Expert Systems

Expert systems are designed to provide "expert quality" performance on domain-specific problems. In this chapter we look at the structure of expert systems focusing on the classical rule-based system. We then look at the process of developing an expert system.

7.1 Definitions and Examples

But what exactly is an expert system? Rosenman defined it as:

> *an automated reasoning system that attempts to mimic the performance of the human expert.*

This, of course, poses the question of what exactly is an expert. Michie avoided the comparison with the human:

> *Expert System = Knowledge Base + Inference Engine*

They have been successful in many fields. In general there are two types of problems: (a) select one of several hypotheses (e.g. diagnosis and advice); and (b) make a solution that meets requirements (e.g. design and planning). MYCIN was used for medical diagnosis (for a specific group of diseases). PROSPECTOR evaluated geological sites for commercial development. DENDRAL helped to infer a molecule's structure given its chemical formula and other data. XCON was developed to customize a network system to meet the customer's needs.

7.2 Design of An Expert System

Like a human expert, an expert system is expected to

- *be specialist*: know facts and procedural rules
- *use heuristics*: interpolate from known facts
- *justify its conclusions*: to establish credibility and confidence. The user can ask: how do you know a particular fact? why do you ask a particular question?
- *be able to learn*: be able to absorb new knowledge and apply it
- *estimate the reliability of its answer.*

The structure of a typical expert system is shown on the next page.

There are many advantages to the separation and modularity. For example, the knowledge base and inference engine being separate allows one to inspect the knowledge of the system, not just its responses to particular questions.

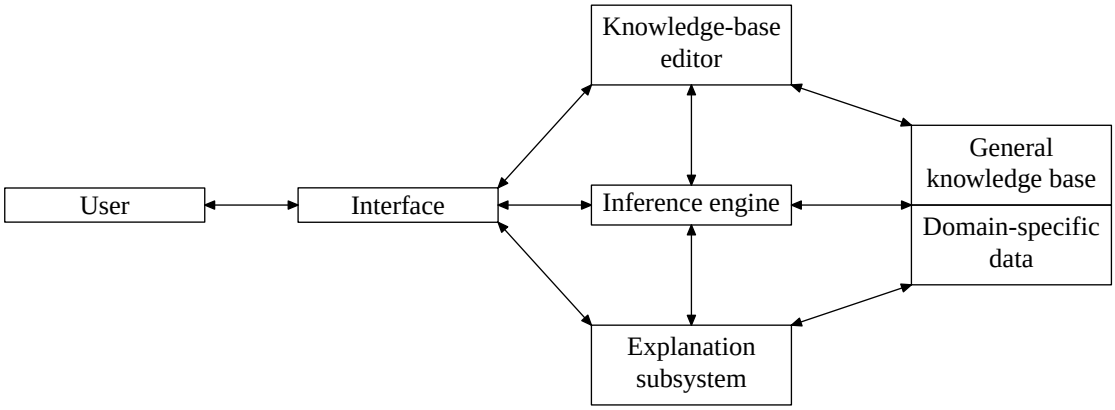

Figure 1: Typical Architecture (from Luger and Stubblefield)

If we return to Michie's definition of an expert system, we need to discuss the workings of the inference engine, and the structure and acquisition of the knowledge base. While early work on expert systems focussed on the inference mechanism used, it seems that the problem of constructing the knowledge base is much more taxing.

For, an expert system requires a large amount of knowledge. This knowledge is domain-specific. It should capture the typical knowledge of a human expert: facts, procedural rules, heuristic rules, as well as a general conceptual model and overall scheme. Surprisingly perhaps, common-sense reasoning systems may require even more knowledge than expert systems, but we do not explore this here.

7.3 Rule-Based Systems

A very common inference engine is based on representing knowledge in a rule base. Each rule is of the form:

if X *then* Y.

The set X is called the ***conditions*** and the set Y the ***consequents***. A rule is ***triggered*** if all the conditions are satisfied and then the consequents are ***fired***.

The working expert system has both a rule base and ***working memory***—the latter has accumulated facts or data. Rule base can be used as a ***deduction system***—each consequent is a new fact—or a ***reaction system***—each consequent is an action.

The following example (modeled on one of Winston) might be the rule base of a simple vehicle recognizer.

R1: If ?x has wings
then ?x is a plane

R2: If ?x flies
then ?x is a plane

R3: If ?x runs on tracks
then ?x is a train-or-tram

R4: If ?x is a plane
?x can take off vertically
?x has rotors
then ?x is a helicopter

R5: If ?x is a train-or-tram
?x stays underground
then ?x is a subway car

R6: If ?x is a helicopter
?x made in South Africa
then ?x is a Rooivalk

Rule-based expert systems have had considerable success and are versatile. And they are able to explain somewhat their reasoning. However, they are sometimes criticized as "idiot savants": they do not understand why the rules hold; they do not know how or when to break their own rules; they are very brittle (limited domain) and they do not know their limits.

7.4 Deductions in Rule Bases

With a rule base, knowledge can be developed by either ***data-driven*** or ***goal-driven*** search. In the former, also known as ***forward chaining***, one has a supply of facts and repeatedly applies legal moves or rules to produce new facts to get (hopefully) to the goal. In the latter, also known as ***backward chaining***, one repeatedly considers the possible final rules that produce the goal and from these creates successive subgoals. A rule might have several variables in it: the particular choice of substitution is known as the ***binding***.

FORWARD-CHAINING

- while (no new assertion made) and (unresolved)
  - for each rule
    (and for each possible binding)
    - try to support rule's conditions from known facts
    - if all supported then assert consequent

For example, one might know that the vehicle flies and so can fire Rule R2 to show that it is a plane. Then if one knows that it takes off vertically and has rotors, one can fire Rule R4 to show that it is a helicopter. Finally, if one knows that it made in South Africa, then one can fire Rule R6 to show that it is a Rooivalk.

Trying to support a rule's conditions from the known facts involves a considerable effort in ***pattern matching***. To achieve efficiency, one must have a method to re-use work from one round to the next. Forward chaining is used in production systems (discussed later) and in XCON for example. To provide an explanation facility, one stores all the successful firings.

BACKWARD-CHAINING

- while (no untried hypothesis) and (unresolved)
  - for each hypothesis
    - for each rule with hypothesis as consequent
      - try to support rule's conditions from known facts or via recursion (trying all possible bindings)
      - if all supported then assert consequent

For example, one may try to show that the vehicle is a Rooivalk by using Rule R6. The one fact—South African—is known, but it is not known whether the vehicle is a helicopter. That may be done by Rule R4. Two of the three facts there are known, but it is not known that the vehicle is a plane. So one can try Rules R1 and R2, etc.

Backward chaining is used in ***logic programming***, in particular inside Prolog. The recursion creates the danger of infinite regress and redundant computation: one idea to avoid this is memoization (remember partial results) or more generally dynamic programming (from algorithms).

In a reaction system (such as a game player) one has to worry about the order in which rules are triggered. This requires ***conflict resolution***. The most obvious mechanism is rule ordering, but there are many more mechanisms including: context limiting, general rules before (or after) specific rules, etc.

System	*Representation*	*Inference mechanism*
MYCIN	rules in LISP	backward chaining
PROSPECTOR	rules & semantic net	forward chaining & Bayesian reasoning
DENDRAL	rules	graph generation and test (using forward chaining)
XCON	rules	forward chaining; subtasking & constraint satisfaction

7.5 Reasoning with Uncertainty

In some rule-bases, each rule is only partial evidence. For example, a sample rule from PROSPECTOR was:

***If**: magnetite or pyrite in disseminated or veinlet form is present*
***Then**: $(2, -4)$ there is favorable mineralization and texture for the propylitic stage.*

The first confidence level indicates how much the first condition implies the hypothesis; the second indicates how much the converse holds. So the $+2$ says that the condition is mildly suggestive of the conclusion; the -4 says that the absence of the conclusion strongly indicates that the conclusion does not hold.

One mechanism to deal with such suggestive information, is to use conditional probabilities and bayesian reasoning. The ***probability*** of an ***event*** is the likelihood it happens, which can often be interpreted as the proportion of times the particular event occurs. For example, $\Pr(\textit{girl}) \approx 0.51$: the chance of being a girl is about 51%.

The ***conditional probability*** looks out how the odds change when some evidence is known (or equivalently, when we restrict attention to some subset). For example, $\Pr(\textit{girl}|\textit{Carol})$, the probability of being a girl given that your name is Carol, is probably much higher (like 99% in America). On the other hand, $\Pr(\textit{Carol}|\textit{girl})$ is asking what proportion of girls have the name Carol, which (thumb-suck) is around 1%. More generally, we want to know $\Pr(H|E)$: the probability of a hypothesis H given some evidence E.

MYCIN considered medical diagnosis. The trouble is that many conditional properties are not known: given that a person has puffy eyes, how likely is it they have measles? But doctors can readily tell one what proportion of people have measles, what proportion of people have puffy eyes, and what proportion of people with measles have puffy eyes. If we use d for disease and s for symptom, then we can use Bayes' theorem

$$\Pr(d|s) = \frac{\Pr(s|d)\Pr(d)}{\Pr(s)}$$

to infer the desired probability. The theory of ***Bayesian reasoning*** is to extend these ideas to multiple symptoms (even though the symptoms are not independent).

7.6 Knowledge

It is of course a hard and deep question to say what knowledge is. But certainly an expert has at her disposal at least (a) facts; (b) stored procedures; and (c) heuristics and rules-of-thumb.

We must also distinguish between ignorance and uncertainty. ***Ignorance*** is not knowing something that is knowable. ***Uncertainty*** is where something is not knowable: it is inherent in the situation. Ignorance can come from (a) the limited knowledge of human expert; (b) inexact data; or (c) incomplete data (which forces a premature decision).

To deal with uncertainty and ignorance, we need to keep track not only of our knowledge, but how sure we are and how we got there. Here are some techniques we might use:

- *Knowledge revision*: remember how a fact was deduced and be able to scrap it if supporting facts are overturned.
- *Default assumptions*: to handle incomplete data
- *Fuzzy logic*: Consider the question "Is a greyhound fast?" One might know the exact speed of a greyhound, but does that constitute fast? So we have a fuzzy fact: we get a "likelihood" of being true (a value between 0 and 1). In this system, AND becomes *min*, and OR becomes *max*. Fuzzy logic has had considerable success in systems for control and signal processing.

7.7 Building an Expert System

The expert system life cycle entails much more prototyping and revising than normal software development. The three broad steps in the process are as follows. Note that each stage might be iterated, and one might go back to the previous step.

1. *Problem selection & Prototype construction*: One must decide and narrow the problem, and assess the suitability of an expert system. Then one chooses knowledge representation schemes and the inference mechanism(s). After that one can implement the prototype, and obtain feedback from experts.
2. *Formalization & Implementation*: One draws up a detailed plan. At the same time, one revises the prototype and revisits the inference and representation decisions. After which one can construct the core knowledge base.
3. *Evaluation & Evolution*: Perhaps a modified Turing test for evaluation: see if the system can pass itself off as an expert in the domain.

One of the key tasks is ***knowledge acquisition***. This is the job of a ***knowledge engineer***, and is obtained mostly from domain experts. The process of knowledge acquisition is well-studied. One might proceed as follows:

1. *Assemble knowledge.* What is relevant? What are the facts and procedures? The knowledge engineer might start with interviews (unstructured, structured, etc.), asking questions about what is done. But often he has to watch the domain experts using their heuristics in specific situations. Often the key part is to determine why similar-looking situations are treated differently.

2. *Decide representation.* The knowledge engineer needs to select an ***ontology***: the vocabulary to capture the knowledge and the meaning of that vocabulary.

3. *Encode knowledge.* Translate the assembled knowledge into the chosen representation. This process might be enabled by some expert system ***shells***. Languages which have been popular include Prolog, LISP and OPS5.

4. *Verify.* Show that the knowledge is correct, consistent and complete.

An alternative is to use ***automatic programming*** or ***learning*** to do some of the knowledge assembly, encoding and verification.

7.8 When is an Expert System Appropriate?

Here are factors which suggest an expert system is appropriate.

- Need justifies cost and effort
- Human expertise not always available
- Problem requires symbolic reasoning
- Problem domain is well structured
- Traditional computing methods fail
- Cooperative and articulate experts exist
- Problem is not too large

Exercises

7.1. **Project.** Develop an expert system to play Dueller.