Contents

AI lecture 3

Jocelyn Ireson-Paine

February 14, 1996

1 Introduction — rule-based systems

This lecture and the next will be about rule-based systems. This lecture: rule-based systems as advice-giving tools (AI as engineering). Next lecture: rule-based systems as cognitive models (AI as psychology).

2 Why are rule-based systems important?

- As engineering, they give one way of implementing *expert systems*. These were the first big commercial application to result from AI. Closely related to other developments, including logic programming (the use of Prolog and its ilk as programming languages) and CYC. Made AI visible to outsiders; inspired 5th Generation project and reactions thereto; much hype.
- As psychology, used under the name of *production systems* as cognitive models. Long history, largely inspired by Allen Newell, culminating in his view of the SOAR production system as a unified theory of cognition.

3 History

You won't understand AI unless you know something about its history — including the part played by rule-based systems. Why, for example, did symbolic models become popular in the late 50s?

For a brief run-through, see Chapter 8 of *The Guide to Expert Systems*, by Alex Goodall (Learned Information, 1985; RSL Comp BD 36; PSY KH:G 061). This was written in 1985, near the height of the expert systems hype. Indeed, the book was a result of this. A new expert systems journal had just started up, intended for a commercial audience. The book was to be given away (or sold cheaply) to new subscribers with no knowledge of AI, as a way of easing them into the subject.

For a more recent perspective, see *AI: the tumultuous history of the search for artificial intelligence* by Daniel Crevier (Basic Books 1993; PSY KH:C 068). I've just got this into the library, and will be recommending it as an introduction to AI in future years. Chapters 6 and 8 deal particularly with expert systems and their part in the commercialisation of AI. In a few years, it will be time to do the same for neural nets.

Some points in early AI history:

- Early focus on cybernetic models and simple neural nets. Belief in the ability of self-organising but unorganised systems to learn. Goodall pp 163–164;

Crevier pp 26–37, 102–107. [1]

- Naming the subject! Start of symbolic AI. Closely related to the development of digital computers. Goodall p 165 (see first quote); Crevier 37–50. See also GPS, Goodall p 166.
- Start of production-system models of mind. Crevier p 150. See where Newell and Simon got the idea from?
- Focus on domain-specific knowledge. Goodall pp 167–170. Crevier pp 146–156. See Feigenbaum quote on p 167 of Goodall and p 156 of Crevier. However, Crevier pp 146–147 also cites the limitations of early computers as a reason for lack of work on knowledge.
- Expert systems hype: See quote on Goodall p 173.
- Logic programming and Prolog. Crevier pp 190–196.
- Japanese Fifth Generation Project, and Western reactions. Goodall pp 175–176. Crevier chapter 8.

You can fill in the rest yourself. If you're not interested, you shouldn't be doing the course anyway! Be aware that different people have different interpretations of history, and that I've only mentioned a very small part of even the early work above.

4 What is a rule-based system?

- Consists of a *rule-base* (permanent data); an *inference engine* (process); and a workspace or working memory (temporary data). Not part of the basic reasoning process, but essential to applications, is the *user interface*.

  In the cognitive models, the rule-base is usually equated with LTM and the workspace with STM. There will be restrictions on these to correspond with assumptions about mental architecture: e.g. limited size of STM.
- Knowledge is stored as rules in the rule-base. (Also known as the *knowledge base*. Rules are of the form

  ```
  IF some condition THEN some action
  ```

  The condition tests working memory, e.g. for the presence of certain symbols or patterns of symbols. In many systems, the conditions are expressed logically as conjunctions (occasionally, disjunctions) of predicates. In some systems, some conditions correspond to sensor data. In a cognitive model, this would correspond to direct access to sensory input, rather than to its representation in STM.

  The action can be one of the following:

  - Another symbol or set of symbols to be added to STM. In many systems these will be expressed logically as conjunctions of predicates.
  - Some other action on STM, e.g. "delete the symbols XyZ".

[1] For a brief history of early computer science, connectionism and cybernetics, including their influence on psychology, see Chapter 1 of *Brains, Machines and Mathematics* by Michael Arbib (2nd edition, Springer-Verlag 1987; RSL Comp Bd 98; chapter 1 copied as AI box photocopy A52).

  - Some other action, e.g. turning a motor off, printing. In a cognitive model, this would correspond to a motor command.

- The inference engine applies the rules to working memory. There are various ways of doing this: see later. Why so called? See Crevier p 157.

- The user interface sits between the inference engine and the user. It translates the system's answers from an internal representation to something the user can understand; it passes questions from the system to the user and checks the replies (rejecting, for example, a negative number as the answer to a request for your weight).

- The components are independent, in that (as long as they are compatible), you can apply the same inference engine to many different knowledge bases. Crevier pp 152, 156.

5 Expert systems

Strictly, expert systems need *not* be rule-based. Could store knowledge as (e.g.) frames instead.

The term has no exact definition, but is, like "database" or "word-processor", a vague one, often misused by salesmen. It would have been generally agreed in the 80s, however, that certain properties distinguish an expert system from other software:

- An expert system is a computer program that reaches the performance of an expert in a narrow domain of intellectual expertise. Often this is some kind of diagnostic or planning task.

- Its knowledge is stored symbolically. Or that's what the enthusiasts said. In a sense this is misleading: isn't most stored knowledge symbolic in some sense?

  But here, the word "symbolic" comes from the same place as in "symbolic AI". It refers back to the symbolic AI tradition of knowledge representation, and implies that the knowledge which causes the system to make its decisions (the system's expertise) is coded in a form which can be made explicit to the human user. This allows him or her to appreciate how and why the decisions were made.

- It can justify its decisions, and explain why it is asking whatever questions it asks.

I have started to see "neural net" expert systems appearing. Presumably, these can't explain their conclusions, so maybe the sense of the word is changing slightly.

Other words for expert system: *intelligent knowledge-based system* (IKBS). I think this one was popularised by the British government in 5th generation project.

In the early days, "expert system" and "production system" were synonymous. Nowadays in AI, *production system* almost always means a rule-based cognitive model. These always restrict the style of inference and STM to reflect assumptions about the mind's cognitive architecture.

6 Examples of expert systems

See most AI books. See also the video. I'll try to get a copy of the latter into the library.

6.1 The "Classical" expert systems

The best-known and most typical was Mycin. See e.g. Winston *Artificial Intelligence* 3rd ed, pp 130–131. Also Goodall pp 16–20. Its job was to diagnose blood infections and recommend treatments, given lab data about tests on cultures taken from the patient, etc.

One of the earliest expert systems was Dendral, described on page 51 of Winston. It was designed to analyse mass spectra. For the purpose of this description, a mass spectrum is a particular trace or analytical record formed when a molecule is bombarded with electrons. Each molecule has its own spectrum, determined by the way it breaks into fragments when hit. Chemists have charts of mass spectra for some common molecules. And they know some general rules that determine how a given type of molecule will break up, and what kind of spectrum it will give. But identifying a new molecule from its spectrum is not easy.

Be warned. Dendral did contain rules, but it worked differently from most expert systems. Mycin is more typical.

And the two literature examples of commercially exciting systems you'll always find are Prospector and XCON/R1. Mycin has never been used for real diagnosis, perhaps partly because of fears over who'd be legally responsible for mistakes; Dendral was used by many chemists at Stanford, but probably never outside acadamia.

Prospector took geological information about rock formations, chemical content, etc, and advised on whether there were likely to be exploitable mineral deposits nearby. Popular accounts of AI say that Prospector (in 1978-ish) discovered a hundred-million-dollar deposit of molybdenum. Certainly, this "fact" helped raise business interest in expert systems. But the deposit was never exploited: I don't know the details, but it may be that Prospector mis-advised on the value or ease of extraction of this deposit.

XCON is described in Winston p 139. It is often mentioned because it was one of the most successful expert systems: it performed a task that couldn't be done manually or with a conventional computer program. It helped "configure" computer systems for DEC (who make VAXes amongst other things). See Crevier pp 161–162 for a summary, and for the origin of one of its names. ("I always wanted to be a (knowledge) engineer, and now I are one".)

Summary of configuration problem: When you buy a big computer, you need to order lots of parts: processor, power supplies, cables, discs, memory, etc. All this must be wired up sensibly, and arranged in the right kind of cabinet. For any set of parts, there are many ways to do this; complicated by the vast number of similar but not identical discs, memories etc that DEC sell. DEC were finding that their customers made mistakes in ordering; technicians failed to catch these errors, and made mistakes themselves in designing the configuration; so they decided to design a program to do it. XCON took about 3 years to get right, but it eventually handled, according to one writer, most (90passing those it can't cope with to a human configurer. DEC said it had saved them significant quantities of cash. However, see also Crevier p 204. The joke was that XCON may have replaced 75 people, but 150 were needed to keep it running.

6.2 Other expert systems

- DELTA, a system for diagnosing electric loco repair problems. Crevier pp 198, 204.
- Tax Advisor. Asked questions about its users' financial state and advised on how to minimise tax while maximising investment. Goodall p 21.

- A number of automated dealing systems. Hooked to electronically transmitted share prices and other data; contain rules telling when, and what, to buy or sell. (It's been suggested that the crash in 1987 was caused by there being many such systems, all running essentially the same program; this acted to amplify market fluctuations, and since lots of people were prevented by the hurricane from getting to work, the market was less damped than usual.)
- A program that planned where to site transistors on silicon chips so as to achieve the most compact and easy to fabricate layout. Written for Texas Instruments by Tony Priest and Bob Hoyle of Oxford Brookes University.
- Wheat Counsellor from ICI. Advised farmers on the diseases of their crops, and on treatments.

  Illustrates the danger of *"knowledge imperialism"*; the system is probably biassed to ICI products, and could be written to recommend more than really needed — how would a user be able to tell?

7 How the inference engine works

In simple rule-based systems, there are two kinds of inference, *forward chaining* and *backward chaining*.

Forward chaining: data gets put into working memory. This triggers rules whose conditions match the new data. These rules then perform their actions. The actions may add new data to memory, thus triggering more rules. And so on. This is also called *data-directed* inference, because inference is triggered by the arrival of new data in working memory.

Backward chaining: the system needs to know the value of a piece of data. It searches for rules whose conclusions mention this data. Before it can use the rules, it must test their conditions. This may entail discovering the value of more pieces of data, and so on. This is also called *goal-directed* inference, or *hypothesis driven*, because inferences are not performed until the system is made to prove a particular goal (i.e. a question).

7.1 Forward chaining

To see how forward chaining works, consider a system with three rules:

```
1 If someone is a third year, then they need a job.
2 If someone is a third year, then they live in.
3 If someone needs a job, they will become an accountant.
```

Suppose we put the following fact into STM:

```
John is a third year.
```

Being a forward chainer, the system is constantly on watch for new data. As soon as this data arrives, the system searches all the rules for any whose conditions weren't true before but are now. It then adds their conclusions to STM.

In this case, rules 1 and 2 have conditions which match this new fact. So the system will immediately create and add the two facts below.

```
John needs a job.
John lives in.
```

These facts in turn can trigger rules. As each arrives, the system would look for yet more rules that are made true. In this case, the fact `John needs a job` would trigger rule 3, resulting in the addition of another fact to STM:

```
John will become an accountant.
```

The fact `John lives in` would not trigger anything else though.

7.2 Backward chaining

This is a lazy kind of inference. It does no work until absolutely necessary, in distinction to forward chaining, where the system eagerly awaits new facts and tries applying conditions as soon as they arrive.

So, if we had this knowledge base again

```
1 If someone is a third year, then they need a job.
2 If someone is a third year, then they live in.
3 If someone needs a job, they will apply to be an accountant.
```

and we were to add

```
John is a third year
```

the system would do nothing at all.

But if we were then to ask the question

```
Is there anyone who is going to become an accountant?
```

the system would try to answer. It would begin by searching either for a fact that gives the answer directly, or for a rule by which the answer could be inferred. To find such a rule, it searches the entire knowledge base for rules whose conclusions, if made true, will answer the question.

In this example, there are no facts giving the answer; there's one rule whose conclusion, if true, would supply an answer, and that's rule 3.

The system next checks the rule's conditions. Is there anyone who needs a job? As with the original question, we look either for a fact that answers directly, or for a rule. There are no facts, but rule 1 is relevant.

So we now check *its* conditions. Is there a third year? This time, there is a fact that answers this: `John is a third year`. So we've proved rule 1, and that's proved rule 3, and that's answered the question.

7.3 Longer example of backward chaining

Start with this knowledge base. A simple set of rules to infer which resort a skier should go to, given his purpose (to have fun or to practise serious skiing), the number of lessons he's had so far, and the number of press-ups he can do without collapsing. Scenario: this knowledge base is built into an expert system. A client comes along for a consultation.

```
1 Resort = st sartre if
      Rating = beginner and
      Purpose = fun.

2 Resort = schloss heidegger if
      Rating = beginner and
      Purpose = serious.

3 Resort = chateau derrida if
      Rating = advanced and
      Purpose = serious.
```

```
4 Resort = wittgenstein gladbach if
      Rating = advanced and
      Purpose = fun.

5 Rating = beginner if
      Lessons < 30.

6 Rating = beginner if
      Lessons >= 30 and
      Fitness = poor.

7 Rating = advanced if
      Lessons >= 30 and
      Fitness = good.

8 Fitness = poor if
      Pressups < 10.

9 Fitness = good if
      Pressups >= 10.
```

In an expert system, we have to distinguish between *observable* or *askable* data which the client can supply, and *computable* data which the system calculates using the rules. Let's suppose that `Pressups`, `Fitness`, `Purpose` are askable. A real expert system language would make you specify that with the rules. Suppose also that the goal is to find a value for `Resort`.

So: goal is to find `Resort`. Do we know it already? No. So start by locating all rules which can tell you about `Resort`, i.e. which have it in their conclusions. These are rules 1, 2, 3, and 4.

Now, we have to investigate one or more of these rules. There are many possible orders: serially from top to bottom; serially from bottom to top; in parallel; etc. For simplicity, assume serial top to bottom. So we start with rule 1.

Test its conditions next. Do we know a value for `Rating`? No. Is `Rating` askable? No. So we have to calculate it from another rule; we can't ask the client for it.

So we now set a *subgoal* of calculating `Rating`. Locate all rules which have it in their conclusions. These are 5, 6, 7. Start with rule 5.

Test its conditions next. Do we have a value for `Lessons`? No. Is `Lessons` askable? Yes. So we ask the client for it. He types in `178`. We store this in STM. Now, is `Lessons` less than 30? No. So the condition fails. Rule 5 can't tell us anything, so we go on to rule 6.

Test its conditions next. Do we have a value for `Lessons`? Yes, we do now. Is `Lessons` 30 or more? Yes. So the first part of the condition succeeds. Go on to `Fitness`. Do we have a value for `Fitness`? No. Is it askable? No. So we have to calculate it.

So we now set another subgoal of calculating `Fitness`. Locate all rules which have it in their conclusions. These are 8 and 9. Start with 8.

Test its conditions. Have we a value for `Pressups`? No. Is it askable? Yes. Ask the client. He types `15`. Store it in STM. The condition fails, so abandon rule 8 and go on to 9. That condition succeeds, so we conclude `Fitness = good`. Store that in STM.

Now we've finished with the latest subgoal, so we return up a level, to rule 6. Its condition fails. So we go on to rule 7. Its condition succeeds. So we have now achieved the subgoal of calculating `Rating`. We can back up another level, to rule 1. Its conditions fail. Go on to rule 2, which also fails. Go on to rule 3. The first part of its condition succeeds; what about the second? Do we know `Purpose`? No. Is it askable? Yes. Ask the client. He types `fun`. So the condition fails. Go on to rule 4. This one's conditions succeed, we now have a value for `Resort`, and everyone's happy, especially me because I know I can always get a job as an expert system if all else fails.

7.4 Longer example of forward chaining

Take the same knowledge base as before. Assume now that we start with an empty STM. We add the fact `Purpose = serious`. Does this trigger any rules? Well, it matches the conditions of 2 and 3. However, there's nothing to match the rest of those conditions, so neither of the rules can be triggered. Unlike the last example, forward chaining will not follow the rules backwards to try satisfying conditions; it just tests the data already there.

Now add another fact to STM, /verb/Lessons = 178/. This matches the condition of rules 6 and 7, but neither of them can be triggered since there's nothing else to match the `Lessons` part.

Now add a third fact, `Pressups = 5`. This triggers rule 8, which adds a fourth fact, `Fitness = poor`. This in turn triggers rule 6 (both parts of whose conditions now match STM). Rule 6 adds `Rating = beginner`. And that triggers rule 2, which can now add a value for `Resort`.

7.5 More complicated inference methods

Some expert systems use more complex methods, for example mixtures of forward and backward chaining. Some, like Mycin, have probability factors attached to rules. Yet others store their rules in frames, and trigger them when an object is recognised as matching that frame.

8 Logic programming

In many cases, rules can be interpreted as logical statements: if X is true then Y is true. We can then think of the rules as a programming language based on logic. The inference engine runs this program. That's the idea behind Prolog ("PROgramming in LOGic") and related languages. Often linked to the idea that you tell the computer *what* to do, rather than as in (e.g.) Basic or Pascal, *how* to do it.

8.1 **Inference in Prolog

Skip this section if you're not doing the Prolog block practical.

If you used Eden in the Prolog practical, you probably used the production system that came with it. This works by forward chaining. If you printed any traces from it, look back at them. Rule conditions can test STM ("stm predicates") and sensory data ("perceptual predicates"). Rule actions can add data to STM, delete it from STM (using `erase`), or perform other actions ("motor commands").

This production system was built (by me) as a Prolog program. Prolog itself works by backward chaining, not by forward chaining. If you tell it

```
needs( X, job ) :- is_a( X, third_year ).
```

and you then add the fact

```
is_a( john, third_year ).
```

and ask

```
?- needs( Who, job ).
```

it will reply with `Who=john`. (This will not mean a lot to you unless you've used raw Prolog.)

However, Prolog is *not* an expert system. It can't explain *how* it inferred an answer. And even if it could, its user interface leaves a lot to be desired! But it can be, and often is, used as a language in which expert systems are built.

Whether you've used Prolog or not, it's important to note that the Eden production system (a forward chainer) was written in a programming language whose inference strategy is different (backward chaining). This illustrates the point I made in lecture 1 about a hierarchy of functional architectures. It shows that when you investigate the behaviour of a system, you may get information about one level, but not about a level lower down.

9 How expert systems explain themselves

Typically, they can answer two kinds of question: "how" and "why". 'How" means "how did you conclude that fact?"; "why" means "why are you asking me this question?". For examples, see the Mycin dialogue or the plant-cure one in Goodall.

To answer a "how did you prove X" question, a backward chainer would keep note of all the rules that were proven true en route to proving X. It would translate each into pseudo-English, and display the results.

To answer a "why are you asking this" question, the system looks to see what rule it's currently trying to prove true. (Remember that questions ensue from trying to prove the conditions of rules). Suppose the rule is, schematically, "If Q1 and Q2 and Q3 then C". It translates that rule into pseudo-English, and says "I am asking you Q1 because I want to prove C, and rule N says that C if Q1 and Q2 and Q3".

10 Where do the rules come from?

In a cognitive model (production system), they will be derived from psychological hypotheses about the contents of LTM: rules acquired from earlier learning. See next week's examples.

In an expert system, they will not correspond nearly as closely. The objective is to get a job done, not to build an accurate cognitive model. However, they will often be derived as descriptions of the behaviour of one or more human experts: by taking protocols, conducting interviews, setting classication tasks, etc. This is known as *knowledge engineering* or *knowledge acquisition*. Properly done, it involves a lot of psychological expertise, to make sure that you describe what the expert does rather than what he thinks he does.

Note that many statements that don't look like rules can be formulated as such. For example: "all mammals have fur" means "If something is a mammal, then it has fur". "Lockjaw is fatal" and "A patient with lockjaw will die" both mean "If the disease is lockjaw, then it is fatal". A small part of the knowledge engineer's job is recognising the hidden structure in such statements.

10.1 Rules by machine learning

There are also machine-learning programs which can induce general rules from specific cases. I gave a reference to one, using ILP, in my first lecture.

11 What's wrong with expert systems?

In the early days, thought to be wonderful. Particularly because

- They could generate their own explanations.
- The knowledge is modular: it's easy to add rules, and each rule corresponds to a nice little chunk of knowledge that an expert knows.

But there are problems. This leads to the subject of *human-computer interaction* amongst other things.

11.1 Maintaining large knowledge bases

Is difficult and expensive. See Crevier p 204.

11.2 Strategy and questioning

In this context, strategy means the type of inference used. This can determine the order in which questions are asked. It may be important, not only to have a strategy that gets the right answer, but one which asks questions in the same kind of order that a human expert would. For example, some inference engines would require that all answers to be input in advance. This may irritate a user, especially if he knows some of these answers may never be used. And for some questions, it can lead to silly situations: "Your sex please?"; "Male"; "Are you pregnant?" …

But systems that don't ask until they need data can also cause problems. Some sets of rules may result in a system that constantly switches from one topic to another and appears annoyingly unfocussed.

It's also important that the user be given a chance to volunteer data. With a pure backward chaining system, no data can be input until it's requested. If answering one question reminds you of the answers to another few, you should be allowed to volunteer the answers on the spot, rather than having to wait.

11.3 Explanations

Look at the Mycin dialogues again. Are Mycin's answers to how and why questions adequate? All it does is to feed back the rules that it's currently investigating (for why questions) or has investigated (for how questions). As you should see from the backward chaining examples, this is easy; but it is not good enough.

Until about 1983, it was generally believed that this kind of explanation was perfectly adequate. A great advantage of expert systems was said to be their ability to automatically generate explanations suitable for novices, without any extra work by the knowledge engineer. But these little nuggets of detail are useless to someone who doesn't know the underlying structure of a subject. What's needed for novices is (?) an overview, concentrating on general principles. A system like Mycin can't provide this because the only knowledge it has are its rules; there's no information on why the rules are what they are, or when they are appropriate. For example, in many expert systems, the order in which rules are used affects the conclusion. But there's no explicit knowledge saying why the rules are in that order. Maybe the author ordered them in order of specificity, or execution time. In the terms of Goodall pp 100–101, the conflict-resolution strategy is not explicit.

There has been research on making strategies explicit and incorporating them in explanations. For example, Neomycin. See *Strategic explanations for a diagnostic consultation system* by Hasling, Clancey and Rennels. In *Developments in expert systems* edited by Coombs (PSY KH:C 078).

11.4 Shallow reasoning

Even a system like Neomycin lacks a lot of knowledge. Why is it true that drug X cures disease Y? In medicine, we know so little that this must just be taken for granted. But in some other domains, we can try reasoning from first principles, *deep reasoning*. In contrast, Mycin-style reasoning that uses heuristic rules is called shallow reasoning.

For example, a shallow (Mycin-like) system for diagnosing faults in your household wiring might contain rules like "if the power has gone THEN it's a blown fuse with certainty 80would be equipped with basic electrical physics such as Ohm's Law, plus a wiring diagram of your house, and some knowledge of the strategy to be followed when looking for faults. Obviously, such a system could give a more principled explanation of its advice. The subject is very difficult though; I don't know of any deep reasoning systems that are in commercial use. For an experimental example, see *Reasoning from first principles in electronic troubleshooting* by Davis. Also in *Developments in expert systems*.

11.5 Lack of common sense, and restriction to narrow domains

See introduction to *Building large knowledge-based systems* by Lenat and Guha (Addison-Wesley 1990; PSY KH:L 054).

11.6 No learning

Most expert systems can't learn from past solutions. This is now being addressed, partly by Schank's group, as *case-based learning*. Instead of generating new solutions from scratch, these adapt old solutions to new uses. E.g. Chef, mentioned in my second tutorial.

11.7 Systems aren't adaptable

Novices and experts don't need the same type of explanation. Much research has gone into "adaptable systems" which adapt their explanation to the user's skill. This incidentally can apply to programs other than expert systems, and I'll assume it does in the rest of this section. For example, operating systems or databases whose error messages adapt to the user's knowledge.

Building such systems is tremendously difficult. Firstly, you need to find out how capable the user is. That's not easy when the only channel is his keyboard responses. Compare with with all the information a human expert can acquire about his client.

Secondly, you need to know something about the user's intentions and goals. For example, suppose you have a computer filing system, and the user asks to delete a file that doesn't exist. Possible reasons for his action:

- He has made a genuine typing error: e.g. typed 1 for I.
- He has forgotten what the file was called.
- He knows what the file is called; to save typing he tried to abbreviate its name, but did so wrongly (e.g. assuming he could leave the .DOC bit off a file with that extension).
- He's got the file name itself right, but the directory wrong.
- He thought he'd created the file with an earlier command, but he used that command incorrectly and it failed to create the file. E.g. a botched edit.

- He thought he'd created the file with an earlier command, but he used that command incorrectly and it gave the file a name different from what he expected.

When reporting the error, should the system

- Say 'Error'?
- Give a ten-line essay on the exact syntax of filenames?
- Display a list of all the files he owns?
- Keep a log all his previous commands, and search for any that went wrong, then try to work out if any of these failed to create a file he intended to create?

Again, there's been a lot of work on inferring users' goals, plans, and intentions, but it hasn't resulted in much of any use.

12 Rule-based systems and some themes in AI

Some important ideas in AI: goals and subgoals; good representations make needed information explicit; use knowledge instead of search; data-directed vs goal-directed. We can apply these to rule-based systems.

12.1 Backward chaining systems have goals and subgoals

You can view a backward chaining system as having an initial goal: to answer the question set by its user. In answering this question, the system may spawn subgoals: to answer further questions posed by the conditions of rules. In fact, such systems do represent their goals explicitly: they have to, so that they can return up a level.

However, most systems do not reason *about* their goals; e.g. should I abandon attempting to prove this because it's taking too long? Amongst those systems that do are *blackboard systems*. Instead of the simple last-posed first-answered strategy seen above, blackboard systems have an "intelligent" "strategy layer" or "control layer" which attempts to reason about the best order in which to prove something, and the best rules to use.

See also Neomycin. This doesn't reason about its goals in the same way as blackboard systems. But there is a sense in which the meta-rules contain knowledge about goals.

12.2 Rule-based systems and good representations

As with Evans' program and Marrian vision, rule-based systems need to make different pieces of information explicit at different times.

When printing rules for their user, what's important is a textual representation that looks like natural language. But an inference engine can't work with this. The internal form of the rules must (for example) allow the inference engine quickly to access the conclusion and each part of the condition. So the system has two options. Either store every rule together with appropriate natural language text, or back-translate the rule to text on output.

Last week, I said that indexing is (in Schank's view) one of the main issues in AI. An example here: in the backward chaining examples, the system had to search the entire knowledge base for rules whose conclusions were pertinent to a given question. E.g. to find, given `Resort`, all rules with `Resort` in their conclusion.

Efficiency can be raised by making the connection explicit, and building some kind of index which leads you straight from each variable to all the rules which mention it.

12.3 Expert systems and productive laziness

In the first lecture, I quoted from *Computers and Thought* on AI as productive laziness. In a sense, the difference between shallow and deep reasoning is one of laziness. Given a house-wiring problem, you can either solve it from first principles (deep reasoning), or you can rely on ad-hoc rules of thumb gained by previous experience (shallow reasoning). The first is more certain to be right (assuming you don't get tired and make mistakes); the second may be a lot faster.

Most systems have the rules but not the principles; deep reasoning systems have the principles but can't condense them into rules. The ideal would be a combined system. When set a problem, it first tries the rules it's learnt. If they fail, it resorts to fundamentals. Having done so, it encodes the answer to its problem as new rules so they can be used next time round. And when the system is idle, it spends its time preparing itself by solving imaginary problems and encoding the solutions as rules.

There has been some research on these combined systems. But given the difficulty of deep systems by themselves, it's not surprising that there are few results.

12.4 Data-directed vs goal-directed

Part of the general topic of *control* in computational systems; the order in which subtasks are performed. I mentioned this in the context of inference; it can be applied in many other contexts. E.g. learning systems. Do they try to generalise whenever new data comes in, or do they wait until they have some goal in mind, and then start generalising from stored cases? Case-based learning is largely an example of the second; inductive logic programming, of the first.