Neon (Ne) has atomic number 10, and is used in strip lights. Krypton (Kr) has atomic number 36 and is also colurless. Radon (Rn) is a radioactive gas, and has atomic number 86.

1. Problem Description

Write a couple of sentences describing the problem in your own words, and stating the purpose of the program, including the types of questions it should be able to answer.

Write down any boundaries or limitations of the knowledge base. Make it clear what is included, and what is not included.

Check your answer before continuing.

2. Representing the Domain Knowledge as a Semantic Net

From the task described, you have to select the information that you are going to include in your knowledge base. Identify the objects and their characteristics. Are there any groups whose members can inherit group characteristics? Once you have sorted this out, you should represent the knowledge as a semantic net (or some other representation).

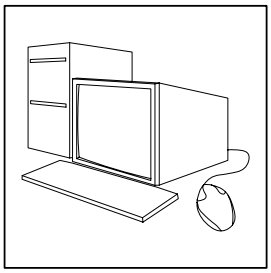
Check your answer before continuing.

3. Representing the Domain Knowledge in Prolog

Use your semantic net to represent the facts as Prolog clauses. Remember to include an inheritance rule so that all the elements inherit the general properties of the noble gases.

Check your answer before continuing.

4. Implementing the Knoweldge Base



You are now ready to enter your program into a Prolog system. Compile it to ensure there are no syntax errors. If there are any errors, correct them, then save and print your program.

5. Testing the Knowledge Base

You know the syntax of your program is correct, because you could compile it without any errors. Now you must test it to see if it is logically correct - that is, it gives the expected answer to any queries.

You should make a list of queries to test all aspects of the knowledge base, then summarise your results. Write down the answers given to all these queries, and comment if there are any surprising results. You should provide evidence for some of the queries by printing the output window. If you are not sure how to do this, ask your tutor.

6. Evaluating the Knowledge Base

Your evaluation should make an honest attempt to answer these questions:

- Does your knowledge base fulfil the specification?
- Could another solution have been found?
- Is your solution efficient?
- Could your solution be improved in any ways?
- Are there any moral, ethical or legal issues associated with your solution?

Try writing answers to these 5 questions - a couple of sentences for each - then check with the expected answers.

8.5 Summary

- A recursive rules is a rule which calls itself;
- To implement recursion requires two rules: the first rule is required to stop the recursive rule being called an infinite number of times, leading to an endless loop;
- A semantic net is a diagram which shows all the relationships in a knowledge base;
- Each arrow in the semantic net can be translated into a Prolog clause;
- Recursive rules can be used to make objects inherit the properties of a group.

8.6 End of Topic Test

An online assessment is provided to help you review this topic.

Glossary

Alan Turing

Alan Turing (1912 - 1954). An English mathematician, famous for his work in breaking German u-boat codes during World War II. This led into the development of some of the earliest computers at Manchester University. Most famous for describing the Turing Test for artificial intelligence.

Algorithm

An algorithm is the set of steps that a computer must follow to accomplish a task or solve a problem. An algorithm is often written in pseudocode.

Analysis

Analysis is the first stage in the software development process. During analysis, a program or system specification is established by negotiation between the software developer and the client.

Axon

An axon is the output part of a neuron. The axon is a long spindly extension whcih can pass signals on to other neurons.

Back-propagation

Back-propagation is a technique used to train neural networks by modifying the strength of the links between artificial neurons in the network to bring the actual output more into line with the expected (or correct) output.

Backtracking point

A backtracking point is a marker left by the Prolog system to mark a point in a Prolog program to which it may need to return later to resume a search.

Backward rules

A rule in a knowledge base of the form conclusion IF condition(s) , e.g. go to the beach IF the sky is blue AND the month is July

Boolean logic

Boolean logic (named after mathematician George Boole (1815 - 64)) is a set of rules which allows conclusions to be draw from statements which are either true or false.

Breadth-first search

Breadth-first search is a search strategy which searches all nodes of a search tree at the first level before moving on to the second level, and so on.

Carel Capek

Czech playwright who wrote the play "RUR" in 1920, and first used the term robot to describe a mechanical labourer.

Certainty factor

A certainty factor is a number assigned to a fact or rule in an expert system to define how certain it is that the fact or rule is true. The certainty factor may be

expressed as a fraction or as a percentage. For example: if it is cloudy, it will rain within 1 hour (0.7); if I am in Scotland, it is raining right now (32%).

Chatterbot

A chatterbot is a program that simulates human conversation

Combinatorial explosion

Combinatorial explosion is the tendency of most search-based problems to quickly develop an unmanageably large number of states after a small number of operations. For example, in chess, there are over 400 trillion possible positions after 6 moves.

Computer vision

Computer vision is the process of enabling a computer system to be able to "understand" or respond to its environment through data received from a camera or other optical device.

Declarative language

A declarative language is one which is used to give an exact description of all the relevant facts and relationships which define the problem, rather than an algorithm to solve it. Prolog is an example of a declarative language.

Dendrite

A dendrite is the input part of a neuron. Neurons may have many dendrites which receive signals from other neurons.

Depth-first search

Depth-first search is a search strategy which searches first down a single branch of a search tree until it finds a solution or comes to a dead end.

Design

Design is the second stage of the software development process, following analysis, and preceding implementation. During the design phase, the specification is used to plan out the overall structure and the required modules, and to identify the data objects involved, and the operations which will be required.

Digital sampling

Digital sampling is the process of converting an analogue electrical signal (e.g. the output from a microphone) and converting it into a stream of digital data by measuring the analogue value at regular time intervals.

Domain expert

The domain expert is the human expert who provides the information which is programmed into an expert system.

Eliza

Eliza was one of the first computer programs (1966) to attempt to provide natural language processing. However, Eliza only simulates a non-directive therapist, and so is designed to respond in a non-directive way to human input. Many versions of Eliza have been developed over the last 40 years

Exhaustive search

Exhaustive search refers to any search method (such as depth-first or breadth-first) which searches every node of a search tree in a methodical way.

Expert System

An Expert System is a program, consisting of a user interface, and inference engine and a knowledge base. The knowledge base contains expert knowledge on a topic (or domain), provided by one or more domain experts. The user interface allows the user to interact with the system, which can provide expert advice to the user.

Expert system shell

An expert system shell is an expert system with no knowledge base. It consists only of a user interface and an inference engine.

Explanatory interface

The user interface of an expert system or shell with a built-in ability to explain and justify reasoning using How? and Why?

Forward rules

A rule in a knowledge base of the form IF condition(s) THEN conclusion, e.g. IF the sky is blue AND the month is July THEN go to the beach

Frank Rosenblatt

Professor at Cornell University. Rosenblatt is famous for inventing the Perceptron in 1962. The Perceptron was the first artificial neuron, and so the building block of artificial neural networks.

Fuzzy logic

Fuzzy logic is a formal set of rules which allow conclusions to be drawn from facts that are uncertain.

Heuristics

Heuristics are rules which are not derived purely from logic, but from the experience of a person. They are often called "rules of thumb". They are usually true, but not always.

Hidden layers

Any layers of artificial neurons between the input layer and the output layer of a neural network are called hidden layers.

Humanoid

A humanoid robot is one which resembles the human form. Currently most robots are **not** humanoid.

Implementation

the third stage in the software development process, following analysis and design, when the design is converted into a program (or set of programs) in a particular programming language.

Inference engine

The inference engine is the part of an expert system which searches the knowledge base, generates questions to the user, and uses the facts and rules in the knowledge base to come to conclusions which can be reported to the user.

Inheritance

A technique used to simplify and shorted a knowledge base using rules. An inheritance rule ensures that all objects in a group have the properties assigned to the group.

Instantiation

Instantiation is the process, in Prolog, where a variable is temporarily assigned a value.

Isaac Asimov

Isaac Asimov (1920 - 92), a Russian by birth, although brought up in the USA. Asimov was a Professor of biochemistry, before becoming a full-time writer. He wrote over 500 books, both fiction and non-fiction, and is most famous for this science fiction novels, which explore many of the questions (philosophical, practical and ethical) which would arise from the development of intelligent robots.

John McCarthy

John McCarthy (1927 -), professor at Stanford University, known as the founding father of Artificial Intelligence, after he called the Dartmouth Conference of academics interested in AI in 1956. Invented the AI language LISP.

Joseph Weizenbaum

Weizenbaum was the inventor of Eliza, one of the first programs to attempt natural language processing.

Knowledge base

The knowledge base is the part of an expert system which contains all the information about the expert system's domain, represented as facts and rules in a knowledge representation language.

Knowledge engineer

The knowledge engineer is the person who takes the information obtained from one or more domain experts, and converts it into a knowledge representation language so that it can become part of an expert system.

Knowledge processing language

A knowledge processing language is one which, like Prolog, is designed for the manipulation of large amounts of related facts and relationships.

Linguistics

Linguistics is the scientific study of language.

LISP

LISP is an interpreted functional language, developed in 1959, and widely used in artificial intelligence. Its name comes from LISt Processing.

Marvin Minsky

Marvin Minsky (1927 -), a very influential thinker in the development of AI. Currently Toshiba Professor of Media Arts and Sciences, and Professor of Electrical Engineering and Computer Science at MIT.

Multi-layer network

A multi-layer network is a neural network consisting of at least 3 layers of artificial neurons. These are the input layer, the output layer, and one or more hidden layers.

MYCIN

MYCIN was one of the earliest successful expert systems. Developed in 1979, its domain was diagnosing blood infections.

Neural network

A neural network is a computing device (either constructed in hardware, or a software model) consisting of many interconnected artificial neurons. It is modelled on the physical structure of the human brain, but the number of neurons is much less. Neural networks show a surprising ability to learn and make "intelligent" decisions.

Neuron

A neuron is a cell which forms part of the nervous system or brain of a human or animal. The human brain consists of billions of interconnected neurons.

NLP

NLP (natural language processing) is concerned with systems which can accept input in the form of human language (either written or spoken), and respond in an appropriate manner.

Parallel processing

In parallel processing, a task is divided up into many sub-tasks, which can be processed simultaneously on many processors.

Parry

Parry (1971) was a program written to model the conversation of a paranoid person. It was one of the significant early developments in natural language processing.

Pattern matching

Pattern matching is the ability of a computer system to classify input data by comparing it to known data. Successful pattern matching should permit a system to recognise an object (or other data) which does not match any know example exactly.

Perceptron

The perceptron was the first artificial neuron, developed by Frank Rosenblatt in 1962.

Personal computer

A personal computer (PC) is a microcomputer (desktop or laptop) designed to be used by a single user.

Phonemes

Phonemes are the smallest identifiable units of sound in a language. They may be represented by a single letter or a group of letters.

Primal sketch

A primal sketch is a representation of an object, simplified to identify only the edges where one surface meets another. One method of creating a primal sketch from an image is to use the Waltz algorithm.

Procedural language

A procedural (or imperative) language is one where the program consists of instructions to the computer which will solve the problem. Examples include Pascal, C, BASIC, Fortran and Algol.

Prolog

Prolog is a declarative or knowledge processing language, used for developing knowledge bases (of facts and rules), which can then be queried by the user. Its name derives from PROgramming in LOGic, and it was developed by Colmerauer and associates at the University of Marseilles in the 1970s

Prospector

Prospector was one of the early successful expert systems. Developed in 1982, its domain was identifying sites for the extraction of minerals.

Psychology

Psychology is the science of the mind; the study of mind and behaviour.

Query

A query is a question in Prolog which results in the system drawing a conclusion from the facts and rules of the Prolog program.

Recursion

Recursion is the process when a rule calls itself.

Robot

A robot is a computer-controlled mechanical device which is flexible enough to be able to be programmed for a variety of tasks.

Robotics

Robotics is the study and design of robots.

Search tree

A search tree is a graphical representation of all the possible states of a system showing the operations required to move from one to another. The starting state is usually shown at the top of the diagram, with branches spreading out below,

Semantic net

A semantic net is a diagrammatic representation of objects and their relationships. It is a useful tool in the design of knowledge based programs.

Sensor

A sensor is an input device which responds to some physical property such as pressure, temperature or light.

SHRDLU

SHRDLU (1970), written by Terry Winograd, was a crucial program in the development of natural language processing. Unlike Eliza, SHRDLU used an internal representation of its world, so that it could respond "with understanding" to commands. It operated in the limited domain of a "block world".

Terminal

A terminal is a device consisting of a monitor and keyboard, which allows the user to communicate electronically with a computer. If the terminal has no processing power of its own, it is known as a dumb terminal.

Terry Winograd

Terry Winograd was the writer of the early natural language program, SHRDLU.

Turing Test

The Turing Test was first described by mathematician Alan Turing. It involved a researcher communicating with a computer via a terminal. If the researcher was unable to determine whether he was communicating with a computer or another human, then the computer had passed the test, and could be considered to be intelligent. No computer system has yet passed the Turing Test.

Unbound variable

An underscore (_) is used in a Prolog query to indicate a variable whose value is not required.

User interface

The user interface is the part of an expert system which allows the user to interact with the system.

Von Neumann computer

A Von Neumann computer is a computer which has a single control unit, which fetches instructions from memory and executes them, in a linear way, i.e. one after another. The Von Neumann architecture is the basis for most "normal" computer systems at present.

Working memory

The RAM used by an expert system during consultation, in which it stores facts and rules that it has deduced.

XCON

XCON was one of the earliest successful expert systems. Developed in 1980, its domain was configuring mainframe computers. It saved DEC millions of dollars by providing expert advice to otherwise inexpert salesmen.

Answers to questions and activities

1 What is Intelligence?

Answers from page 7.

Q1:

- The subject is always developing and changing.
- Many different branches of science are involved (e.g. computer science, biology, psychology, philosophy) which all have different perspectives.
- There are many different views on the definition of natural intelligence.

Q2: There are many aspects to intelligence, and there is no universal agreement about what it means.

Q3:

- ability to learn (e.g. the more you play chess, the better you get)
- creativity (e.g. writing a story)
- reasoning (e.g. being able to explain why the sky is blue)
- use of language (e.g. being able to speak, read or write)
- vision (e.g. being able to recognise a friend's face in a crowd)
- problem solving (e.g. how do I get into my house if I have lost the key)
- adapting to new situations (e.g. finding my way around in a place I have never visited before)

Answers from page 9.

Q4: In the Turing Test, an operator uses a terminal to communicate with a system being tested by asking any questions and observing the responses. If unable to tell whether the responses are coming from a machine or a human, then the system being tested can be described as intelligent.

Q5:

- If it passed the test, it was impossible to distinguish between the computer and a human, so the computer must be as intelligent as the human
- All the test shows is that the computer behaves in a way that *appears* intelligent. It does not necessarily mean that it actually is intelligent.

2 The Development of AI

Answers from page 15.

Q1: The first commercial computer, UNIVAC, was developed. Alan Turing proposed the famous Turing Test. (Also Isaac Asimov published his influential story "I, Robot")

Q2: John McCarthy called together researchers interested in AI at Dartmouth College for a conference. It was at this conference that the term AI was first used.

Q3: To develop computers and programs in the likeness of humans.

Q4: Games are manageable problems; they consist of limited domains (clearly defined logical rules, limited number of options at each stage) which were not too complex.

Q5: Game playing programs may show the ability to learn and adapt, and to carry out logical reasoning.

Q6: No matter how good the program was at playing chess, it could not give intelligent responses to any other topic (e.g. What colour are the walls of the room?)

Q7: A "von Neumann" computer uses a single processor working step by step through a program, whereas a neural network consists of many interconnected (and relatively) simple processors (neurons) and does not follow a step by step program.

Q8: The computer hardware of the time was very limited: processors were very slow, so programs took a long time to run; memory was limited, so there was a limit to how complex programs and data could be; computers were expensive and only existed in a few big research centres, so it was difficult for researchers to get access to them.

How does Eliza work? (page 18)

Q9: No, Eliza has no understanding. It simply responds to your input in a pre-programmed way. You can easily fool Eliza using a sentence like "I wish I had a dfhjkdgkjshd". Eliza may respond with "Why do you wish you had a dfhjkdgkjshd", clearly showing it has no understanding.

Q10: It is much easier to write a program that simply responds to inputs, rather than one which takes the initiative in a conversation.

Q11: No. Although it can respond to statements on any topic which you choose, it is easily fooled into making a response which no intelligent human would make.

Answers from page 19.

Q12: Eliza, Parry and SHRDLU

Q13: SHRDLU operated in simple block world (of coloured blocks, like a child's set of building bricks)

Q14: SHRDLU had a method of representing knowledge, so that it did not simply respond mechanically.

Q15: Prolog gave researchers a tool for representing knowledge. PCs started to become available, so researchers started to have access to their own PC instead of having to negotiate access to shared mainframe systems.

3 Vision, language and movement

Questions (page 37)

Q1:

	formal language	natural language
example	BASIC	English
vocabulary	a few hundred words	millions of words
meanings of word	clearly defined and unique	often ambiguous (many words with same meanings, words with multiple meanings)
grammar / syntax rules	clearly defined rules	illogical, imprecise, often broken
development	does not change	always changing

Q2: There are many possible answers! You could have said:

1. the word "cyberspace" has only appeared recently
2. do you know what a "spoff" is? - it is a word heard often in my school, but children in the neighbouring town have never heard of it!
3. a "bank" can be the side of a river, a place where your money is kept safe, a turning manoeuvre of an aircraft
4. your English teacher might argue that rules should never be broken, but we all do it all the time (especially in speech), so an NLP computer system has to be able to cope!
5. too and two; their and there; right and write.

Answers from page 39.

Q3: Hints:

1. Who was carrying the binoculars - me or the man I saw?
2. Who was aggressive - the teacher or Gary?
3. I also know Margaret Thatcher .... or Tony Blair knows 253 politicians, but I know 374!

Answers from page 39.

Q4: If people eat fat, it accumulates on their bodies. The man who goes out hunting stags always has an excuse for avoiding his responsibilities at weekends.

5 Expert Systems

Review questions (page 68)

Q1: The User Interface allows the user to input information in response to questions generated by the system. The user interface is also able to give advice and (very importantly) explain why it is giving that advice.

Q2: The knowledge base consists of all the facts and information about the topic, and all the rules and relationships between these facts

Q3: The inference engine searches the knowledge base, generates questions to the user, and uses the rules and relationships which have been programmed into the knowledge base to come to conclusions which can be reported to the user.

Q4:

a) Jack McConnell (or some other MSP or political expert)

b) Martin O' Neill, Alex Mcleish, Alex Ferguson or any other footballing expert

c) your teacher (maybe?)

d) anyone rich and famous!

Q5: An expert system shell consists only of a user interface and an inference engine - it has no knowledge base. An expert system consists of all 3 parts.

Who is to blame? (page 75)

The following people could be considered responsible in some way:

- the **domain expert** who provided the information that was used to create the knowledge base;
- the **knowledge engineer** who developed the knowledge base from the domain expert's input;
- any other **programmers** involved in developing or testing the user interface or inference engine of the expert system;
- the **software company** which sold the expert system to the user;
- the **user** who accepted the advice given by the expert system.

6 Search-based Problem Solving

Identifying start and goal states (page 78)

- start state: current position in building;
- goal state: its destination in the building;
- set of operation: the movements it can make, and any rules about where it can or cannot go.

Answers from page 79.

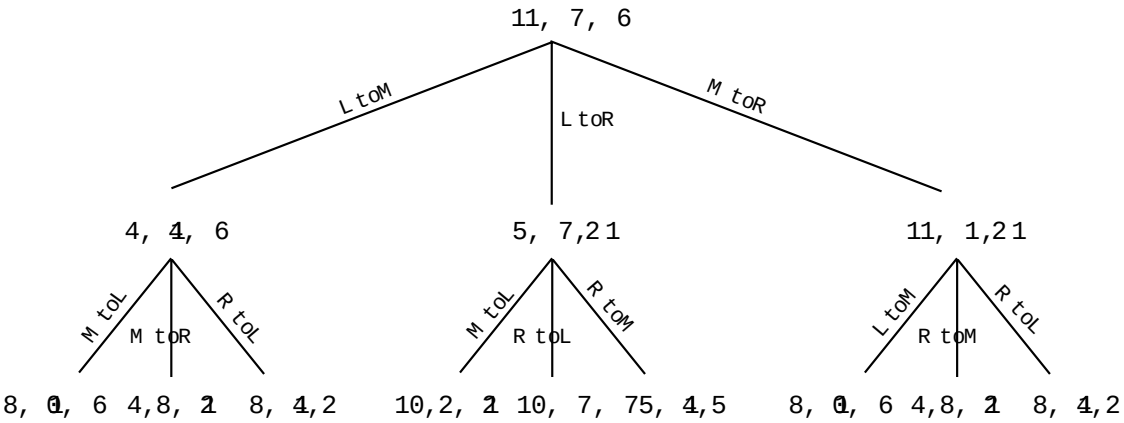
Q1:

1. move 7 matches from the left pile to the middle pile.
2. move 6 matches from the left pile to the right pile
3. move 6 matches from the middle pile to the right pile

Q2: RtoM would need 7 matches to double the number in the middle pile, but there are only 6 in the right pile, so it is not possible. Similarly, RtoL and MtoL would need 11 matches moved, and there are not enough matches in the middle or right piles to complete the move,

Expanding the search tree for the matches problem to the second level (page 80)

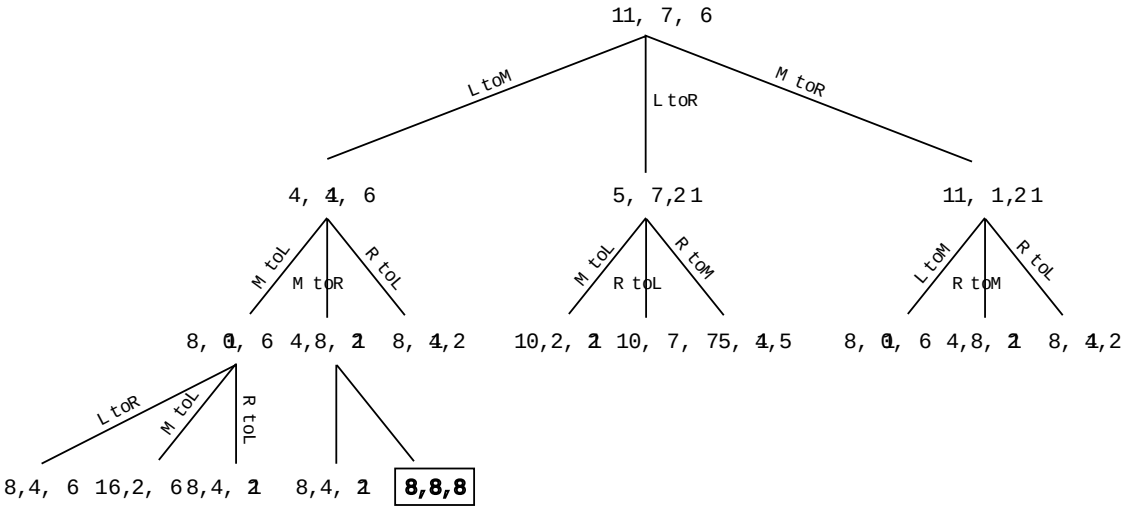
You search tree should now look like this:



After 2 moves, there are 9 possible states. None of these is the goal state (8,8,8), so we need to consider the next level in the search tree.

Expanding the search tree for the matches problem to the third level (page 80)

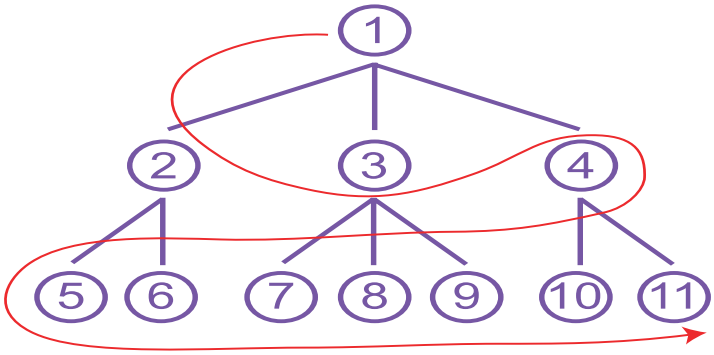
You search tree should now look like this:



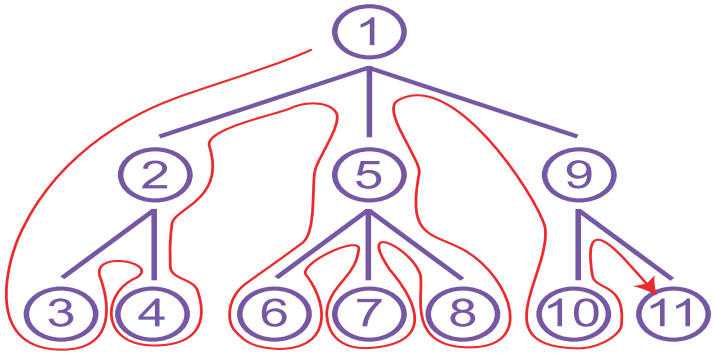
After 3 moves, there are actually 27 possible states (although you will see that some of them are duplicates, i.e. the same state but reached by a different set of operations). One of these is the goal state (8,8,8), so we stop expanding the search tree.

Review questions (page 83)

Q3: Your search tree should look like this:



Q4: Your search tree should look like this:



Q5: a) breadth-first

Q6: b) depth-first

Q7: b) depth-first

7 Prolog (facts, rules and queries)

Analysis (page 92)

Your answer might look like this:

1.
   - Catherine is female;
   - Shiona is female;
   - Tanya is female;
   - Sam is female;
   - Susan is female.
2.
   - Bill is male;
   - Lee is male;
   - Greg is male;
   - Andy is male;
   - Pete is male.
3. Haggis is a dog
4.
   - Bill is parent of Tanya;
   - Bill is parent of Pete;
   - Shiona is parent of Tanya;
   - Shiona is parent of Pete;
   - Catherine is parent of Shiona;
   - Catherine is parent of Sam;
   - Andy is parent of Shiona;
   - Andy is parent of Sam;
   - Sam is parent of Susan;
   - Sam is parent of Greg;
   - Lee is parent of Susan;
   - Lee is parent of Greg.
5. Susan is the owner of Haggis.

Querying the knowledge base (1) (page 95)

1. ?male(lee) ..... yes
2. ?male(sam) ..... no

3. ?male(X) ..... X=bill, X=lee, X=greg, X=andy, X=pete
4. ?parent(sam,greg) ..... yes
5. ?parent(sam,tanya) ..... no
6. ?parent(sam,Child) ..... Child=susan, Child=greg
7. ?parent(bill,X) ..... X=tanya, X=susan
8. ?parent(P,pete) ..... P=bill, P=susan
9. ?owns(X,haggis) ..... X=susan
10. ?child(greg,sam) ..... Prolog cannot answer this question - why not?

Querying the knowledge base (2) (page 95)

?parent(X,tanya),female(X)

Adding rules (page 96)

- child(X,Y) :- parent(Y,X).
- brother(X,Y) :- parent(Z,X), parent(Z,Y), male(X).
- sister(X,Y) :- parent(Z,X), parent(Z,Y), female(X).
- grandparent(G,Y) :- parent(G,Z), parent(Z,Y).
- granny(G,X) :- grandparent(G,X),female(G)
- grandpa(G,X) :- grandparent(G,X),male(G)
- grandchild(X,Y) :- grandparent(Y,X)
- cousin(X,Y) :- grandparent(G,X), grandparent(G,Y).
- aunt(A,X) :- parent(P,X), sister(A,P).
- uncle(U,X) :- parent(P,X), brother(U,P).

Notes:

1. If you have alternative answers that you think are correct, get your tutor to check them for you. As in most programming situations, there is more than one way to solve the problem. For example, an alternative to the single cousin rule could be these two rules:

cousin(X,Y) :- parent(P,X), parent(Q,Y),brother(P,Q)

cousin(X,Y) :- parent(P,X), parent(Q,Y),sister(P,Q)

2. When you tested the brother rule, you may have noticed that it gives X=pete, Y=pete as an answer. According to our definition Pete is Pete's brother, because both Pete and

Pete have the same parent! This seems silly to us, but the computer only knows what we have put into the knowledge base. In Topic 4.4, we will see how to eliminate this type of problem.

3. If you tested the query granny(X,Y), you would get 4 answers:

X=catherine,Y=tanya

X=catherine, Y=pete

X=catherine, Y=susan

X=catherine, Y=greg

but if you only wanted to find out who is a granny (but don't care who their grandchildren are), you can use the **unbound variable**, which is the underscore (_) character.

The query would then be: ?granny(X,_), giving the answer X=catherine.

Try it!

Using negation in Prolog (page 102)

You will need to make the following changes:

sister(X,Y):-parent(Z,X),parent(Z,Y),female(X),**not(X=Y).**

cousin(X,Y):-grandparent(G,X),grandparent(G,Y),**not(X=Y).**

Querying the african countries knowledge base (page 107)

1. X=angola

Prolog searches the knowledge base from top to bottom to find a match for the query. It finds a match at clause 1, instantiating X to angola. There are no other matches.

2. Which = algeria , Pop = 22.2
Which = cameroon , Pop = 9.7

Prolog searches the knowledge base from top to bottom for a match to the first part of the query. It finds a match at clause 1, instantiating Which to angola, and Pop to 22.2, and marks clause 1 as a backtracking point. It then evaluates the second part of the query with Pop=22.2. $22.2 > 8$ so the second part is true (succeeds). Prolog reports a solution Which=algeria, Pop=22.9

If we request further solutions, Prolog continues from the backtracking point at clause 1. Clause 2 gives a possible match with Which=angola and Pop=7.9 However, the 2nd half of the query fails since it is not true that $7.9 > 8$. Prolog continues to search, checking each clause in the same way.

8 Prolog (recursion and inheritance)

Using recursive rules (page 114)

- ?boss(sue,bill) ..... yes
- ?boss(bill,sue) ..... no
- ?boss(graham,X) .....X=florence

- ?above(sue,bill) ..... yes
- ?above(bill,X) ..... no
- ?above(graham,X) ..... X=andrea, X=sue, X=bill, X=ben, X=teaboy

A fishy knowledge base (part 1) (page 117)

1. ?has(fish,X) ..... X = fins ; X = scales ; X = cold_blood ; X = gills
2. ?is_a(sammy,X) ..... X = goldfish
3. ?lives_in(sammy,freshwater) .... no
4. ?is_a(sammy,fish) ..... no
5. ?has(sammy,fins) ..... no

These 4 answers are not what we might expect.

A fishy knowledge base (part 2) (page 118)

1. **?has(fish,X)** ..... X = fins ; X = scales ; X = cold_blood ; X = gills ; **X = backbone** ... Prolog now adds the answer X=backbone to the original list of answers
2. **?is_a(sammy,X)** ..... X = goldfish ... this is still the only answer given by Prolog
3. **?lives_in(sammy,freshwater)** ... now Prolog correctly answers **yes**
4. **?is_a(sammy,fish)** ... Prolog still answers **no** to this query.
5. **?has(sammy,fins)** ... Prologs now correctly answers **yes**

The noble gases (page 120)

1. Problem Description

The problem is to represent information about a group of chemical elements - the noble gases. This information includes some physical properties, the atomic number, the symbol and the uses of each element. It should be able to answer quesions like:

- what is the atomic number of Argon?

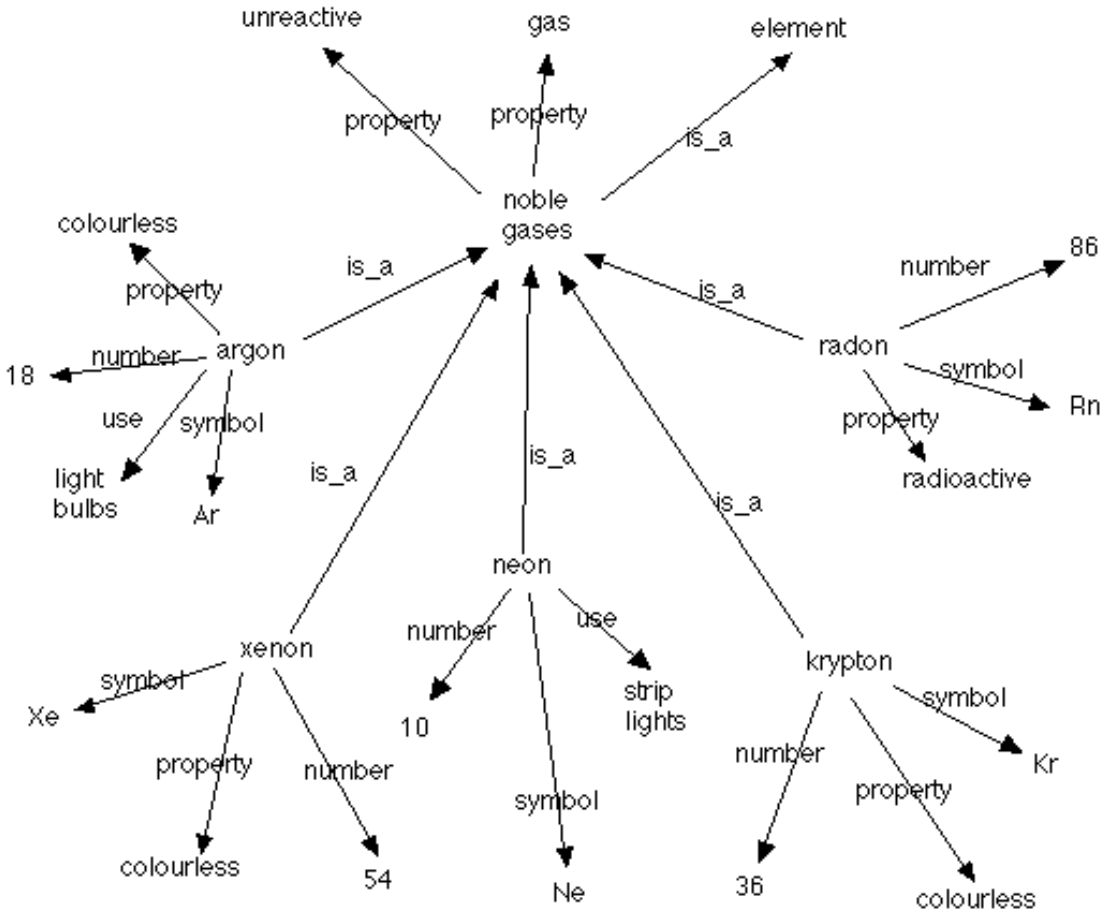
- which element has the symbol Xe?
- what are the physical properties of Neon?

The knowledge base will only contain the information given in the task. It will not contain any extra information about these elements, and will not contain information about any other elements.

2. Representing the Domain Knowledge as a Semantic Net

The objects will be the elements. Each element has 5 properties - its name, atomic number, symbol, description and uses. The elements will inherit the properties of the group (unreactive).

Semantic Net



3. Representing the Domain Knowledge in Prolog

I have chosen to represent the knowledge using simple clauses. The facts about each element are represented using the predicates is_a, property, number, symbol and use. I have used an inheritance rule so that each element inherits the properties of the whole group.

```
is_a(noble_gas,element).

is_a(argon,noble_gas).
is_a(xenon,noble_gas).
```

```
is_a(neon,noble_gas).
is_a(krypton,noble_gas).
is_a(radon,noble_gas).

property(noble_gas,gas).
property(noble_gas,unreactive).

property(argon,colourless).
property(xenon,colourless).
property(krypton,colourless).
property(radon,radioactive).

number(argon,18).
number(xenon,54).
number(neon,10).
number(krypton,36).
number(radon,86).

symbol(argon,ar).
symbol(xenon,xe).
symbol(neon,ne).
symbol(krypton,kr).
symbol(radon,rn).

use(argon,light_bulbs).
use(neon,strip_lights).

property(X,Y):-is_a(X,Group),property(Group,Y).
```

You may have a different representation of the facts. Don't worry - there are lots of different ways of representing this knowledge in Prolog. If you are not sure whether your representation is correct, ask your tutor to check it before you go any further.

5. Testing the Knowledge Base

The knowledge base was tested using the following queries:

QUERY	RESULT	COMMENT
?- number(argon,A)	A = 18	as expected
?- symbol(Element,xe)	Element = xenon	OK, but had to use xe not Xe
?- property(neon,X)	X = gas	inherited properties of
	X = unreactive	noble gases
?- property(radon,X)	X = radioactive	own property
	X = gas	inherited property
	X = unreactive	inherited property

The results show that the knowledge base is operating as expected.

6. Evaluating the Knowledge Base

- **Does your knowledge base fulfil the specification?** The knowledge base

correctly represents all the factual information provided in the specification. It can be queried to find answers to questions about the properties, name, symbol, atomic number and uses of all the noble gases.

- **Could another solution have been found?** There are several other ways of coding this information in Prolog. For example, I could have used multi-argument clauses, like element(argon,ar,53,colourless,light_bulbs). This would have taken less space but would have been less readable. Another possible solution would have been to put the information into a simple database or spreadsheet, but this would not have been so easy to query.

- **Is your solution efficient?** The solution could have been made more efficient by using multi-argument clauses (see above). This would have required less memory, and led to quicker searching for solutions. However, it would have been less readable. If it were known in advance which queries are most likely to be asked (e.g. asking about atomic numbers), the appropriate clauses could have been put nearer the top of the knowledge base. However, in such a small knowledge base, the difference in efficiency would be negligible. If the knowledge base were extended to cover all the elements of the periodic table, questions of efficiency would become more important.

- **Could your solution be improved in any ways?** The knowledge base could be improved by adding more information about each element, and by extending it to cover all elements. Rules could be added about which chemicals could react with each other.

- **Are there any moral, ethical or legal issues associated with your solution?** There are no particular issues which arise in this case. All the information is in the public domain. However it would be important to ensure that the data is all correct, as incorrect information might lead to chemical accidents with possible dangerous consequences. Any documentation should draw the user's attention to this.