Expert Systems

The main way that we have been structuring this course is by considering how we might build a robot or a softbot. We have successively added ever greater functionality (reactive agents, memory, search on iconic representations, planning with logical representations, knowledge representation and reasoning). But, as we observed earlier, a knowledge base and an inference engine are the main components of the third kind of intelligent agent: 'standalone' agents. We have now covered enough material on knowledge representation and reasoning to take a look at some 'standalone' agents.

We're going to look at some systems that might make use of quite narrow expertise to assist humans to carry out certain tasks. Specifically, we look at (what used to be called) *expert systems*.

1 What is an Expert System?

1.1 Expertise

If we say that someone is an expert in some field we mean that, within that narrow specialisation, they exhibit high competence. If we build a 'standalone' agent and we call it an expert system, then we are conveying the same ideas. The system does not exhibit much general intelligence at all; it works within a relatively narrow, relatively specialised field. But it performs at expert-level in that field.

These systems may be used for classification, diagnosis, fault-finding, data interpretation, design, configuration or prediction. They might be used to instruct, monitor, analyse, advise, critique or control.

Commercially-used expert systems have included the following: a system used in a bank to give advice on home loan applications; a system used by a computer manufacturer to check customer orders for completeness; a system used in hospitals to interpret measurements of pulmonary function (the workings of the lungs) to spot signs of lung disease; a system used by chemists to interpret mass spectrometer data to help discover the molecular structure of unknown organic compounds; and a system to help geologists to evaluate mineral sites for potential ore deposits.

Of course, in the context of the enormous number of expert systems that have been developed but which did not get used or which underperformed, these successes look more isolated.

1.2 Why build an expert system?

- *To disseminate rare and costly expertise.*
- *To formalise expert knowledge.*
- *To integrate diverse sources of knowledge.*

There are other possible reasons too. Some people also argue that expert systems might be less error-prone than humans, and they might be 'fairer', more straightforward, or more dispassionate than human experts.

Of course, expert systems have many disadvantages too, which should restrict their use to certain domains. They lack insight, compassion, an understanding of human motivations, the ability to guess, the ability to learn (usually), common-sense knowledge and the human experts' sense of a 'gut feeling'. All of this might rule out the deployment of an expert system from the point of view that it wouldn't work and/or might not be wholly ethical to apply it. In fact, it is widely recognised that expert systems are best thought of as adjuncts to experts, not replacements of experts. In this case, the user can provide the compassion and the common-sense.

1.3 Desirable features of an expert system

- If the system is interactive, a good user interface is essential, i.e. the dialogue it carries out with the user must be considered 'natural' by the user. This includes such things as:
  - The order of the questions the system asks should be natural.
  - There should be no stupid questions (ones whose answers the system could reasonably have worked out for itself).
  - The system must be able to *explain* why it is asking a question and justify any conclusions it reaches. This is essential to give the user confidence in the system but may also be useful when debugging or extending the system.
- In its reasoning, an expert system should be able to do the following:
  - It should be able to make inferences that are plausible but not necessarily sound. For example, an expert system might have to infer a medical condition from a set of symptoms; it is unlikely that the condition will be a logical consequence of the symptoms; rather it will be a plausible (likely) conclusion.
  - It should be capable of functioning with imperfect domain knowledge and imperfect case data. Often the domain knowledge and/or the case data will be incomplete, the knowledge and/or data may not be reliable (e.g. it may have errors), and the knowledge and/or data may be expressed imprecisely (e.g. if it has *large* teeth, then it is *dangerous*).
  - Due in part to the previous two points, the system must be able to contemplate multiple competing hypotheses simultaneously.
- They should be designed with maintainability in mind. It must be easily possible to incorporate new knowledge (either through further knowledge engineering or using machine learning).

2 Rule-Based Expert Systems

Many technologies can be used to build expert systems. However, the majority of the expert systems that have been built so far are rule-based systems: the knowledge base contains facts and rules. So overwhelming has the number of rule-based expert systems been that some people (incorrectly) believe that rule-based system $=$ expert system. This is not the case: rule-based systems are one way of building expert systems.

2.1 The Knowledge Base

The knowledge base contains facts and rules. Suppose we were building a rule-based expert system for medical diagnosis. The rules would encode associations between symptoms and medical conditions. The facts would encode knowledge about the current patient.

Much research and development of rule-based systems has been conducted in a way that is disconnected from the work that has gone on in using logic to encode knowledge. But from our point of view, we can see that, in a rule-based expert system, the knowledge base contains positive Horn clauses. In rule-based systems, the rules are often written in an **if** ... **then** ... format, but they could equally well be written in either of the formats that we are used to:

$$\textbf{if} \quad p_1 \text{ AND } \ldots \text{ AND } p_n \quad \textbf{then} \quad q$$

$$(p_1 \wedge \ldots \wedge p_n) \Rightarrow q$$

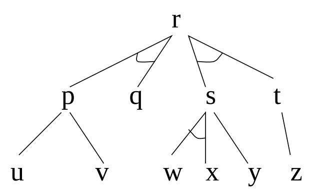
$$q \vee \neg p_1 \vee \ldots \vee \neg p_n$$

It's quite common for rule-based systems to use propositional logic where the predicates have no arguments. So there are no variables and function symbols in the facts and rules. (In some systems, the syntax used may make it look as though the predicates have arguments. But even in some of these systems, mathematically, the logic used is equivalent to propositional logic.) We'll confine ourselves to this situation in our examples below.

Rule chain together. The conclusion of one rule may be an atom that is in the antecedent of another rule. We can depict this chaining graphically as an AND-OR graph. For example, for the following rules:

if p AND q **then** r
if s AND t **then** r
if u **then** p
if v **then** p
if w AND x **then** s
if y **then** s
if z **then** t

we get the the following AND-OR graph:



Here's an example knowledge base for animal identification:

if *animalGivesMilk* **then** *animalIsMammal*
if *animalHasHair* **then** *animalIsMammal*
if *animalIsMammal* AND *animalChewsCud* **then** *animalIsUngulate*
if *animalIsUngulate* AND *animalHasLongNeck* **then** *animalIsGiraffe*
if *animalIsUngulate* AND *animalIsStriped* **then** *animalIsZebra*

We'll use this example below.

2.2 The Inference Engine

The inference engine will draw conclusions using the knowledge in the knowledge base. The work done by the inference engine of a rule-based system is referred to as *rule-based reasoning* (RBR). Of course, from a logic point of view, it is basically using resolution.

A perspective on rule-based reasoning is that it is doing a search of the AND-OR graph. Effectively, we are looking for a path that connects the root and the leaves which obeys the following:

- if a node is an OR node, it is sufficient to show there is a path for only one of its descendants;
- if a node is an AND node, it is necessary to show there is a path for each of the descendants;
- if a node is a leaf, the node represents a fact that must be true.

In interactive expert systems, we do not assume that all facts are known and stored already in the knowledge base. So, when we are trying to see whether a leaf node (fact) is true or not, we will check the knowledge base but, if it is not in the knowledge base, then we will ask the user a question to determine whether the fact is true or false. The user's answer can be added to the knowledge base. (This interactiveness is one thing that makes rule-based reasoning different from SLD-resolution.)

2.2.1 Backwards-chaining

Most rule-based reasoning is done in a way that is analogous to SLD-resolution. (As noted in the previous paragraph, there are reasons why it is rarely identical to SLD-resolution.) However, the phrases *backwards-chaining*, *goal-driven reasoning* and *hypothesis-driven reasoning* are more commonly used in the expert systems community.

In terms of the AND-OR graph, this kind of reasoning starts at the root of the graph and tries to find a path from the root to the leaves.

In the lecture, we'll trace an example of backwards-chaining using the animal identification rule base from earlier. In particular, we'll try to establish whether some animal is a giraffe. Notice from the example why phrases such as backwards-chaining, goal-driven reasoning and hypothesis-driven reasoning are used.

This kind of reasoning is most usually invoked when a user poses a query, i.e. when a user invokes the system's `ask` method. The user's query is the point from which the backward-chaining begins.

2.2.2 Forwards-chaining

An alternative to backwards-chaining is possible. This alternative goes by the name of *forwards-chaing*, although it is also called *data-driven reasoning*.

In terms of the AND-OR graph, this kind of reasoning starts at the leaves and tries to find a path from the leaves to the root.

This kind of reasoning is most usually invoked when a user supplies a fact, i.e. when a user invokes the system's `tell` method. The user's fact is inserted into the knowledge base and is then the point from which the fowards-chaining begins.

In the lecture, we'll trace an example of forwards-chaining again using the animal identification rule base. In particular, we'll assume that the user tells the system three facts: that the animal chews the cud, has hair and has a long neck. Notice from the example why phrases such as forwards-chaining and data-driven reasoning are used.

2.2.3 Interleaved Backwards- and Forwards-Chaining

Some systems work exclusively using one or other of backwards-chaining or forwards-chaining. But many interleave the two. This can be very natural. Consider a consultation with a human doctor. The patient describes some symptoms. Conclusions are drawn (maybe only tentatively) using forwards-chaining. The doctor selects a hypothesis and, through backwards-chaining, arrives at a question that is addressed to the patient. The patient speaks again, perhaps answering the question and/or volunteering other information. And so the process repeats.

2.3 Explanations of Reasoning

I have already indicated the importance that is attached to having explanation facilities in an expert system. Rule-based systems usually offer users two opportunities to request explanations. They may ask either `why` or `how` questions.

The system answers the questions by displaying some relevant rules. So note that explanations in rule-based systems amount to little more than traces of the reasoning.

Suppose an expert system is trying to establish whether the fact represented by a leaf node holds. It does this by asking the user a question. For concreteness, suppose the animal identification expert system asks the user whether the animal chews the cud. At this point, the user can, instead of answering the question, ask for a `why` explanation: why are you asking me this question? To this, the expert system replies by displaying an *ascent* of the AND-OR graph. The idea is that the reason the system is asking the question is given by showing which rules could fire on the basis of its answer. For example, the system might display:

```
I'm asking you whether animal chews cud because
this will help establish whether animal is ungulate

It has already been established that
 animal is mammal
Therefore if animal chews cud then animal is ungulate.

In turn this will help establish whether animal is giraffe.
If animal is ungulate and animal has long neck then animal is giraffe.
```

Suppose on the other hand, an expert system has established some node to be true. On telling the user its conclusion, the user can request a `how` explanation: how did you come to that conclusion? To this, the expert system replies by displaying a *descent* of the successful parts of the AND-OR graph. The idea is that to justify a conclusion the system should show which rules did fire in reaching that conclusion. For example, suppose at some point that the system establishes that the animal is an ungulate. The user might request a `how` explanation for this assertion, and this might look like the following:

```
The following rule was used to establish animal is ungulate
   if animal is mammal and animal chews cud then animal is ungulate
The following rule was used to establish animal is mammal
   if animal has hair then animal is mammal
You told me animal has hair.
You told me animal chews cud.
```

3 Concluding Remarks

Most of the ground-breaking research into rule-based expert systems was done on a system called MYCIN. The ideas in MYCIN have influenced all rule-based expert systems, even though MYCIN itself never saw regular, real-world use. MYCIN was designed to be used by a physician with the task of diagnosing bacterial infections of the blood. It has about 450 rules and and it worked (almost) exclusively using backwards-chaining.

The efficiency of MYCIN and the naturalness of its dialogue were improved by the incorporation of a whole range of extra little piece of knowledge that might be exploited during the backwards-chaining. One of the most significant of these was the use of a set of meta-rules, that were designed to help decide which normal rules should be used next.

Other famous ground-breaking rule-based expert systems are PROSPECTOR (for evaluating mineral sites for potential ore deposits) and R1 (a.k.a. XCON) for checking, completing and configuring customer computer equipment orders.

After some experience with building several rule-based expert systems, AI researchers realised that only the knowledge-base was domain-dependent. The inference engine and user interface were (relatively) domain-independent. These two could be packaged together and sold as an *expert system shell*. In principle, to create a new expert system for a different domain would require a process of knowledge engineering to obtain the rules.

Nowadays, it's more common to find the facilities that these shells offer as part of a larger programming language or programming environment. Examples are CLIPS, KEE and KAPPA. These tools might offer rule-based reasoning alongside object-orientation, connectivity to database and easy GUI creation.