UNIT 5

CHAPTER 1

5.1 EXPERT SYSTEMS

An expert system is a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice.

To solve expert-level problems, expert systems will need efficient access to a substantial domain knowledge base, and a reasoning mechanism to apply the knowledge to the problems they are given. Usually they will also need to be able to explain, to the users who rely on them, how they have reached their decisions.

They will generally build upon the ideas of knowledge representation, production rules, search, and so on, that we have already covered.

Often we use an expert system shell which is an existing knowledge independent framework into which domain knowledge can be inserted to produce a working expert system. We can thus avoid having to program each new system from scratch.

5.2 Typical Tasks for Expert Systems

There are no fundamental limits on what problem domains an expert system can be built to deal with. Some typical existing expert system tasks include:

1. The interpretation of data

    Such as sonar data or geophysical measurements

2. Diagnosis of malfunctions

    Such as equipment faults or human diseases

3. Structural analysis or configuration of complex objects

    Such as chemical compounds or computer systems

4. Planning sequences of actions

    Such as might be performed by robots

5. Predicting the future

Such as weather, share prices, exchange rates

However, these days, "conventional" computer systems can also do some of these things

5.3 Characteristics of Expert Systems

Expert systems can be distinguished from conventional computer systems in that:

1. They simulate human reasoning about the problem domain, rather than simulating the domain itself.

2. They perform reasoning over representations of human knowledge, in addition to doing numerical calculations or data retrieval. They have corresponding distinct modules referred to as the inference engine and the knowledge base.

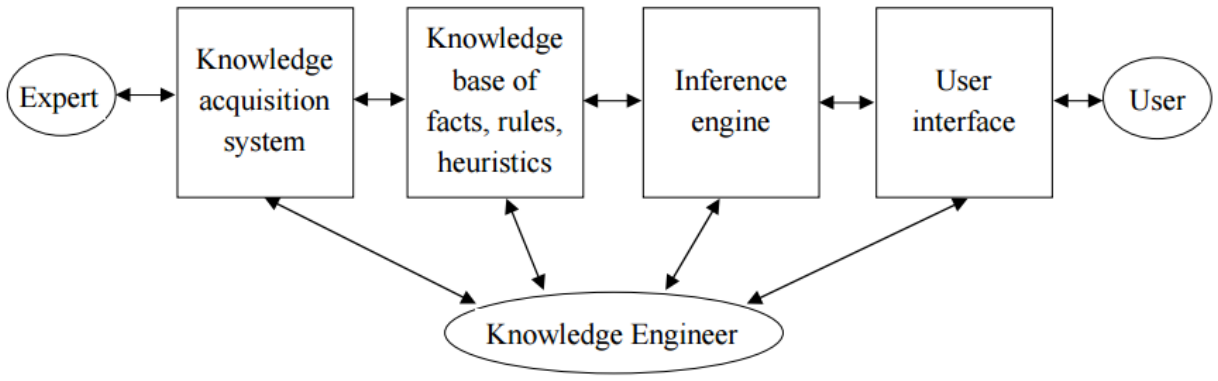
3. Problems tend to be solved using heuristics (rules of thumb) or approximate methods or probabilistic methods which, unlike algorithmic solutions, are not guaranteed to result in a correct or optimal solution.

4. They usually have to provide explanations and justifications of their solutions or recommendations in order to convince the user that their reasoning is correct.

Note that the term Intelligent Knowledge Based System (IKBS) is sometimes used as a synonym for Expert System.

5.3 The Architecture of Expert Systems

The process of building expert systems is often called knowledge engineering. The knowledge engineer is involved with all components of an expert system:


Expert
Knowledge acquisition system
Knowledge base of facts, rules, heuristics
Inference engine
User interface
User
Knowledge Engineer

Building expert systems is generally an iterative process. The components and their interaction will be refined over the course of numerous meetings of the knowledge engineer with the experts and users. We shall look in turn at the various components.

5.3.1 Knowledge Acquisition

The knowledge acquisition component allows the expert to enter their knowledge or expertise into the expert system, and to refine it later as and when required.

Historically, the knowledge engineer played a major role in this process, but automated systems that allow the expert to interact directly with the system are becoming increasingly common.

The knowledge acquisition process is usually comprised of three principal stages:

1. Knowledge elicitation is the interaction between the expert and the knowledge engineer/program to elicit the expert knowledge in some systematic way.

2. The knowledge thus obtained is usually stored in some form of human friendly intermediate representation.

3. The intermediate representation of the knowledge is then compiled into an executable form (e.g. production rules) that the inference engine can process.

In practice, much iteration through these three stages is usually required!

5.3.2 Knowledge Elicitation

The knowledge elicitation process itself usually consists of several stages:

1. Find as much as possible about the problem and domain from books, manuals, etc. In particular, become familiar with any specialist terminology and jargon.

2. Try to characterize the types of reasoning and problem solving tasks that the system will be required to perform.

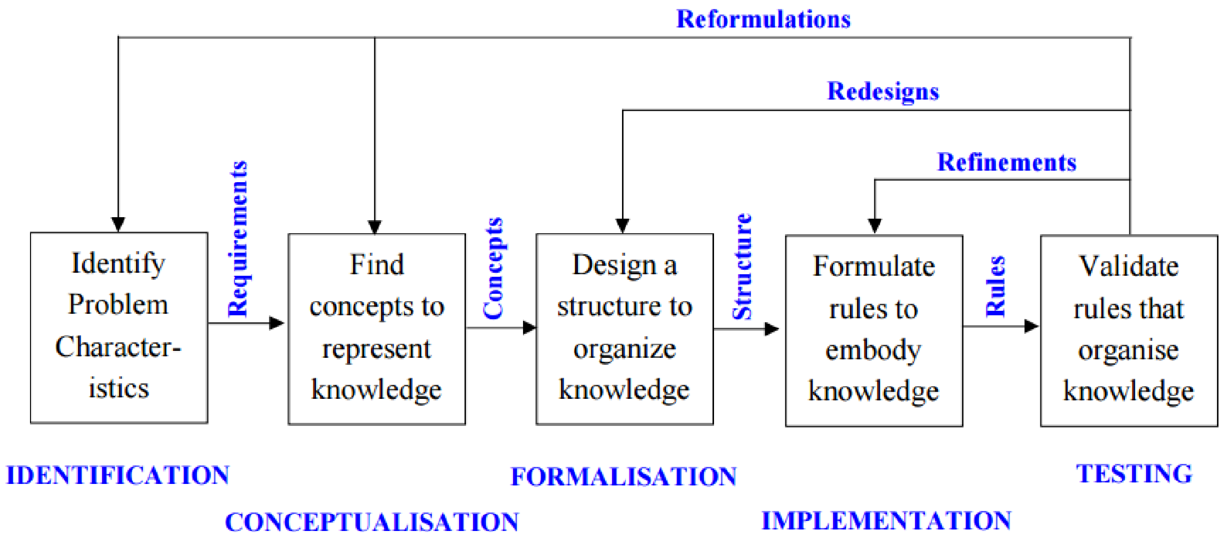
3. Find an expert (or set of experts) that is willing to collaborate on the project. Sometimes experts are frightened of being replaced by a computer system!

4. Interview the expert (usually many times during the course of building the system). Find out how they solve the problems your system will be expected to solve. Have them check and refine your intermediate knowledge representation.

This is a time intensive process, and automated knowledge elicitation and machine learning techniques are increasingly common modern alternatives.

5.3.3 Stages of Knowledge Acquisition

The iterative nature of the knowledge acquisition process can be represented in the following diagram.


Reformulations
Redesigns
Refinements
Identify Problem Character-istics
Requirements
Find concepts to represent knowledge
Concepts
Design a structure to organize knowledge
Structure
Formulate rules to embody knowledge
Rules
Validate rules that organise knowledge
IDENTIFICATION
CONCEPTUALISATION
FORMALISATION
IMPLEMENTATION
TESTING

5.3.4 Levels of Knowledge Analysis

Knowledge identification: Use in depth interviews in which the knowledge engineer encourages the expert to talk about how they do what they do. The knowledge engineer should understand the domain well enough to know which objects and facts need talking about.

Knowledge conceptualization: Find the primitive concepts and conceptual relations of the problem domain.

Epistemological analysis: Uncover the structural properties of the conceptual knowledge, such as taxonomic relations (classifications).

Logical analysis: Decide how to perform reasoning in the problem domain. This kind of knowledge can be particularly hard to acquire.

Implementation analysis: Work out systematic procedures for implementing and testing the system.

Capturing Tacit/Implicit Knowledge

One problem that knowledge engineers often encounter is that the human experts use tacit/implicit knowledge (e.g. procedural knowledge) that is difficult to capture.

There are several useful techniques for acquiring this knowledge:

1. **Protocol analysis**: Tape-record the expert thinking aloud while performing their role and later analyze this. Break down their protocol/account into the smallest atomic units of thought, and let these become operators.

2. **Participant observation**: The knowledge engineer acquires tacit knowledge through practical domain experience with the expert.

3. **Machine induction**: This is useful when the experts are able to supply examples of the results of their decision making, even if they are unable to articulate the underlying knowledge or reasoning process.

Which is/are best to use will generally depend on the problem domain and the expert.

5.3.5 Representing the Knowledge

We have already looked at various types of knowledge representation. In general, the knowledge acquired from our expert will be formulated in two ways:

1. **Intermediate representation** – a structured knowledge representation that the knowledge engineer and expert can both work with efficiently.

2. **Production system** – a formulation that the expert system's inference engine can process efficiently.

It is important to distinguish between:

1. **Domain knowledge** – the expert’s knowledge which might be expressed in the form of rules, general/default values, and so on.

2. **Case knowledge** – specific facts/knowledge about particular cases, including any derived knowledge about the particular cases.

The system will have the domain knowledge built in, and will have to integrate this with the different case knowledge that will become available each time the system is used.

CHAPTER -2

5.4 Meta Knowledge

Knowledge about knowledge

- Meta knowledge can be simply defined as knowledge about knowledge.
- Meta knowledge is knowledge about the use and control of domain knowledge in an expert system.

5.5 Roles in Expert System Development

Three fundamental roles in building expert systems are:

1. Expert - Successful ES systems depend on the experience and application of knowledge that the people can bring to it during its development. Large systems generally require multiple experts.

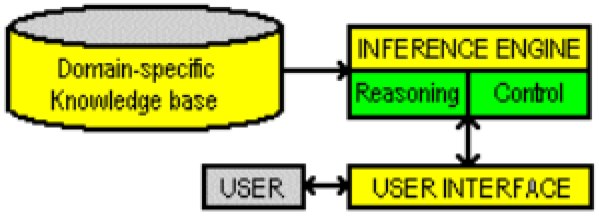
2. Knowledge engineer - The knowledge engineer has a dual task. This person should be able to elicit knowledge from the expert, gradually gaining an understanding of an area of expertise. Intelligence, tact, empathy, and proficiency in specific techniques of knowledge acquisition are all required of a knowledge engineer. Knowledge-acquisition techniques include conducting interviews with varying degrees of structure, protocol analysis, observation of experts at work, and analysis of cases.

On the other hand, the knowledge engineer must also select a tool appropriate for the project and use it to represent the knowledge with the application of the knowledge acquisition facility.

3. User - A system developed by an end user with a simple shell, is built rather quickly an inexpensively. Larger systems are built in an organized development effort. A prototype-oriented iterative development strategy is commonly used. ESs lends themselves particularly well to prototyping.

5.6 Typical Expert System

1. A problem-domain-specific knowledge base that stores the encoded knowledge to support one problem domain such as diagnosing why a car won't start. In a rule-based expert system, the knowledge base includes the if-then rules and additional specifications that control the course of the interview.



2. An inference engine a set of rules for making deductions from the data and that implements the reasoning mechanism and controls the interview process. The inference engine might be generalized so that the same software is able to process many different knowledge bases.

3. The user interface requests information from the user and outputs intermediate and final results. In some expert systems, input is acquired from additional sources such as data bases and sensors.

An expert system shell consists of a generalized inference engine and user interface designed to work with a knowledge base provided in a specified format. A shell often includes tools that help with the design, development and testing of the knowledge base. With the shell approach, expert systems representing many different problem domains may be developed and delivered with the same software environment. .

There are special high level languages used to program expert systems egg PROLOG

The user interacts with the system through a user interface which may use menus, natural language or any other style of interaction). Then an inference engine is used to reason with both the expert knowledge (extracted from our friendly expert) and data specific to the particular problem being solved. The expert knowledge will typically be in the form of a set of IF-THEN rules. The case specific data includes both data provided by the user and partial conclusions (along with certainty measures) based on this data. In a simple forward chaining rule-based system the case specific data will be the elements in working memory.

How an expert system works Car engine diagnosis

1. IF engine_getting_petrol

   AND engine_turns_over

   THEN problem_with_spark_plugs
2. IF NOT engine_turns_over

   AND NOT lights_come_on

   THEN problem_with_battery

3. IF NOT engine_turns_over

   AND lights_come_on

   THEN problem_with_starter
4. IF petrol_in_fuel_tank

   THEN engine_getting_petrol

There are three possible problems with the car:

- problem_with_spark_plugs,
- problem_with_battery,
- problem_with_starter.

The system will ask the user:

- Is it true that there's petrol in the fuel tank?

Let's say that the answer is yes. This answer would be recorded, so that the user doesn't get asked the same question again. Anyway, the system now has proved that the engine is getting petrol, so now wants to find out if the engine turns over. As the system doesn't yet know whether this is the case, and as there are no rules which conclude this, the user will be asked:

- Is it true that the engine turns over?

Lets say this time the answer is no. There are no other rules which can be used to prove ``problem_with_spark_plugs" so the system will conclude that this is not the solution to the problem, and will consider the next hypothesis: problem_with_battery. It is true that the engine does not turn over (the user has just said that), so all it has to prove is that the lights don't come one. It will ask the user

- Is it true that the lights come on?

Suppose the answer is no. It has now proved that the problem is with the battery. Some systems might stop there, but usually there might be more than one solution, (e.g., more than one fault with the car), or it will be uncertain which of various solutions is the right one. So usually all hypotheses are considered. It will try to prove ``problem_with_starter", but given the existing data (the lights come on) the proof will fail, so the system will conclude that the problem is with the battery. A complete interaction with our very simple system might be:

- System: Is it true that there's petrol in the fuel tank?

User: Yes.

System: Is it true that the engine turns over?

User: No.

System Is it true that the lights come on?

User: No.

System: I conclude that there is a problem with battery.

Note that in general, solving problems using backward chaining involves searching through all the possible ways of proving the hypothesis, systematically checking each of them.

Questions

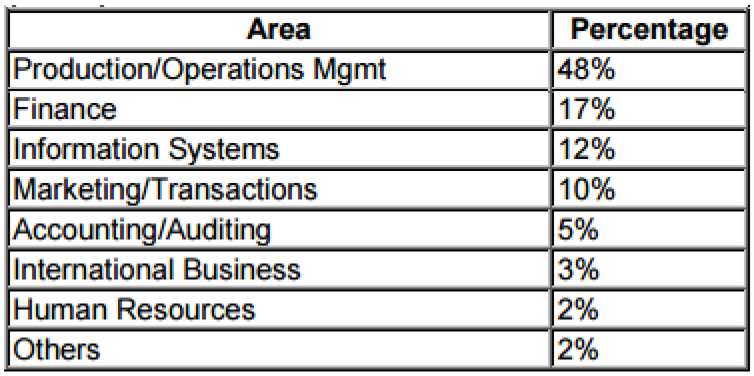
1. ``Briefly describe the basic architecture of a typical expert system, mentioning the function of each of the main components."

2. ``A travel agent asks you to design an expert system to help people choose where to go on holiday. Design a set of decisions to help you give advice on which holiday to take.

Expert System Use

Expert systems are used in a variety of areas, and are still the most popular developmental approach in the artificial intelligence world.

The table below depicts the percentage of expert systems being developed in particular areas:



Area	Percentage
Production/Operations Mgmt	48%
Finance	17%
Information Systems	12%
Marketing/Transactions	10%
Accounting/Auditing	5%
International Business	3%
Human Resources	2%
Others	2%

• Medical screening for cancer and brain tumours

• Matching people to jobs

• Training on oil rigs

• Diagnosing faults in car engines

• Legal advisory systems

• Mineral prospecting

5.6.1 MYCIN

Tasks and Domain

- ► Disease DIAGNOSIS and Therapy SELECTION
- ► Advice for non-expert physicians with time considerations and incomplete evidence on:
    - ▪ Bacterial infections of the blood
    - ▪ Expanded to meningitis and other ailments

System Goals

- ► Utility

- Be useful, to attract assistance of experts
- Demonstrate competence
- Fulfill domain need (i.e. penicillin)

► Flexibility

- Domain is complex, variety of knowledge types
- Medical knowledge rapidly evolves, must be easy to maintain K.B.

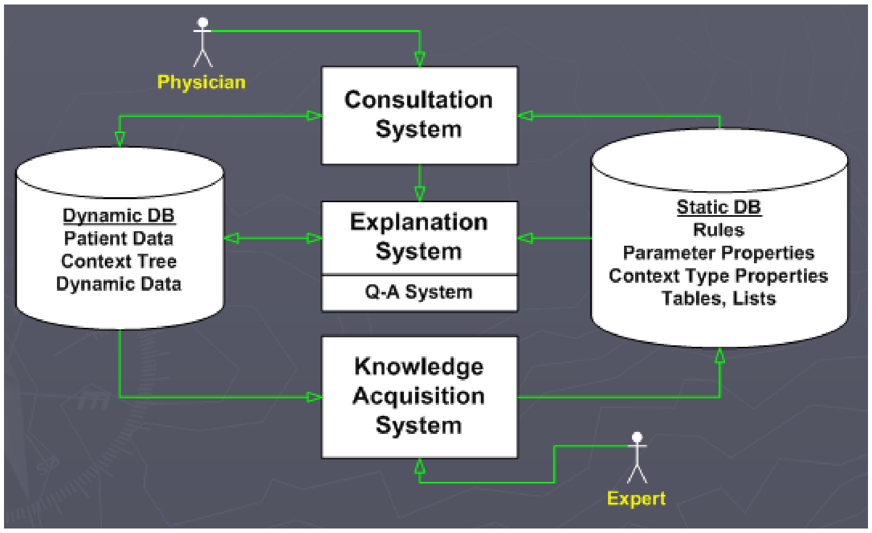
► Interactive Dialogue

- Provide coherent explanations (symbolic reasoning paradigm)
- Allow for real-time K.B. updates by experts

► Fast and Easy

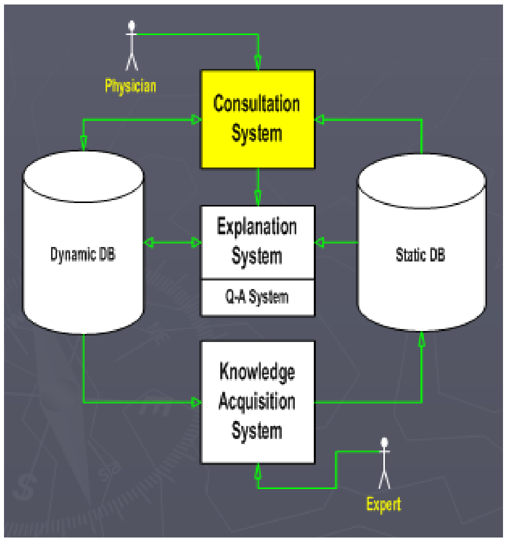
- Meet time constraints of the medical field

Architecture



Consultation System

- ▶ Performs Diagnosis and Therapy Selection
- ▶ Control Structure reads Static DB (rules) and read/writes to Dynamic DB (patient, context)
- ▶ Linked to Explanations
- ▶ Terminal interface to Physician

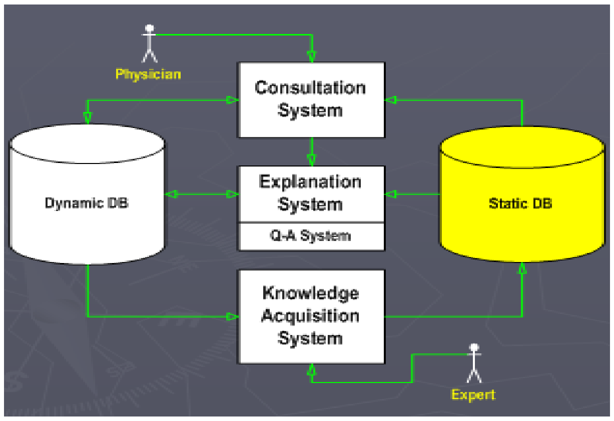


- ▶ User-Friendly Features:

    - Users can request rephrasing of questions
    - Synonym dictionary allows latitude of user responses
    - User typos are automatically fixed
- Questions are asked when more data is needed
    - If data cannot be provided, system ignores relevant rules
- Goal-directed Backward-chaining Depth-first Tree Search
- High-level Algorithm:
    - Determine if Patient has significant infection
    - Determine likely identity of significant organisms
    - Decide which drugs are potentially useful
    - Select best drug or coverage of drugs

Static Database

- Rules
- Meta-Rules
- Templates
- Rule Properties
- Context Properties
- Fed from Knowledge Acquisition System



Production Rules

- Represent Domain-specific Knowledge
- Over 450 rules in MYCIN
- Premise-Action (If-Then) Form:

  <predicate function><object><attrib><value>
- Each rule is completely modular, all relevant context is contained in the rule with explicitly stated premises

MYCIN P.R. Assumptions

- Not every domain can be represented, requires formalization (EMYCIN)
- Only small number of simultaneous factors (more than 6 was thought to be unwieldy)
- IF-THEN formalism is suitable for Expert Knowledge Acquisition and Explanation sub-systems

Judgmental Knowledge

- Inexact Reasoning with Certainty Factors (CF)
- CF are not Probability!
- Truth of a Hypothesis is measured by a sum of the CFs

    - Premises and Rules added together
    - Positive sum is confirming evidence
    - Negative sum is disconfirming evidence

Sub-goals

- At any given time MYCIN is establishing the value of some parameter by sub-goaling
- Unity Paths: a method to bypass sub-goals by following a path whose certainty is known (CF==1) to make a definite conclusion
- Won't search a sub-goal if it can be obtained from a user first (i.e. lab data)

Preview Mechanism

- Interpreter reads rules before invoking them
- Avoids unnecessary deductive work if the sub-goal has already been tested/determined
- Ensures self-referencing sub-goals do not enter recursive infinite loops

Meta-Rules

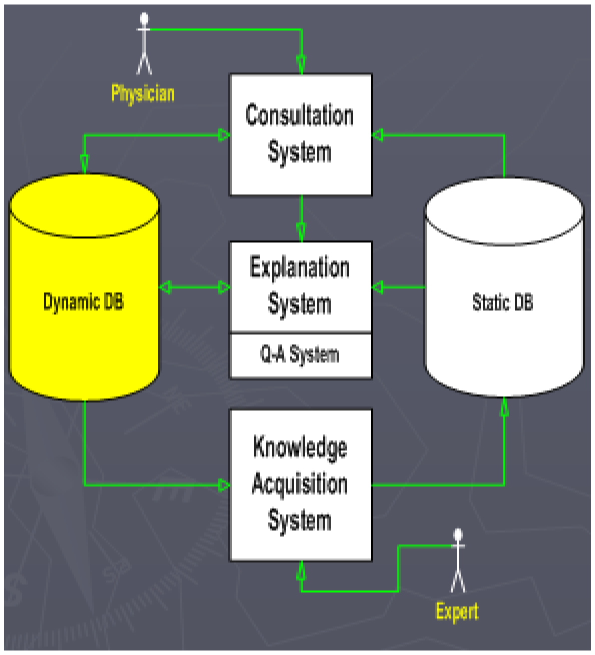
- Alternative to exhaustive invocation of all rules
- Strategy rules to suggest an approach for a given sub-goal
    - Ordering rules to try first, effectively pruning the search tree
- Creates a search-space with embedded information on which branch is best to take
- High-order Meta-Rules (i.e. Meta-Rules for Meta-Rules)
    - Powerful, but used limitedly in practice
- Impact to the Explanation System:
    - (+) Encode Knowledge formerly in the Control Structure
    - (-) Sometimes create "murky" explanations

Templates

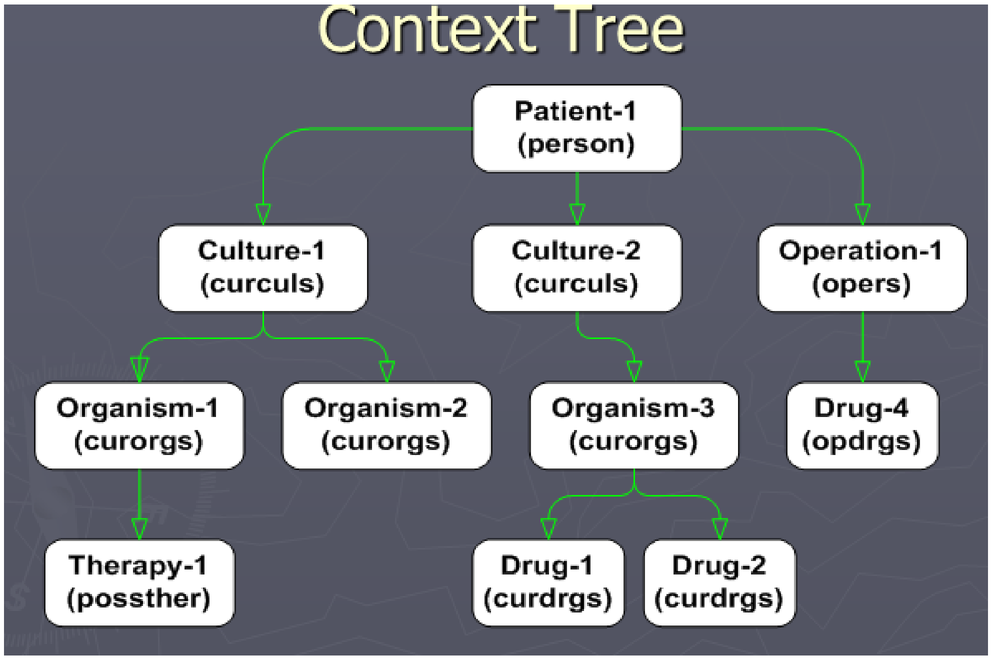
- ▶ The Production Rules are all based on Template structures
- ▶ This aids Knowledge-base expansion, because the system can "understand" its own representations
- ▶ Templates are updated by the system when a new rule is entered

Dynamic Database

- ▶ Patient Data
- ▶ Laboratory Data
- ▶ Context Tree
- ▶ Built by Consultation System
- ▶ Used by Explanation System



Context Tree



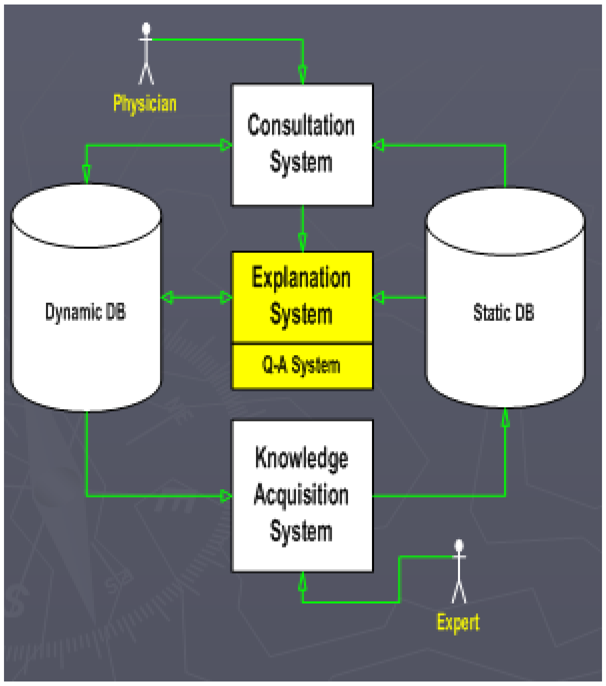
Therapy Selection

- Plan-Generate-and-Test Process
- Therapy List Creation
  - Set of specific rules recommend treatments based on the probability (not CF) of organism sensitivity
  - Probabilities based on laboratory data
  - One therapy rule for every organism
- Assigning Item Numbers
  - Only hypothesis with organisms deemed "significantly likely" (CF) are considered
  - Then the most likely (CF) identity of the organisms themselves are determined and assigned an Item Number
  - Each item is assigned a probability of likelihood and probability of sensitivity to drug
- Final Selection based on:

    - Sensitivity
    - Contraindication Screening
    - Using the minimal number of drugs and maximizing the coverage of organisms
- Experts can ask for alternate treatments
    - Therapy selection is repeated with previously recommended drugs removed from the list

Explanation System

- Provides reasoning why a conclusion has been made, or why a question is being asked
- Q-A Module
- Reasoning Status Checker



- Uses a trace of the Production Rules for a basis, and the Context Tree, to provide context
    - Ignores Definitional Rules (CF == 1)

- ► Two Modules
    - ▪ Q-A Module
    - ▪ Reasoning Status Checker

Q-A Module

- ► Symbolic Production Rules are readable
- ► Each <predicate function> has an associated translation pattern:

GRID (THE (2) ASSOCIATED WITH (1) IS KNOWN)

VAL (((2 1)))

PORTAL (THE PORTAL OF ENTRY OF *)

PATH-FLORA (LIST OF LIKELY PATHOGENS)

i.e. (GRID (VAL CNTXT PORTAL) PATH-FLORA) becomes:

"The list of likely pathogens associated with the portal of entry of the organism is known."

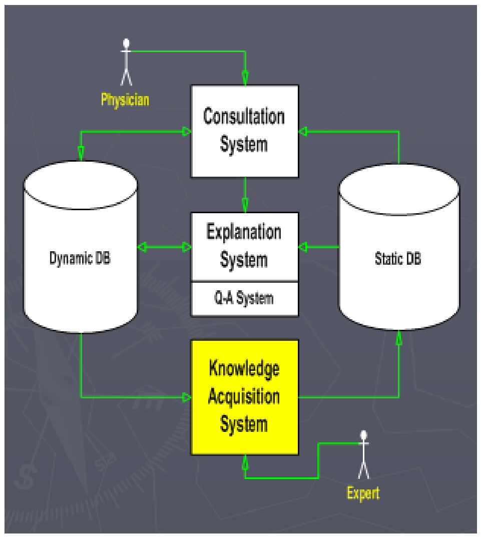
Reasoning Status Checker

- ► Explanation is a tree traversal of the traced rules:
    - ▪ WHY – moves up the tree
    - ▪ HOW – moves down (possibly to untried areas)
- ► Question is rephrased, and the rule being applied is explained with the translation patterns

Knowledge Acquisition System

- ► Extends Static DB via Dialogue with Experts
- ► Dialogue Driven by System
- ► Requires minimal training for Experts

- Allows for Incremental Competence, NOT an All-or-Nothing model



- IF-THEN Symbolic logic was found to be easy for experts to learn, and required little training by the MYCIN team
- When faced with a rule, the expert must either except it or be forced to update it using the education process

Education Process

1. Bug is uncovered, usually by Explanation process
2. Add/Modify rules using *subset of English* by experts
3. Integrating new knowledge into KB
    - Found to be difficult in practice, requires detection of contradictions, and complex concepts become difficult to express

Results

- Never implemented for routine clinical use
- Shown to be competent by panels of experts, even in cases where experts themselves disagreed on conclusions
- Key Contributions:

- Reuse of Production Rules (explanation, knowledge acquisition models)
- Meta-Level Knowledge Use

5.6.2 DART

The Dynamic Analysis and Replanning Tool, commonly abbreviated to DART, is an artificial intelligence program used by the U.S. military to optimize and schedule the transportation of supplies or personnel and solve other logistical problems.

DART uses intelligent agents to aid decision support systems located at the U.S. Transportation and European Commands. It integrates a set of intelligent data processing agents and database management systems to give planners the ability to rapidly evaluate plans for logistical feasibility. By automating evaluation of these processes DART decreases the cost and time required to implement decisions.

DART achieved logistical solutions that surprised many military planners. Introduced in 1991, DART had by 1995 offset the monetary equivalent of all funds DARPA had channeled into AI research for the previous 30 years combined

Development and introduction

DARPA funded the MITRE Corporation and Carnegie Mellon University to analyze the feasibility of several intelligent planning systems. In November 1989, a demonstration named The Proud Eagle Exercise indicated many inadequacies and bottlenecks within military support systems. In July, DART was previewed to the military by BBN Systems and Technologies and the ISX Corporation (now part of Lockheed Martin Advanced Technology Laboratories) in conjunction with the United States Air Force Rome Laboratory. It was proposed in November 1990, with the military immediately demanding that a prototype be developed for testing. Eight weeks later, a hasty but working prototype was introduced in 1991 to the USTRANSCOM at the beginning of Operation Desert Storm during the Gulf War.

Impact

Directly following its launch, DART solved several logistical nightmares, saving the military millions of dollars. Military planners were aware of the tremendous obstacles facing moving military assets from bases in Europe to prepared bases in Saudi Arabia, in preparation for Desert Storm. DART quickly proved its value by improving upon existing plans of the U.S. military. What surprised many observers were DART's ability to adapt plans rapidly in a crisis environment.

DART's success led to the development of other military planning agents such as:

- RDA - Resource Description and Access system
- DRPI - Knowledge-Based Planning and Scheduling Initiative, a successor of DART

5.7 Expert Systems shells

Initially each expert system is build from scratch (LISP). Systems are constructed as a set of declarative representations (mostly rules) combined with an interpreter for those representations. It helps to separate the interpreter from domain-specific knowledge and to create a system that could be used construct new expert system by adding new knowledge corresponding to the new problem domain. The resulting interpreters are called shells. Example of shells is EMYCIN (for Empty MYCIN derived from MYCIN).

Shells − A shell is nothing but an expert system without knowledge base. A shell provides the developers with knowledge acquisition, inference engine, user interface, and explanation facility. For example, few shells are given below −

- Java Expert System Shell (JESS) that provides fully developed Java API for creating an expert system.
- Vidwan, a shell developed at the National Centre for Software Technology, Mumbai in 1993. It enables knowledge encoding in the form of IF-THEN rules.

Shells provide greater flexibility in representing knowledge and in reasoning than MYCIN. They support rules, frames, truth maintenance systems and a variety of other reasoning mechanisms.

Early expert system shells provide mechanisms for knowledge representation, reasoning and explanation. Later these tools provide knowledge acquisition. Still expert system shells need to integrate with other programs easily. Expert systems cannot operate in a vacuum. The shells must provide an easy-to-use interface between an expert system written with the shell and programming environment.

PART-A

1. What are the phases in Expert system development?

2. Explain identification phase.

3. Explain conceptualization phase.

4. Explain formalization phase.

5. Explain implementation and testing phases.

6. Describe demonstration prototype.

7. Describe research prototype.

8. Describe field prototype.

9. What is meant by production prototype?

10. What is meant by commercial system?

11. When is Expert System Development Possible?

12. When is Expert System Development justified?

13. When is Expert System Development appropriate?

14. What are the questions to ask when selecting an Expert system tool?

15. What are the limitations of expert systems?

16. What are the problems the company faces when trying to apply expert system?

17. What are the common pitfalls in planning an expert system?

18. What are the pitfalls in dealing with the domain expert?

19. What are the pitfalls during the development process?

20. Where is expert system work being done?

21. What is meant by a commercial expert system?

22. Name any two expert system used in research?

23. Name any two expert system used in business?

24. Explain XCON?

25. Name any three universities and mention the expert system tools developed there?

26. Name any three research organization and mention the expert system tools developed there?

27. What are expert systems?

28. What are the most important aspects of expert system?

29. What are the characteristics of expert system?

30. Sketch the general features of expert system?

31. Who are all involved in the expert system building?

32. Explain the role of domain expert?

33. Explain the role of knowledge engineer?

34. What is the use of expert system building tool?

35. Give the structure of an expert system?

36. Explain the knowledge acquisition process?

37. Define MYCIN.

38. Define DART.

PART-B

1. Explain the tasks involved in building expert system?

2. Explain the various stages of expert system development?

3. Explain the difficulties involved in developing an expert system?

4. What are common pitfalls in planning an expert system?

5. Explain the pitfalls in dealing with the domain expert?

6. Explain the pitfalls during the development process?

7. Explain expert system work at universities and research organizations?

8. Explain expert system work at knowledge engineering companies?

9. What is meant by high performance expert system? How is it used in research and in business?

10. In detail Explain XCON?

11. Draw the schematic of expert system and explain.

12. Explain the following in detail.. a. MYCIN   b. EMYCIN