Expert Systems

Lectured: K. MacGregor 2 April - 25 April 1986

Appendices

1. WHAT IS AN EXPERT SYSTEM?

An ES is regarded as the embodiment within a computer of a knowledge-based component from an expert skill in such a form that the system can offer intelligent advice or take an intelligent decision about a processing function.

A desirable characteristic which many would consider fundamental is the capability of the system on demand to justify its own line of reasoning in a manner directly intelligible to the enquirer. The style adopted to attain these characteristics is RULE-BASED PROGRAMMING.

NOTE:

(i) "Expert" is taken seriously:
- high level of proficiency (correct results or most likely answer)
- solutions must be found quickly using:
  - heuristics (high-level inference patterns)
  - hunches (tricks of the trade)
- applied to complicated problems requiring expertise (not trivial)
- specific FIELD of expertise (as you move away from the field, answers are weaker)

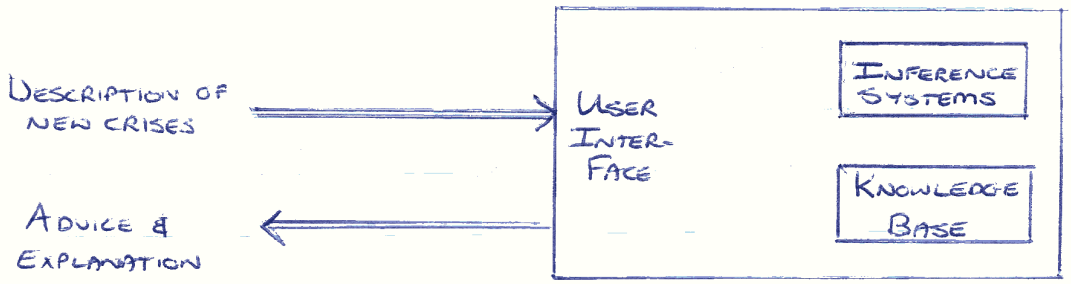
(ii) DATA REPRESENTATION: Data must be arranged "naturally" in some data- or knowledge-base. These could be networks: trees or frames which are nodes with special attributes

(iii) PROGRAMMING METHODOLOGY: Rules which are premise-action pairs. IF p & q then r (prob). The rules can be driven forward to

produce a result or backwards to justify a conclusion.

The rules can be linked together with 1 rule calling another rule, each triggered by the action or consequent becoming true.

RULE-BASED EXPERT SYSTEM



INFERENCE ENGINE is the software component of an expert system which draws inferences (based on the description) from the facts and rules of the knowledge-base. It is domain independent and corresponds to the generic intelligence of the ES (a generalised problem-solving device)

2. SAMUELS' CHECKERS-PLAYING PROGRAM

(Computers & Thought E FEIGENBAUM pp 50-75)

There are 10^{40} choices in draughts, so the problem is large enough to require expertise.

The method chosen was a simple α-β pruning algorithm tied to an evaluation function which returned a value of the position. The system could learn by being given book moves and modifying its own evaluation function to produce the optimum moves.
The evaluation function was a polynomial $Ax + By + Cz + \ldots + E$
where: x, y, z are factors representing board positions
A, B, C, E are numerical constants for weighting

PERFORMANCE: Reached level of US State champion

Components:

database = board positions
rules = evaluation function
explanation = evaluation values
user interface = board positions
learning: from examples (knowledge-base)

Is this an expert system? The rules are not written up explicitely, but nevertheless the evaluation function provides a rule for choice, so we could consider it an ES

The program was written in 1955*, and was working well in 1962, but it was later abandoned as it could not improve

* on an IBM 704, size 1100 words, tables 850 words
times: 2.6 ms to find all available moves 2.4 ms to evaluate a position
1.5 ms to make move & find resulting position

3 MYCIN (A system for diagnosing & recommending treatment on infectious diseases)

Started in 72-76, MYCIN performs well, even beating the medical schools in a 'test'. However, it is not used as it is slow and rests on a poor psychological basis.

FACTS: Stores of triples in the KB of the form:

CONTEXT – PARAMETER – VALUE

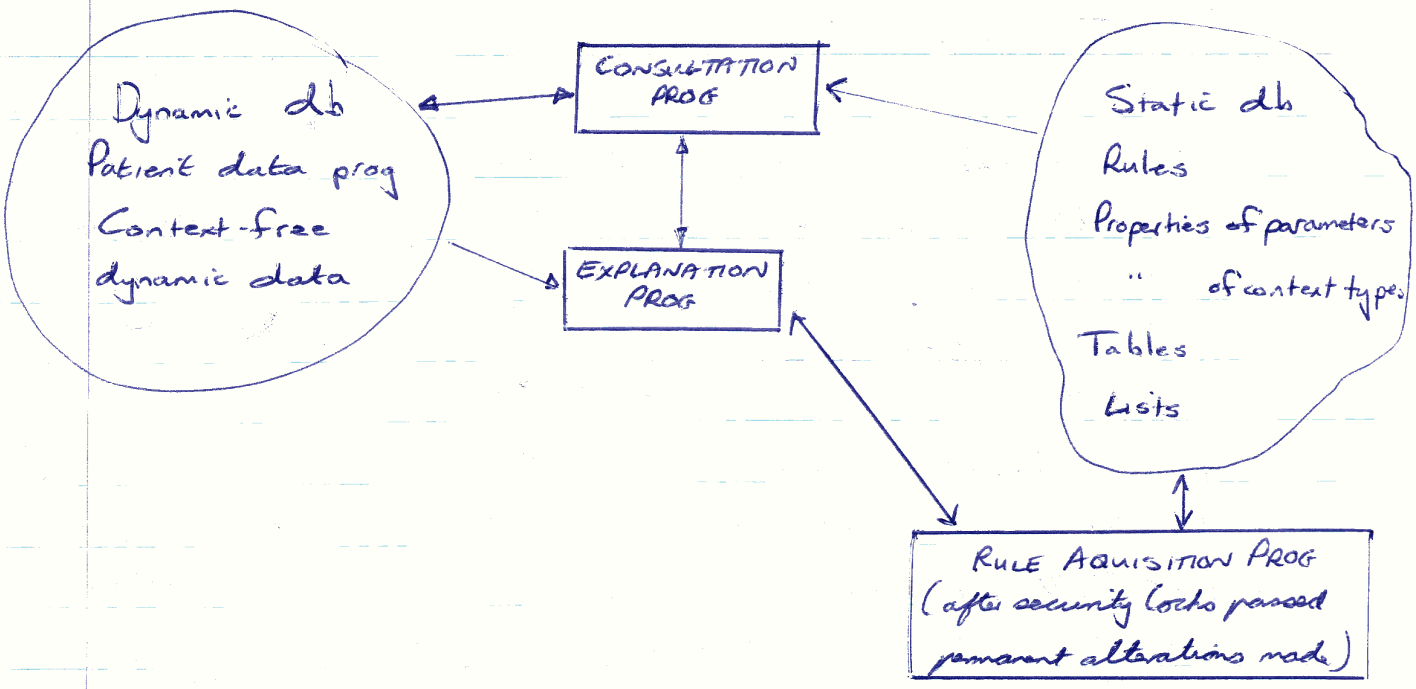
real world entity — attribute — certainty factor.

eg J. Smith - Age 25 - 1 ; Organism - identity pseudomonas - 1

RULES If premise then action (certainty factor def yes < 1, def not > 1)
Can be driven forward or backward.

KNOWLEDGE-BASE Divided into two sections: static data (rules - disease facts) and dynamic data (relating to individual patient information)

KNOWLEDGE AQUISITION: User front end gives explanations and acquires new knowledge (new rules). Uses meta-rules describing (eg) the importance of the data and the order in which the rules must be fired. The user can alter the meta-rules or add rules for specific conditions.



A 4th component, GUIDON, also exists. This is a teaching system containing about 200 tutorial rules. It builds a tutorial model, responding to student initiative. Uses meta-rules:

eg <u>if</u> 3 rules relating to the goal are not discussed
<u>and</u> student has asked questions
<u>and</u> student has applied one rule
<u>then</u> state the rule & conclusion (?)

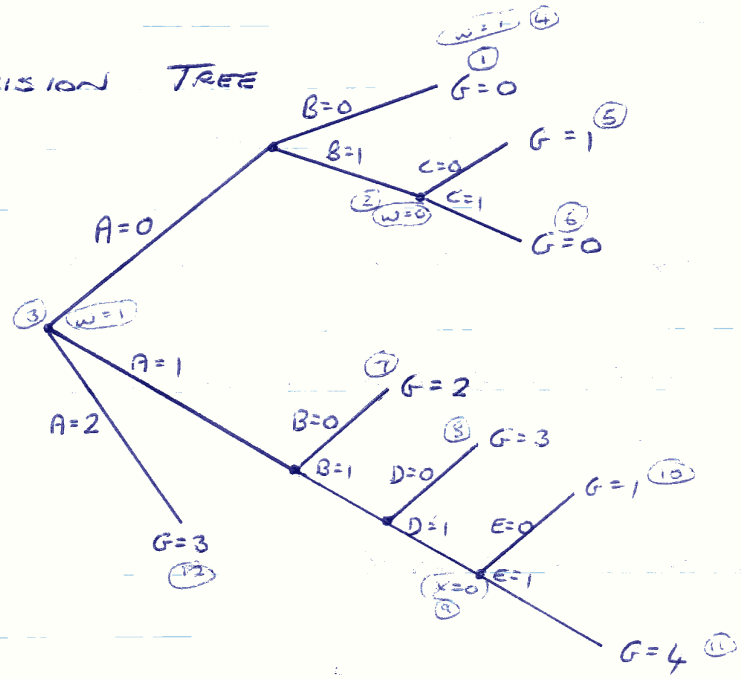
GUIDON is used as a seperate shell or system.

4. The Inference Engine

We assume the rules are coded like if_then_ rules, with multi-valued variables (of pascal sets) With rules of this form we can consider forward chaining or backward chaining.

Example: Assume we have a goal parameter G whose value depends on other values A, B, C, D and E. We generate internal values x and w.

Decision Tree



A, C, D, E single valued
B multi-valued.

RULES

1 if A=0 and B=0 then G=0
2 if A=0 and B=1 then W=0
3 if A<>0 then W=1
4 if A=0 and B<>1 then W=1 (to exclude from further checks)
5 if W=0 and C=0 then G=1
6 if W=0 and C=1 then G=0
7 if A=1 and B=0 then G=2
8 if A=1 and B=1 and D=0 then G=3
9 if A=1 and B=1 and D=1 then X=0
10 if X=0 and E=0 then G=1
11 if X=0 and E=1 then G=4
12 if A=2 then G=3

FORWARD CHAINING (DATA-DRIVEN CHAINING)

Start from the known information. The system asks for the values of some parameters which appear in the condition parts of the prospective rules. Inspection of rules will be deferred if the rules involve other parameters whose values can be inferred by other values. Forward-chaining corresponds to a tree traversal from the root to the leaves. The forward chaining of our example is:

INSPECT THE RULES ONE-BY ONE

At rule :	Action
1	Don't know A. Enquire (say A=1) REJECT
2	Reject, A<>0

3 A<>0, so fire. W ← 1

4 reject as A<>0

5 reject as W<>0

6 reject as W<>0

7 don't know B. Enquire (say B ← 0 + 1) fire G ← 2

8 don't know D. Enquire (say D ← 0)

fire G ← 3

9 reject as D <> 1

10 suspend execution (X unknown)

11 " " "

12 reject as A<> 2

10 reject as X has default value 1

11 " " " " " " " "

Give result: G = 2 (rule 7)

G = 3 (rule 8)

BACKWARD CHAINING

Backward chaining is goal-driven. All rules which draw conclusions on goals are marked. The clauses in the IF part of the first inspected rule are created as subgoals and so on, recursively, until no rule is found for inferring a value for a parameter. The system then asks for the value of the parameter. This corresponds to traversing the decision tree from the leaves to the root.

In our example rules 1, 5-8 and 10-12 are marked for inspection. Suppose X is initialised to 1. The result is G = 2 (rule 7) G = 3 (rule 8)

9.

• at rule 1	try to infer A's value	fail
	ask what A is	A ← 1
	reject rule 1	A <> 0
• at rule 5	try to infer W's value	
	inspect rules 2, 3 4	
[• at rule 2	reject	A < 8
• at rule 3	fire	W ← 1
• at rule 4]	reject	A <> 0
	reject rule 5	W <> 0
• at rule 6	reject	W <> 0
• at rule 7	try to infer B's value	fail
	ask what B is	B ← 0 + 1
	fire	G ← 2
• at rule 8	try to infer D's value	fail
	ask what D is	D ← 0
	fire	G ← 3
• at rule 10	try to infer X's value	
	inspect rule 9	
[• at rule 9	reject	D <> 1
	reject rule 10 (X defaults 1)	X <> 0
• at rule 11	reject	X <> 0
• at rule 12	reject	A <> 2

The knowledge engineer can in many cases supply additional information with the parameter. A parameter can be described as ask-first. The system will not attempt to infer the value of the parameter but will ask for the information first.

eg if we added the rule (with B ask first):

(13) if A = 1 then B = 0, B = 1

At rule (7) we would still ask for B and not fire (13) to try to infer it.

If the user responds "I don't know" we can still infer the value and continue. Why is this necessary? Because the user may have incomplete or even incorrect information. Some systems can monitor the "expertise" of the user.

EXPLAINING THE REASONING

We can explain the reasoning by adding two user commands: HOW and WHY

- WHY is the system enquiring for this information?
- HOW did the system conclude that?

WHY can be used recursively. Eg:

```
What is A? > Why                          | > How A = 1
(1) if A=0 & B=0 <-> G=0                  | You told me >
> How W=1                                 |
3) if a<>0 (known) then W=1 >             | >How X = 1
                                          | Default >
```

The user can then alter data or even update the rules.

UNCERTAINTY

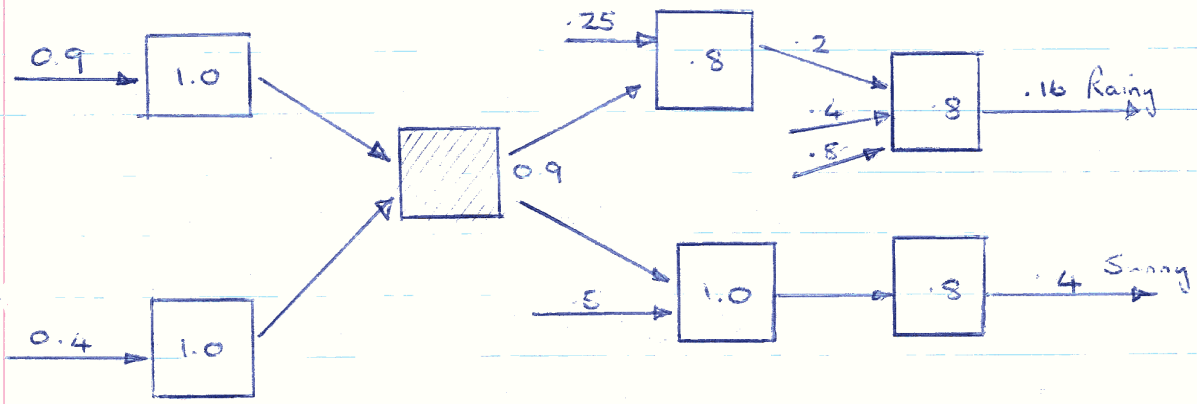
ES are required to work in domains where conclusions are rarely certain. Certainty computing procedures usually generate a value between 0 and 1 associated with each fact (0 - definitely false) (1 - definitely true)

A procedure for calculating certainty must have 3 components:

1. combining input certainties
2. producing an output certainty
3. producing a certainty for a fact which is argued by more than one rule

A simple procedure could be:

1. the minimum certainty associated with the rules antecedents becomes the certainty of the overall input (weakest link)
2. multiply input certainty by rule certainty
3. take maximum certainty preferred by the supporting rules

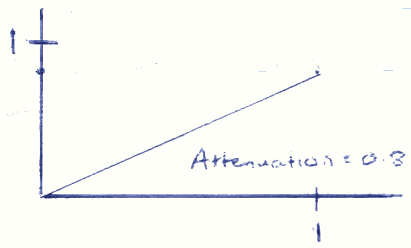


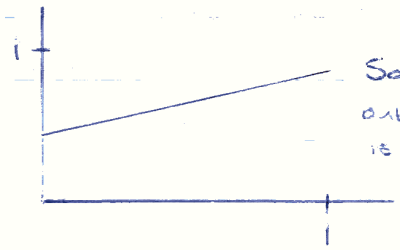
However, this method proves to be too simplistic. A more sophisticated one is:

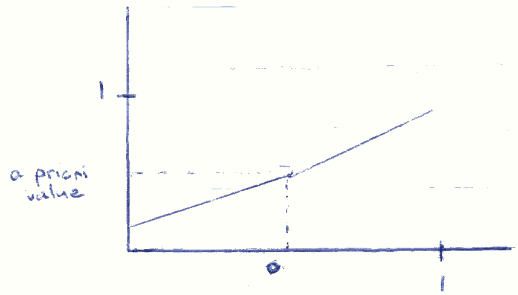
1. certainty of input is product of inputs
2. certainty of output results from a single-valued function with output certainty dependent on input certainty.
3. we transform the certainties into related measurements, certainty ratios, which are put through a simple formula and finally transformed back into a certainty.

The first is based on probability theory for unrelated events - if the events are unrelated we have no support from probability theory.

The second requires the expert to produce a function relating input to output; for example:







We adjust for a priori values (eg if we have no knowledge, the default probability should be on the graph).

In the third case we must produce a certainty for a multi-valued argument. The certainty and certainty ratio (respectively c and r) are related by:

$$r = \frac{c}{1-c} \qquad c = \frac{r}{1+r}$$

The certainty ratio of a multiply-argued consequent is

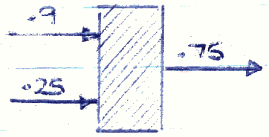
$$r_0 \cdot \frac{r_1}{r_0} \cdot \frac{r_2}{r_0} \cdot \ldots \cdot \frac{r_n}{r_0}$$ where r_0 is the a priori certainty ratio

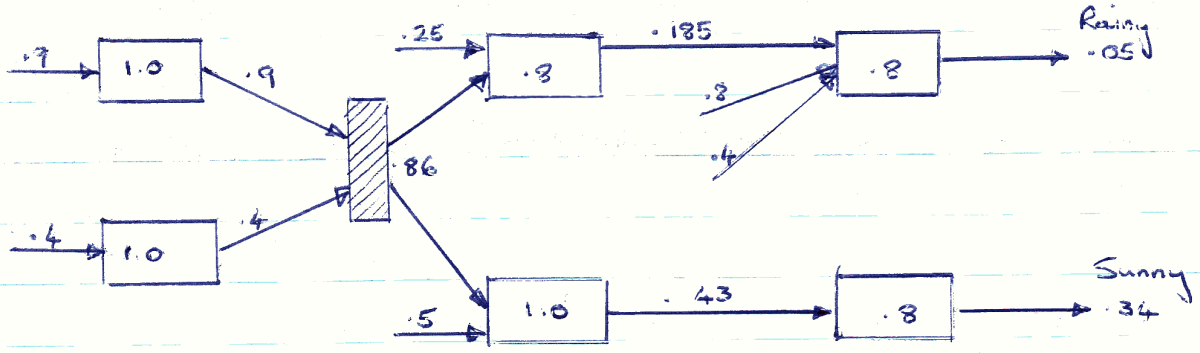
Note that if the the a priori certainty is .5 (typical), r_0 is 1, so this reduces to a product of values.

Example.

1. 

2. 

3. 



KNOWLEDGE AQUISITION

There are several techniques for knowledge aquisition, one of the best being aquiring knowledge from class descriptions. That is, the user supplies both positive and negative samples and the system derives the rules for determining the conclusion. The result is very dependent on the input values given and the closeness of the masses between various interrelating examples.

The best methods of such learning from examples have been produced by Winston, and Michalsky and Chelansky (sp?). The latter two produced a diagnosis system system for soya bean diseases better than that which had previously existed. The methods are similar: we consider Winston's first:

WINSTON'S LEARNING METHOD.

The system initially requires an example whose description forms the initial description. During learning, the initial description is augmented by information which says which links are important in identification. The augmented description is called a model.

A near miss is a sample that does not qualify as an instance of the class being taught for some small number of reasons.

Heuristics for Restricting Models.

There are a number of heuristics which can be used to make a more restrictive specialised model. These are:-

1. The Require-link Heuristic

If the evolving model has a link in a place where a near-miss sample does not, the link is converted to MUST form and becomes a distinguishing criterion between the model and near-miss.

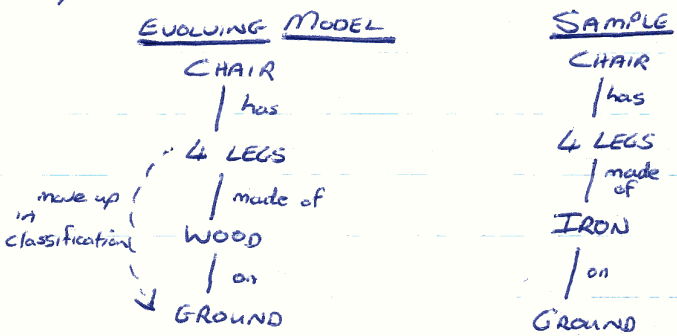
2. The Forbid-link Heuristic

This is used when a near-miss has a link in a place where the evolving model does not. A MUST-NOT link is installed in the evolving model.

Heuristics for Generalising Models

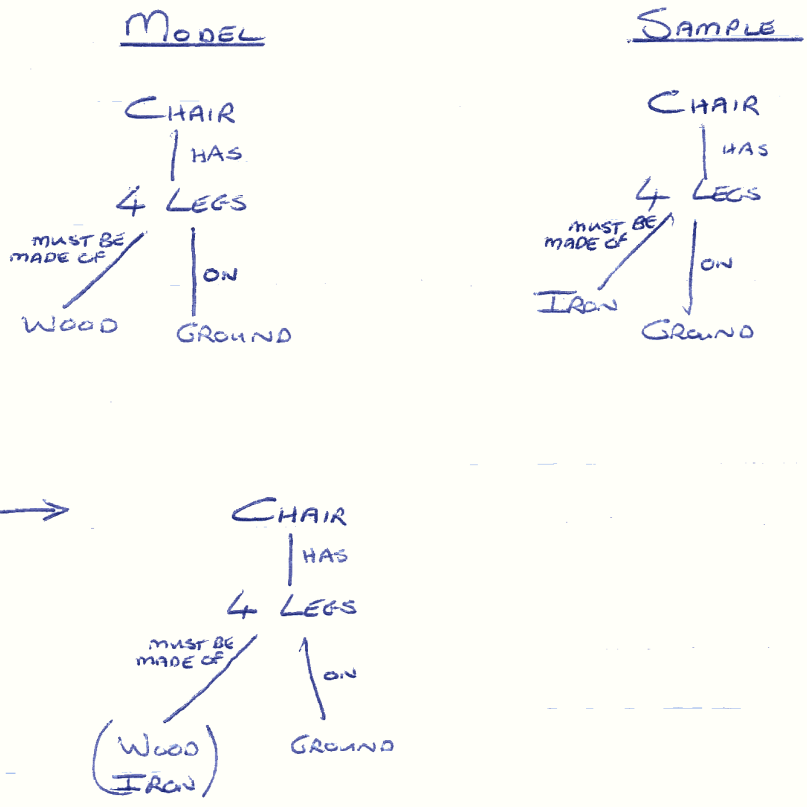
1. The Climb-tree Heuristic

When an example is presented to the system which differs from the evolving model this heuristic is used. MUST-BE links are routed to the most specific shared class in the classification tree above the model item and the example.



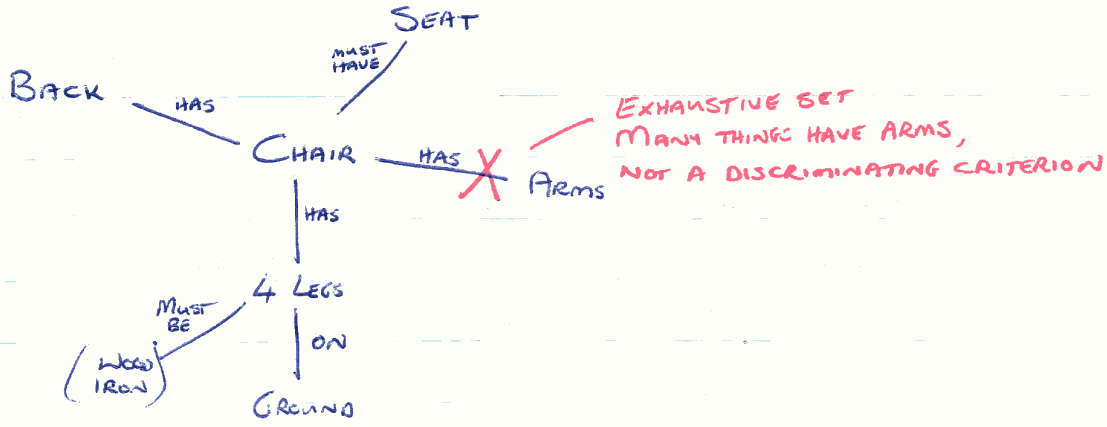
2. The Enlarge-Set Heuristic

This heuristic is used (like the climb-tree heuristic) when something in an evolving model corresponds to a different item in an example. MUST-BE links are routed to the new class composed of the union of the differing classes.



3 The Drop Link Heuristic

This heuristic is used when the things that are different in an evolving model and an example form an exhaustive set, or when an evolving model has a link that is not in the example - the link is dropped.



4. THE CLOSE INTERVAL HEURISTIC

This heuristic is used when a number or an interval in an evolving model corresponds to a different number in an example. If the model uses a number it is replaced by an interval spanning the models number and the examples number. If the model uses an interval, the interval is enlarged to include the examples number.

These heuristics can be used by two procedures:
'specialise', which uses near-misses to refine (restrict) the model
'generalise', which uses samples to expand the model.

Possible Exam Question - How to build a given model

SPECIALISE

1 - Match the evolving model to the sample to (near-miss) establish correspondence among parts

2 - Determine whether there is a single most important difference between the model and the near-miss.

If such a difference exists:

2.1 - Determine whether either the evolving model or the near miss has a link that is not in the other

2.2 - If the evolving model has a link that is not in the near-miss, use the require link heuristic

2.3 - If the near-miss has a link which is not in the model, use the forbid link heuristic.

Otherwise ignore the sample.

NOTES - 'Specialise' does nothing if it cannot identify the most important difference. This can be done by ranking the differences by difference-type or link type. (some types of links / differences are more important than others).

- Both 'specialise' and 'generalise' involve matching. We must have a matcher

GENERALISE

1 - Match the evolving model to establish correspondences among parts

2 - For each difference, determine the difference type:

2.1 If the difference is that the link points to a different class in the evolving model from the class the link points to in the sample, determine if the classes are part of the classification tree.

2.1.1. If the classes are part of a classification tree, use the climb-tree heuristic

2.1.2. If the classes form an exhaustive set, use the drop link heuristic

2.1.3. Otherwise use the enlarge set heuristic

2.2. If the difference is that a link is missing in either the evolving model or the example, use the drop link heuristic

2.3 If the difference is that different numbers or an interval and a number outside the interval are involved use the close interval heuristic

2.4 Otherwise ignore the sample (discard)

Induction Procedures.

A procedure can be designed which learns from sequences of user-supplied examples and near-misses. The first must be an example. The system analyses each sample as it is given and does not keep the samples after they have been analysed.

The procedure is:

1 - Let the description of the first sample be the initial description

2 - For all subsequent inputs
   (a) If the input is an example use procedure 'generalise'
   (b) If the input is a near-miss use procedure 'specialise'.

Axioms of Learning (Winston's method based on these)

1 - When in doubt about what to learn, learn nothing (worse to learn rubbish)

The procedure never unlearns something. If we cannot decide on the difference between a near-miss and an example, then throw the near-miss away.

2 - Do not overgeneralise

When something known to be an example fails to match a general model, create a special case. Avoid all temptations to change a general model; it will most likely break in another case.

3 - Learning is incremental

Martin's law: You can't learn something unless you almost know it already.

The previous system requires the teacher to make an irrevocable decision about the sample order and if the sample is a near-miss. The learner is forced to make decisions about what difference to consider most important and if to ignore a near-miss.
An alternative is to give the learner all the samples at one time. This permits the learner to search for the best sample order and to search for the best difference to use. (Note: often we are dealing with an expert who may be a poor teacher; this can cause problems with Winston's method) This second method is more powerful but less humanlike. There is no incremental learning. All samples are available when learning starts and all are retained until learning stops.

LEARNING PROCEDURE

The second learning procedure has two steps.

1- Start with an empty description that matches everything. An empty description has no MUST or MUST-NOT links. It will match everything.

2 - Try to specialise the description by adding MUST and MUST-NOT links until none of the wrong things match.

3 - Try to generalise this description in order to match as many class examples as possible while never generalising so as to start matching any non-members.

Before giving the abstract description, let us consider an example. The number of samples is left small for obvious reasons.

We have two chair samples:

(S1)
4 legs
Square seat
Back

(S2)
4 legs
Round seat
Back

We distinguish between them by saying

S_1 is square
S_2 is round

(stored in DIFFERENCE MATRIX)

We have 3 examples of near-misses to chair

(S3) 4 legs
Square seat
Back
Seat on floor

(S4) 4 legs
Square seat
Back
Legs touching

(S5) 4 legs
Ring seat
Back

We can describe the differences between the examples and the near-misses by the DIFFERENCE MATRIX

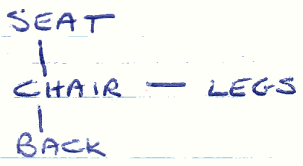
	1	2
3	L SUPPORT S	S_3 IS SQUARE S_2 IS ROUND L SUPPORTS
4	L TOUCH	S_4 IS SQUARE S_2 IS ROUND L TOUCH
5	S_1 IS SQUARE S_5 IS RING	S_2 IS ROUND S_5 IS RING

The differences can be classified as coupled or ordinary. There is one coupled pair of differences between 1 & 2, and either:

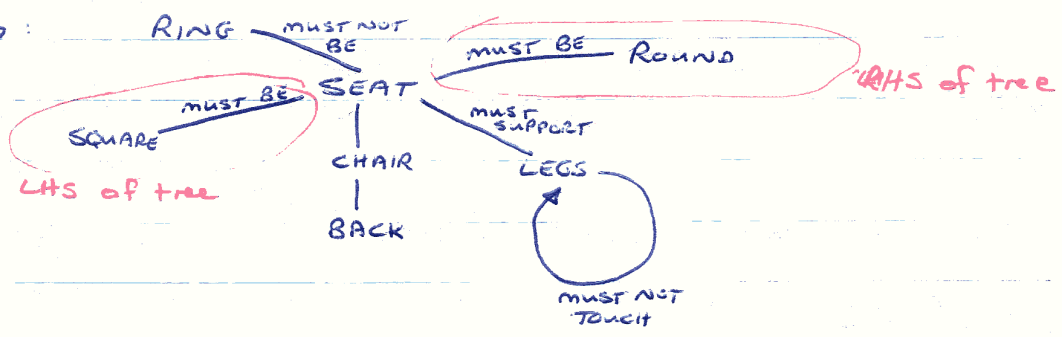
- one coupled pair
- one ordinary, or
- one coupled pair and one ordinary difference between the samples and near misses.

We begin by selecting a sample, say S1. A near miss is also selected. The process can be described by the tree structure opposite. To limit the explosion of the tree we evaluate the degree of constraint. If we take a <u>support</u> branch we add 4 constraints to our naked initial model; similarly for <u>touch</u> branches. If we take a <u>round</u> or <u>ring</u> branch we add only one constraint. We prune off the branches with the least constraints (indicated by x).

The initial description:



becomes:

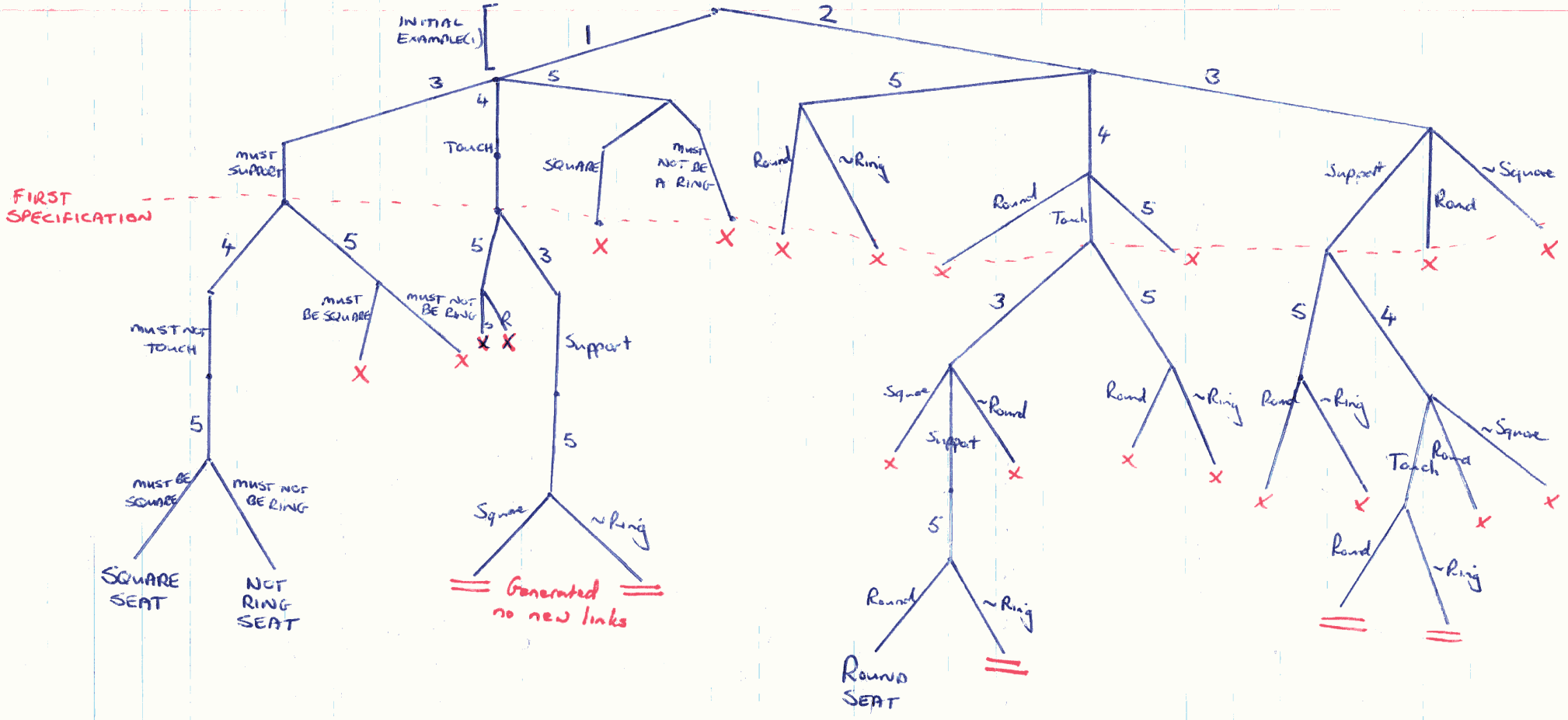


We are left with 3 distinct results:

MUST_BE_SQUARE_SEAT

MUST-BE-ROUND-SEAT

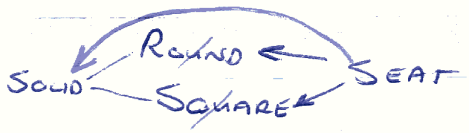
MUST-NOT-BE-RING-SEAT


INITIAL EXAMPLE(1)
1
2
3
4
5
FIRST SPECIFICATION
MUST SUPPORT
TOUCH
SQUARE
MUST NOT BE A RING
Round
~Ring
Support
~Square
MUST NOT TOUCH
MUST BE SQUARE
MUST NOT BE RING
MUST BE SQUARE
MUST NOT BE RING
Square
~Round
~Ring
SQUARE SEAT
NOT RING SEAT
Generated
no new links
ROUND SEAT
X - pruned
= - redundant

Step one is now complete as all the non-members have now been excluded using "specialise" and the require-link and forbid-link heuristics.

We can now commence step 2 - the generalisation. One example of a chair remains imperfectly matched along two of the three links. The problem is the MUST-BE links. If we assume that square and round are in the same class as SOLID, then we would use the climb-tree heuristic to move the link to solid; otherwise the enlarge-set heuristic would be used.

CLIMB-TREE

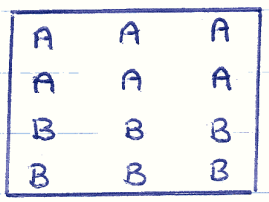


ENLARGE-SET

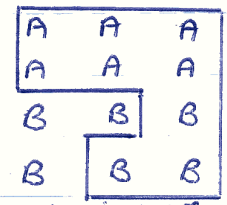
Let us now consider this procedure theoretically:

In step 1 a class member is first selected. A non-member to be excluded is also selected. This is excluded by an addition to the class description under construction. A difference is selected and a heuristic used to augment the description. Non-member exclusion continues until all non-members are excluded. Some members may be excluded at the same time.

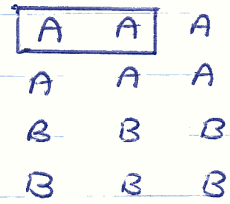
eg



General description



By selecting 1 difference can 'knock out' many



Left with small subset of members

Step 2 begins. The description is generalised. The description is used with the unmatched members to drive the generalisation heuristics. NO generalisation is permitted if it would include a non-member.

There may be no way to make the description match all members. The matched members are put aside. One of the remaining members is selected and the specialisation / generalisation process begins again, creating a new class

There are many decision points in this process The decision tree can be very large:

eg for m members
n non-members
d differences

the size of the tree would be $m!\, n!\, d\,(m-1)!$

The procedure is: (similar to Winston, but less restricted)

Carry out all the search procedures in a pruned-tree search manner using an "evaluation function".

Select one class member that is not yet described. This is the current class member.

Start a description with only nodes and no links. Call this the current description

SPECIALISING:

Repeat
    Select a non-member
    Select a MUST producing difference between the member and non-member
    Use 'specialise' and the difference to add a link to the current description
Until all non-members are excluded

GENERALISING

Repeat
    Select a class-member not yet described
    Attempt to generalise the current description using 'generalise' and the selected class member.
    Attempt fails if we include a non-member
Until all such class-members have been selected.

The technique has been used in practice in the Soya bean expert system, and has been found to be very efficient.

So far we have discussed the state of production rule systems up to the early 1980's. The knowledge was contained in the production rules and a database of facts - unrelated items. The inference engine could chain backwards and forwards to produce the desired results. The system could have some means of aquiring knowledge.

About 1980 it was realised that the production rule approach to knowledge, and the resulting differential diagnosis approach that results, could not solve all the problems of the world. We need to consider more advanced knowledge representation techniques.

The field of natural language processing presents certain techniques. Because of ambiguities associated with natural language it was required to build a model of the world and to use this as the back end for the parser and semantic analyser.

Natural language processing approaches include:

- frames
- semantic nets
- object-oriented systems.

6 FRAMES

A frame is a prototype for some object or event. The strength of a frame is that all the elements that are normally present in the description of the object are grouped together and thus may be accessed or processed as a unit.

eg

frame
name : ~~
type of : ~~ (= is a)
name ~~
age : ~~
salary : ~~
debt : ~
alive : ~

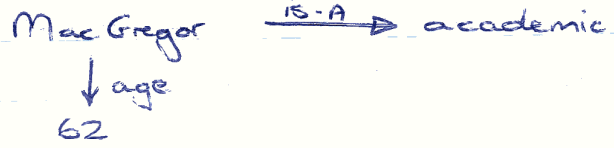
Frame names or data names appear in the fields (with links to other entities of loop). Fields can also be programs called compute fields, whose value is the result of the computation.

Instantiation of a frame produces a specific event with a specific name. The specialisation frames can be grouped into hierarchies, and the hierarchies searched via the link fields. This is an attempt to produce the necessary network associated with real world knowledge. The area of frames led in large to the idea of data abstraction in programming languages.

Real-time : Ada
AI : Smalltalk, Flavors Loops.

7 SEMANTIC NETWORKS

Semantic networks comprise of nodes and links. Each node is a concept; each link is a relation. Thus we can have:



The important feature of semantic networks is the inheritance of properties. The specific element inherits all the properties of a general class. The links can be moved by learning algorithms between nodes to permit more or less generalisation depending on the class examples.

This permits the knowledge to be represented in a real-world fashion and exist "temporarily" or be destroyed as the data in the model changes.

Semantic networks are once again easily programmed in LISP using atom/property lists. This accounts for a large part of LISP's popularity in AI development.

8 ARCHITECTURES FOR AI

Before considering possible AI architectures, let us consider the tools for AI and expert system usage. There are two different approaches to AI development. The first is a traditional approach developed over 20 years and considers Lisp and functional programming to be the correct approach for AI development. (multiprocessors)

The second and more modern approach based originally in Europe but more recently in Israel and Japan regards PROLOG and logic programming as the correct approach (asynchronous processor)

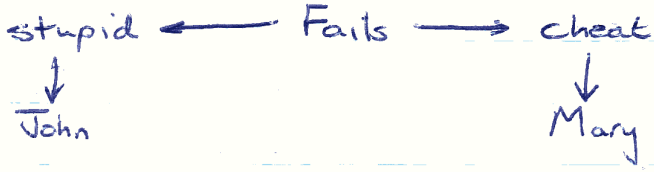
These two groups are opposed by a third group originating in Europe which argues that neither of the language is suitable per se but rather a novel architecture should be designed and the language then built on this architecture (new architecture)

PROLOG

Prolog is a logic-programming language which permits facts to be entered and deductions made from these facts. Effectively, Prolog is implemented by a number of linked lists each containing information about a fact.

eg John is stupid ≡ stupid (John)
Anyone who is stupid will fail ≡ Fail(x) :- stupid(x)

Could be implemented as



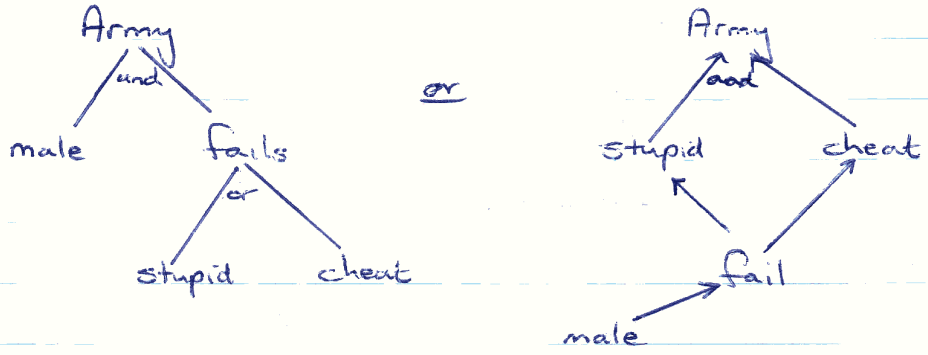
The system can search a list to produce facts which are then put into another search on another list to see if they match.

eg: army(x) :- male(x) $\overset{(?)}{\vee}$ fails(x)

Here we could do a search of our data base on the list ~~made~~ male and return each elt to carry out a search for the fails condition

The conceptual model is that of asynchronous processes. The first scans the list of males and returns values to the next process which scans the list for fails. This scanning creates two processes - one to look for stupids and one for cheats.

eg



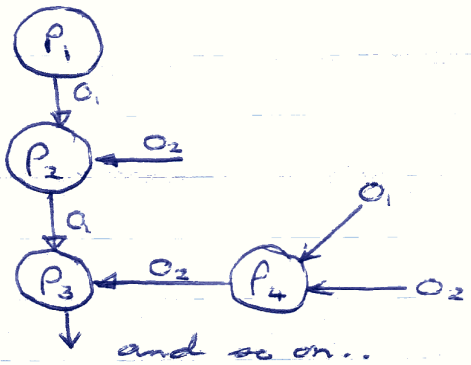
To implement Prolog effectively we require an asynchronous architecture

Dataflow Architecture

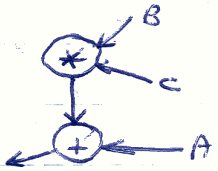
The Dataflow architecture is a parallel asynchronous architecture. It has a number of processors, each of which waits until its operands are ready before commencing processing.

Conceptually it is a pipeline processor where each stage of the pipeline is a processor and the operands flow from one processor to another during the computation.

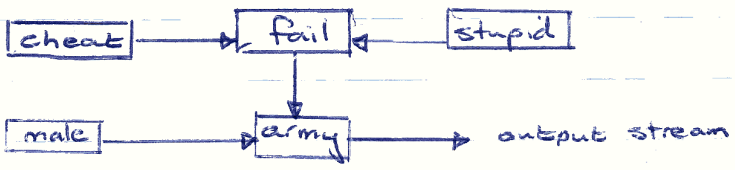
The Dataflow architecture can be represented as:



Each processor passes the operands to another processor to be used in its calculation. It can be used in arithmetic calculations, eg A + B * C



It can be used together with Prolog to scan lists, eg.



Each connection between nodes is a stream of facts. The processors can then join these facts together depending on whether an AND or OR is required, and pass the stream on to the next node.

Dataflow architecture can be regarded as a physical implementation of the Unix pipe, with each process accepting a text stream from the neighbouring processes

The implementation of Dataflow machines can be:

- static architectures
- reconfigurable static architectures (set at load time)
- dynamic architectures

The last is the most interesting. Processors can be defined as
- Database access processors (type 1)
- AND/OR processors. (type 2)

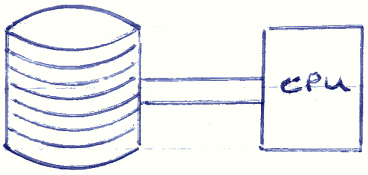
The high-level processors (type 2) control slave processors (type 1). For example, the process fails can be considered as a type 2 process which can trigger two type 1 processes to run on processors, the result being ANDed and passed to the next processor.

To run Prolog effectively the machine must have adequate processors to handle all the processes. For a complex query we may have many subgoals and a great complexity of processes running. The system must thus have many processors, at least 256.

The processors must be dynamically configurable (of the Connection Machine a bit later)

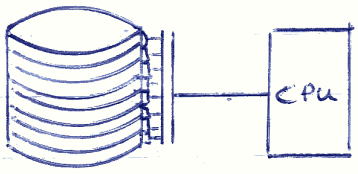
Two approaches to improving throughput:

IBM



High bandwidth bus with large transfer capacity.

ICL



CAFS - content addressable file store

Low bandwidth bus
Processors (type 1) attached to each track head retrieving only useful data.

LISP MACHINES

Lisp has certain properties which make it unsuitable for execution on conventional architectures:

- Dynamic binding (interpretive code - variables bound at run-time; compilers do not always produce the same results)
- Interactive (cannot be run in a batch environment)
- No seperation between programs and data. All are lists and can be operated on by any command.
- Garbage collection. Lisp is based on a dynamic heap construct which requires periodic garbage collection - difficult on most computers.

Conventional Operating Systems.

Most computer systems execute compiled code efficiently with great inefficiencies for interpretive code. (>500%)
The level of user-interaction is generally very poor - it occurs when using editors, debuggers, etc. The full level of interaction required by an incremental development language such as Lisp is not available.
Most O/S split program and data. The executable code is usually re-entrant code not accessible by the user program. In LISP we require to access and modify a function and then execute it.
Most operating systems do not perform garbage collection it is left to the user application or the run-time package to manage its own heap space.
In LISP it is meant to be primitive to the O/S and totally transparent to the user.

LISP Machines

These are in many ways "conventional architectures". They have a processor and memory but only run LISP programs. The run-time package ie debugger, garbage collection etc. is microcoded into the machine

Features:

- Lists tagged for use by reference counts and freed when these become zero.
- Large single-user machines with virtual segmented storage (typ. 5 MB)
- Excellent debugger - if an error occurs we fall into the inner level of procedure and from the top we fall into the debugger (always resident)

- Communication via networks to the file servers (eg Ethernet, file servers at back?) built around individual users.
- Windowing functions
- Graphics (eg icons, mouse)
- Object-oriented facilities.

CONNECTIONS

When we consider brains, we find that although neurons are much slower than gates, the brain is still much faster than conventional computers. The theory is that this is due to a high degree of parallelism in the brain. This has prompted to types of architecture:

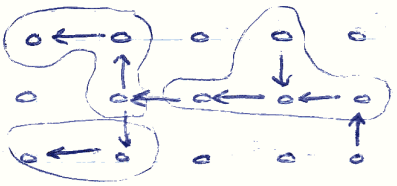
- the Butterfly connection
- the Connection machine

The Butterfly connection architecture is MIMD (Multiple instruction multiple data). Many powerful processors are hooked together with their own memory devices. Memory accesses can be "cascaded"; thus the further the information lies away, the longer it takes for the processor to access it. Experiments in this direction currently being undertaken involve connecting up 256 or more Motorola 68000's.

The Connection machine is a SIMD (Single instr, multi.data) machine consisting of an enormous number (currently 256K) of 1 bit processors. There is no distinction

between memory and processors - each bit of memory has (is) a processor on top of it. If we then wished to have for example, a 16-bit adder, we could hook together 16 processors sequentially.

In both types of connection machine, we can't use conventional busses as our communication medium, as bottlenecks occur and after about 6 processors share a bus additional processors are virtually ineffective. Instead the processors are connected in a n-dimensional "grid", with direct communication possible with immediate neighbours only. Longer communications are routed through adjacent processors



grouped into "tasks".

The system should be dynamically reconfigurable, with groups of processors being allocated to different tasks. These architectures will result in new types of languages, operating systems, etc.

APPENDIX A : PROSPECTOR & MYCIN

Both of these expert systems operate on well-organised knowledge bases - infectious disease knowledge (MYCIN) and geological models (PROSPECTOR). Both deal with uncertainties, and must thus have inference mechanisms based on measures of belief. MYCIN uses certainty factors and the theory of confirmation; PROSPECTOR uses conditional probabilities and Bayes' Theorem.
Both systems have been designed with relatively simple control structures, with architectures based upon the production system approach and consisting of:

- a collection of facts
- a set of production rules
- an inference engine together with a knowledge structure which enables the control structure to decide which candidate rules should take part in the inference mechanism.
- a mechanism for drawing inference from uncertain or incomplete evidence.

In MYCIN, groupings of rules within a particular context are defined at generation time; in PROSPECTOR, the rules are described in terms of a network in which the set of rules to be considered are those defined by the exit (certainty) arcs of a node. The inference engine in each case provides goal-directed searching (backward chaining: a rule is used to derive more information about its premises) or data-directed searching (forward chaining: a rule is used to cause alterations to the database via its action part).
Both systems allow an intermixing of the two modes but are mainly backward chained.
While both systems have domain-dependent control structures,

attempts have been made to apply the approach to other knowledge areas with similar characteristics (hence EMYCIN ("empty" Mycin) is the basis of PUFF, SACOM, GRAVIDA, CLOT and VM; PROSPECTOR has been used for HYDRO and implemented "empty" as SAGE and MICROEXPERT). In all these cases, however, the knowledge domain must be capable of being cast into MYCIN or PROSPECTOR-like representations.

A.1 MYCIN (EMYCIN) OVERVIEW

MYCIN was developed at Stanford with the aim of identifying and treating infections diseases

A.1.1 FACTS

These are <context, parameter, value> triples. A context is some real world entity (eg a patient), a parameter is an attribute of the context (eg age) and the value is an instance of a parameter (eg 25 years). Each part triple has an associated certainty factor (CF) ranging between -1 (negation) and +1 (certainty). At the start of a consultation there will be many triples, some of which will be incomplete (eg goals).

A.1.2 PRODUCTION RULES

IF <premise> THEN <action> (<CF>) The premise is some conjunction of triples and the action usually involves the instantiation of a triple. The CF is used in conjunction with the CF's of the premise triples to calculate the CF of the action. A rule can be applied either

- from known premises to instantiate action triples (forward c)
- from a known action triple to determine what premise triples need to be determined (backward c)

In MYCIN rules are defined to be either forward or backward chained. The rules are repeatedly applied until the goal triple(s) is instantiated. Domain-dependent knowledge determines which rules to apply. (see Alty & Coombs pp 95-6 for eg)

A.1.3 CERTAINTY FACTOR CALCULATIONS.

The CONCLUDE predicate in MYCIN rules adds a triple to the database with a computed certainty factor based on the CF's of the premise, the rule, and the triple (if it had a previous CF).

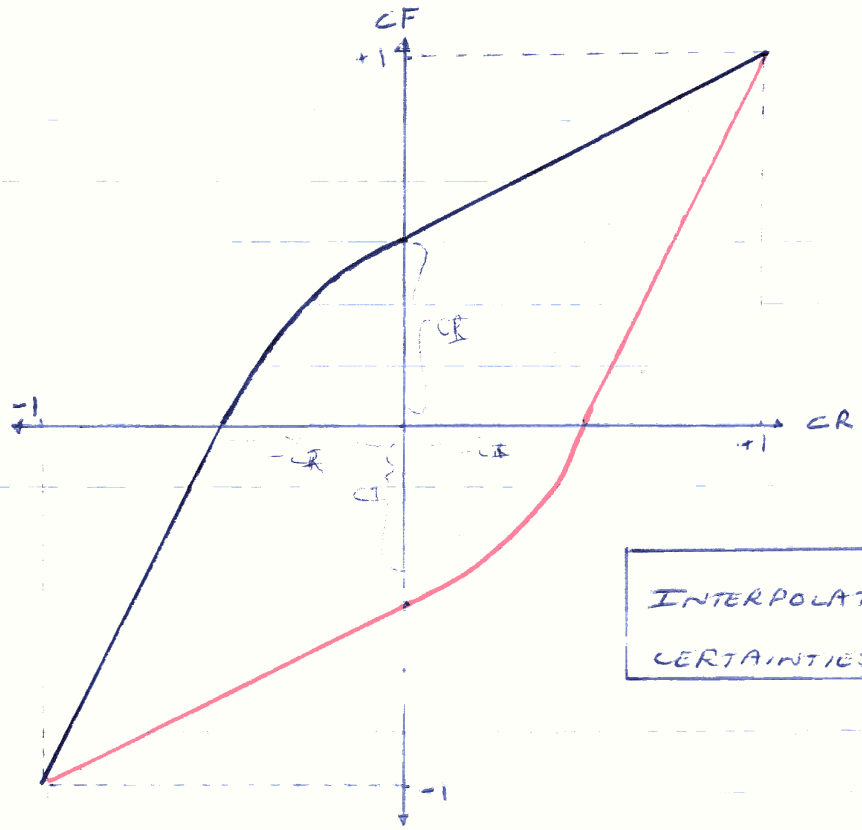
The CF of the premise is the minimum of its clauses. This is then multiplied by the CF of the rule to give a cert part CR. If the triple did not exist previously then CR would become its CF. If, however, it already existed with a CF of CI, the new CF is calculated as

$$CF = CI + CR(1 - CI) \qquad CR, CI > 0$$

$$CF = -(|CI| + |CR|(1 - |CI|)) \qquad CR, CI < 0$$

$$CF = \frac{CI + CR}{1 - \min(|CI|, |CR|)} \qquad CR \cdot CI < 0$$

Combination of 1 and -1 is defined to be 1. A typical interpolation graph is shown opposite



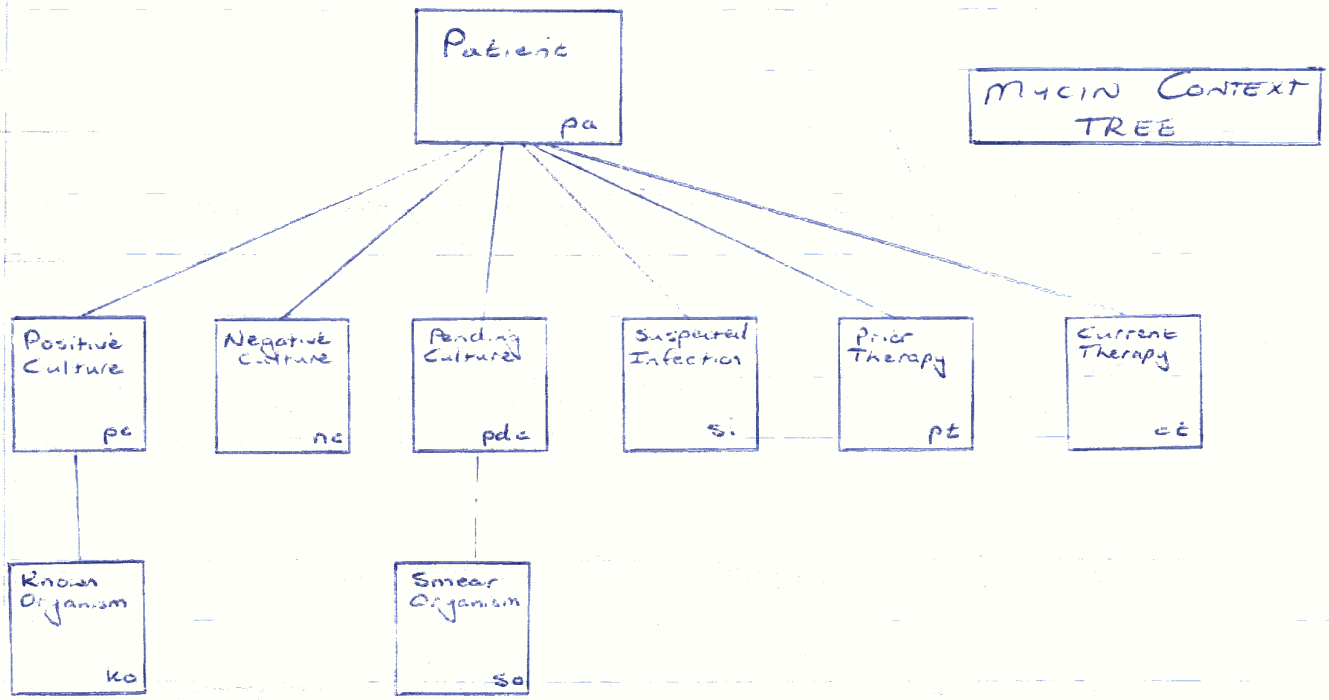
INTERPOLATION OF CERTAINTIES IN MYCIN

A.1.4 Domain Dependent Control Structure

The control structure consists of an inference engine for forward or backward chaining, together with a knowledge structure which enables the system to decide what rules to try next. The ordering of the rules is not significant. All relevant rules are applied until one particular rule concludes a goal with certainty, in which case the rest of the rules are ignored. The relevance of a rule is defined by the domain dependent control structure. MYCIN uses mainly backward chaining, although a few rules are defined to apply in forward chaining mode only.

The domain dependent control structure is a context tree which organises the database triples hierarchically and directs the consultation. The root is the starting point for the selection of initial goals which the system attempts to satisfy. The tree is a hierarchy of context types (or templates) and the actual triples which exist or are created during the consultation form a

structure which mimics the context tree. A rule always has a context; rules are thus grouped into rulegroups defined by a set of applicable context types. All rules belonging to a rulegroup referring to a context of interest are considered for firing. A MYCIN context tree is:



The context tree, by defining hierarchical relationships between contexts and parameter groupings, thus steers the rule interpreter during the consultation. Furthermore, since the context tree reflects the relationships of objects in the domain it helps to provide a consultation system which is familiar to the user.

A.1.6 OBTAINING INFORMATION FROM THE USER.

Questions are asked either when the rules fail, in order to determine necessary data, or when the data is defined as having to come from the user (ie lab data). A question can be elaborated if it is not understood. An ASKFIRST property tells the system always to endeavour to find out the

value of a parameter from the user first. The user interface in MYCIN proved unsatisfactory and a different approach was used in ONCOCIN.

A.1.6 EXPLANATION FACILITIES.

At any time the user may question HOW or WHY. When asked WHY, MYCIN lists the action part of the rule. It then lists the premises which are required to be instantiated by the question. Repeated use of WHY calls for display of the next rule in the current reasoning chain. HOW acts in the opposite direction, showing how MYCIN came to a conclusion.

Whilst the questioning facility in MYCIN is quite powerful, it is limited to reasoning about the immediate vicinity of the MYCIN reasoning state during the consultation. Furthermore there are many knowledge aspects which MYCIN knows nothing about, ie it has no explicit causal knowledge of the domain.

A.1.7 TEIRESIAS AND GUIDON.

The objective of T. was to reduce the role of the knowledge engineer by allowing the domain expert to interact directly with the system. The system is concerned with three key issues

- comprehensibility - the end user must be able to understand
- debugging - What if expert disagrees with system?
- knowledge elicitation

T. provides explanations which can be examined and debugged, and allows the knowledge base to be added to or changed. T. thus requires knowledge about itself - metarules

The meta-rules contain information about how it reasons and how much it knows. The system also has 'rule models' abstract descriptions of subsets of rules which have characteristics in common. It therefore has expectations about rules and will check with the user if these expectations aren't satisfied. The models are organised in a tree and T starts at the top descending until it finds the closest fit. The system also has additional natural language facilities (essentially keyword based) which improve interaction with the user.

GUIDON was developed to use the explanation facilities of MYCIN as a teaching tool. It contains 200 tutorial rules containing additional knowledge which guides the tutorial, constructs a student model and responds to initiatives from the student.

A.2 PROSPECTOR OVERVIEW

Prospector was designed to:

- evaluate sites for the existence of certain deposits
- evaluate geological resources in a region
- select favourable drilling sites.

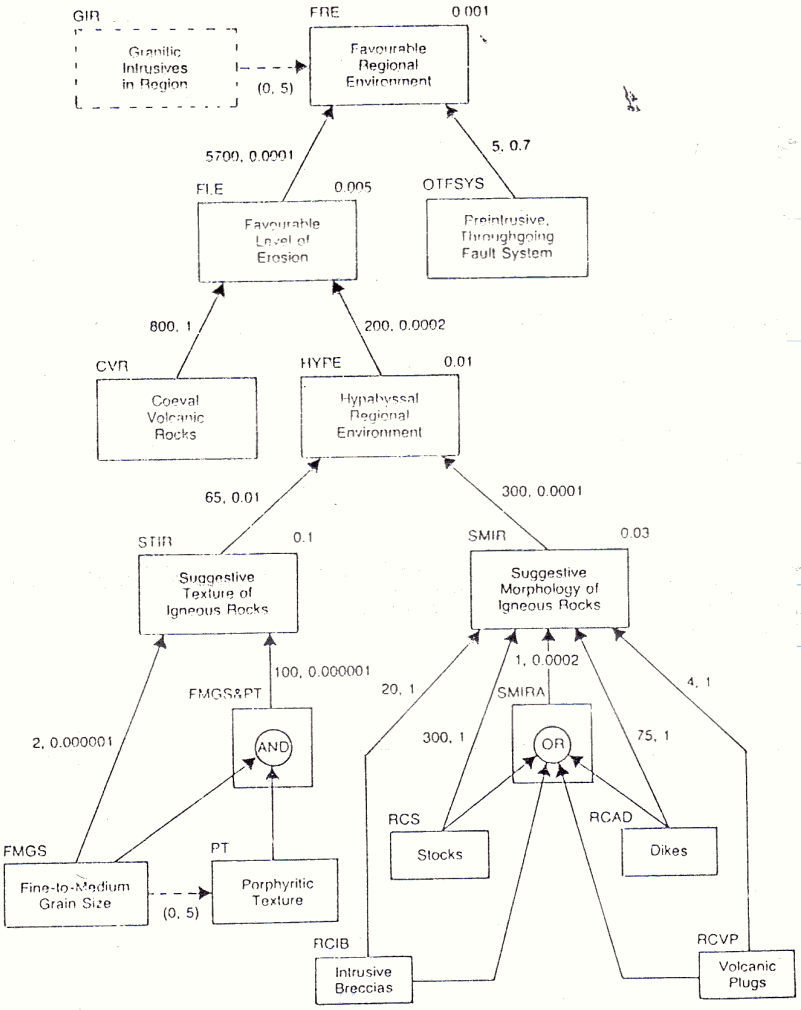
Expert geologists apply highly specific models when identifying likely sites for ore deposits. In P. these models are systematically interpreted to give site evaluation advice.

A.2.1 GEOLOGICAL MODELS.

A model (15 are mentioned in the final report) consists of spaces connected by rules. A space may be some observable evidence or a hypothesis, and each space has a probability value indicating its 'truth' content. At the start

of the nm these probabilities are usually low. Rules specify how a change in the probability of one space affects another.

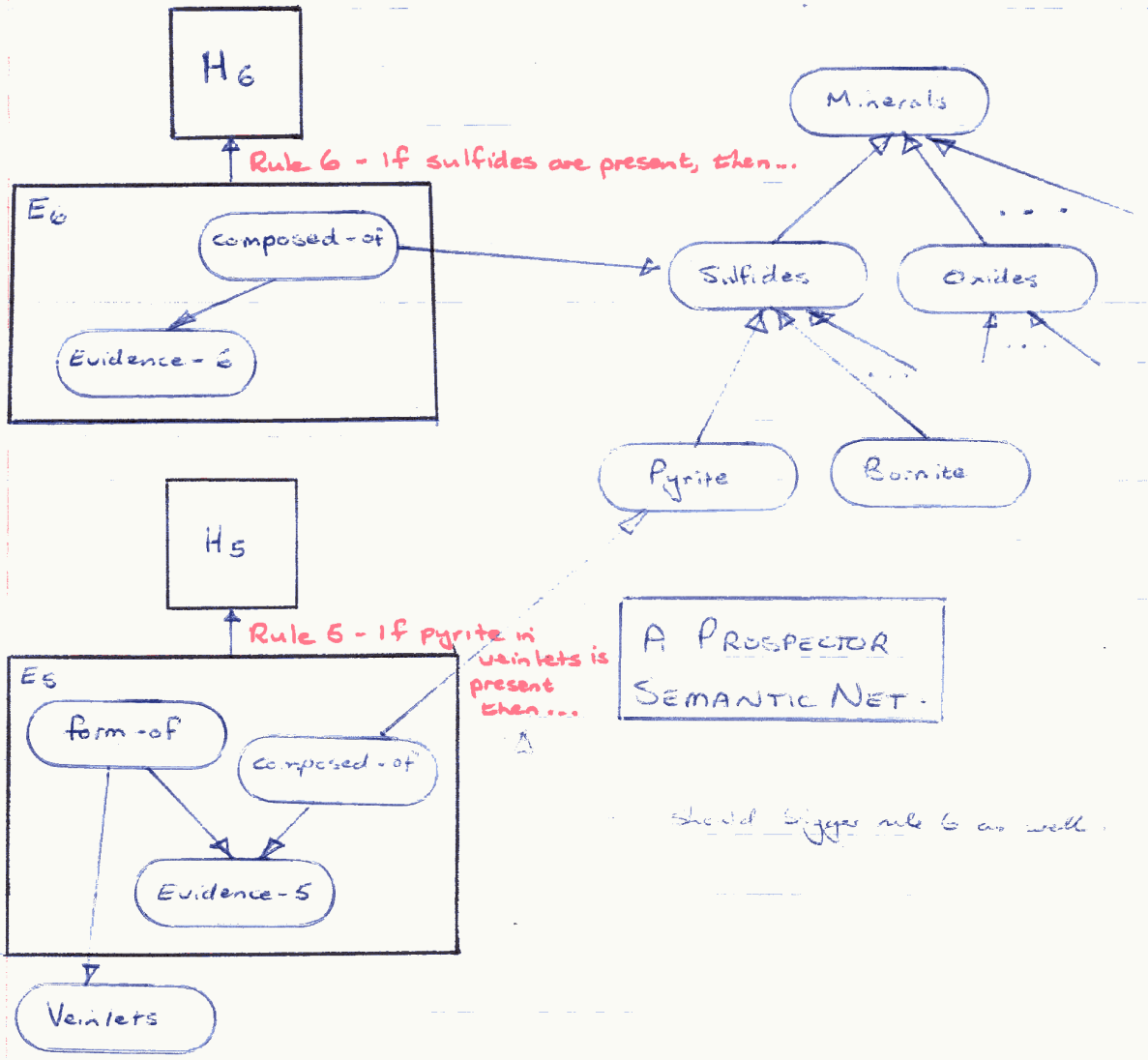
A model is built up by connecting spaces with rules in the form of a network. In the network rules may be and'ed, or'ed or not'd. Each time a space probability is altered, its effect is propagated through the network.



PART OF A PROSPECTOR MODEL

Each space is labelled with its prior probability and the rules have two factors which determine their strength - a sufficiency factor and a necessity factor. The entire model deals with probabilities of existing spaces. Spaces are not created or instantiated, only their probabilities are changed as evidence is collected.

The spaces at the base of the model elicit information from the user. Because the spaces are related and the rules are context sensitive, P. has a taxonomy of minerals stored as a semantic net.



should trigger rule 6 as well.

The square boxes are spaces in the inference net. Each space can have an internal structure and may be connected to an external one. The additional semantic network enables the system to deduce that pyrite is a sulfide.

In P. therefore we have a semantic network of nodes and arcs which is further partitioned into higher level spaces which form the nodes of the inference network. Nodes which are unique to a space are included within the space and nodes which can be referred to elsewhere appear outside

A.2.2 CONTROL MECHANISM

The control mechanism supports a mixed initiative interaction. The user can take control at any time using certain commands. The user can volunteer observations or ask PROSPECTOR to investigate a particular hypothesis. When P. is in control it decides which hypothesis to work on and uses a depth-first strategy to traverse the inference net. Where there is a choice of rules it will select the one with the greatest potential for increasing the odds on the hypothesis (or ruling it out). Ineffectual questioning is minimised. P. can rephrase questions and explaining why and how. Rules can be employed in either direction.

A.2.3 RELATIONSHIPS AND UNCERTAINTY

In P. there are three types of relationships between spaces:

- rule strength using necessity & sufficiency factors
- logical relationships for combining assertions
- contextual relationships indicating an ordering of assertion determination

A.2.3.1 RULES

A rule relates a source space (evidence or assertion) with a target space (hypothesis) using a rule strength comprising two numbers, the necessity factor (LN) and sufficiency factor (LS). A P. rule takes the form

IF <evidence> THEN <hypothesis> (<LS>,<LN>)

LS is the certainty ratio of the hypothesis being true given the evidence (how SUFFICIENT is E for H?)

LN is the certainty ratio of the hypothesis being true given that the evidence is false (how NECESSARY is E for H?)

ie $LS = \frac{P(E:H)}{P(E:\sim H)}$ $\quad LN = \frac{P(\sim E:H)}{P(\sim E:\sim H)}$

where $P(x:y) \equiv$ probability of x given y true.

LN and LS are related by: $LN = \frac{1 - LS \cdot P(E:H)}{1 - P(E:\sim H)}$

BAYES' THEOREM: $P(H:E) = P(E:H) \cdot \frac{P(H)}{P(E)}$

or: $O(H:E) = LS \cdot O(H)$ (the odds version - tells how odds of H change with E)

If a rule is likely to be true, then $1 \leq LS \leq \infty$ and $0 \leq LN \leq 1$
Generally, if $LS > 1$ then $LN < 1$ and vice-versa. If $LS = 1$ then $LN = 1$

Example: Let E be "it is snowing", and H by "it is cold".
Then $P(E:H)^{high}$ is the probability of snow falling when it is cold
$P(E:\sim H)^{low}$ is the probability of snow falling if it is not cold
$P(\sim E:H)^{mod}$ is the probability of snow not falling if it is cold
$P(\sim E:\sim H)^{high}$ is the probability of snow not falling if it is not cold

If we assume that the evidence is true ($P(E) = 1$) then Bayes' Theorem reduces to $P(H:E) = P(E:H) \cdot P(H)$

P(H) = 0.5

Thus if the prior probability of it being cold was 0.5, it is snowing (P(E)=1), and the probability of snow falling when it is cold is 0.5, then

the probability of it being cold given that it is now snowing is $P(H).P(E:H) = 0.5 * 0.5 = 0.25$.

Although LN and LS are related, experts do not agree on whether this is true. Often an expert will declare "The presence of E enhances the odds on H, but the absence of E has no effect", ie LS > 1, LN = 1, which is inconsistent. P. handles this inconsistency - LS tells us how to alter the odds of H if E is true; LN tells us how to alter the odds of H if E is untrue. One or the other is selected depending on whether E is true or untrue.

A.2.3.2 UNCERTAIN EVIDENCE

What if E or ¬E are not known with certainty? We could use interpolation. A problem with this is that when the network is set up, the expert assigns LN/LS values to each node in a subjective manner, and the network is thus not mathematically consistent.

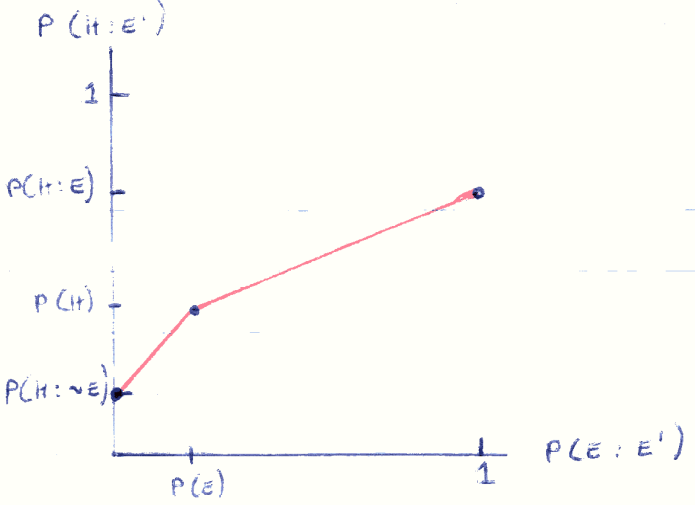
Let E' be the evidence causing the user to suspect E. This alters P(H) to P(H:E') which lies between P(H:¬E) and P(H:E). Then

if P(E:E') = 0 then P(H:E') = P(H:¬E) ①

if P(E:E') = 1 then P(H:E') = P(H:E) ②

If we know nothing about E (ie P(E:E') = P(E)) then the odds on H should not change. ie P(H:E') = P(H) ③

These three points provide the relationship between P(H:E') and P(E:E') (see over.)

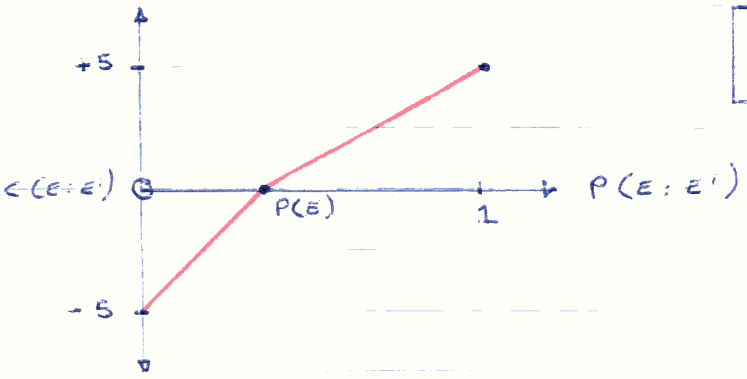


DEALING WITH UNCERTAIN EVIDENCE

This technique eliminates the propagation of erroneous probabilities through the network due to inconsistent LS/LN values. The same technique is used to force agreement with user input probabilities and the prior odds. The user gives a certainty factor $C(E:E')$ between -5 (negation) and 5 (affirmation). The principle is to use the three points:

- if $C(E:E') = -5$ then $P(E:E') = 0$
- if $C(E:E') = 0$ then $P(E:E') = P(E)$
- if $C(E:E') = 5$ then $P(E:E') = 1$

Thus:



RELATIONSHIP BETWEEN CERTAINTY AND PROBABILITY

and: $P(E:E') := P(E) + C(E:E') \cdot \beta / 5$

where $\beta = \begin{cases} 1 - P(E) & \text{for } C(E:E') > 0 \\ P(E) & \text{for } C(E:E') \leq 0 \end{cases}$

Example: If $P(E) = 0.2$ and $C(E:E') = 2$,

then $P(E:E') := 0.2 + \frac{2}{5} \cdot (1 - 0.2) = 0.52$

A.2.3.3 LOGICAL RELATIONS (AND, OR, NOT)

Fuzzy set theory is used.

AND - take minimum probability

OR - take maximum probability.

NOT - negate probability

A.2.3.4 CONTEXTUAL RELATIONS.

Some hypotheses cannot be considered as being in an arbitrary order - certain evidence must be available before other evidence is sought. Certain target spaces must therefore be determined before the "consequent" spaces can be examined (dashed arrows on P. 47 models)

A.3 CONCLUSIONS

APPENDIX B - ATTRIBUTES OF A GOOD ES DOMAIN

- The task requires expertise and there is a need to 'capture' the expertise
- Conventional algorithmic techniques are unsatisfactory
- Symbolic reasoning and heuristics are required
- A recognised expert exists and will cooperate
- A reasonable payoff is expected
- The task is neither too trivial nor too ambitious
- The task is well-defined and well-bounded
- A system performing a portion of the total problem would be useful
- The task is decomposable for development.
- Incorrect or non-optimal results can be tolerated
- The task definition is fairly stable
- Any real-time response requirement will not involve extensive effort
- The user-interface will not require extensive effort.

Appendix C - Issues in selecting an ES dev. tool.

- Can it be easily obtained & installed? (cost factors, compatibility with existing, legal arrangements)
- How well is it supported? Will later upgrades be backward-compatible? Is the current version sufficiently stable?
- Has it been used before successfully in a variety of domains?
- Expansion? Source code?
- Compatability with other s/ft systems (eg Lisp)
- Knowledge representation? (Rules, networks, frames, other?) How well does it match the intended application? (eg rules for empirical associations, networks for complex interrelationships)
- Can it handle the expected form of data? (inconsistencies, etc errors, uncertainties)
- Do the inference mechanisms suit the problem?
- Does the allowable granularity of the knowledge match what is needed by the problem?
- Does the expected speed of the developed system match the problem if real-time use is required?

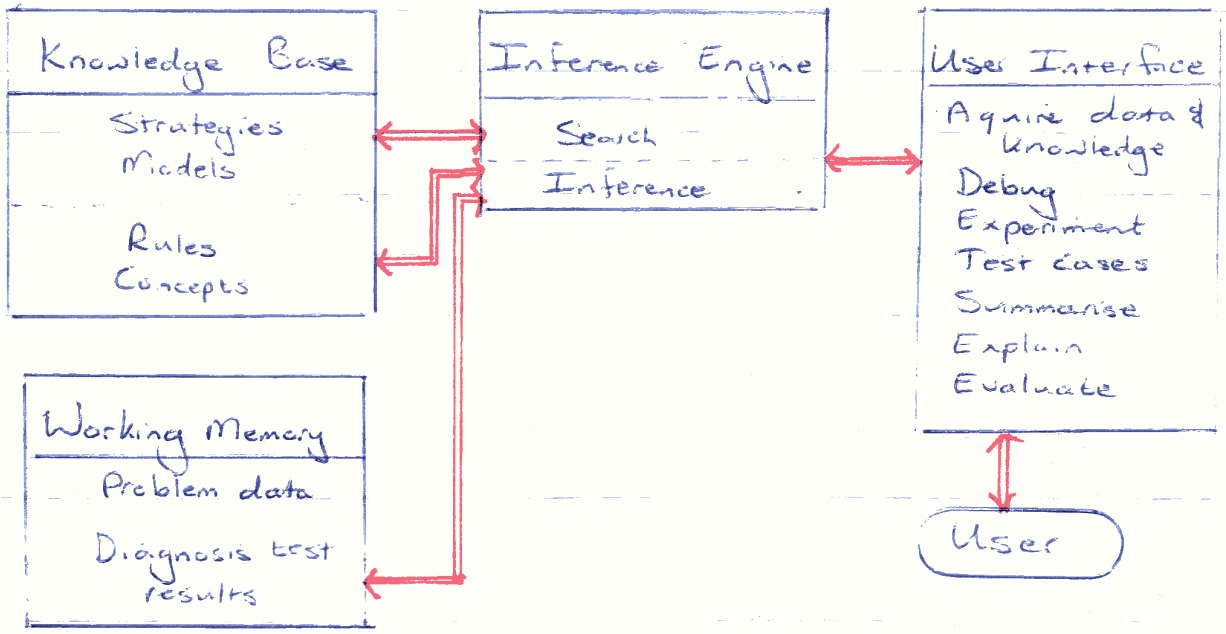
Appendix D - Attributes of a Good Domain Expert

- The expert should be thoroughly familiar with the domain, including
  - task expertise built up over a long period of performance
  - knowledge of the organisations that will be developing and using the ES
  - knowledge of the user community
  - knowledge of technical and technological alternatives
- The expert's knowledge and reputation must be such that if the expert system is able to capture a portion of the expert's expertise, the system's output will have credibility and authority
- The expert will commit a substantial amount of time to the development of the system, including temporary relocation to the development site, if necessary.
- The expert is capable of communicating his/her knowledge, judgement and experience.
- The expert should be cooperative, easy to work with and eager to work on the project
- The expert should be interested in computers, even if not a computer specialist.

APPENDIX E - ISSUES IN ES DEVELOPMENT

- AI is a new and unproved technology, and thus has a unique set of problems in bridging the gap from research institutions to users.
- Since ES development is incremental, there is a need for a development group within the operational environment to maintain and upgrade the system.
- The involvement of at least one domain expert should extend from problem identification to product design and to the release of the product.
- Informal communication at all levels should be encouraged.
- Formal documentation in the interim stages should be avoided.
- The design responsibility should gradually be shifted from the lab to the central technology organization.
- Identify real users who will participate in the project. Development should be user-driven or user-encouraged.
- Plan for the system to be integrated into the existing method of operations.
- Promote an awareness of the technology at all levels with the users from the early stages.
- Do not attempt a cost-benefit analysis until a prototype system is in operation.
- Get an impartial (critical) technical consultant to participate in regular design reviews.
- Avoid unrealistic expectations.

Appendix F - Components of an ES



IE The inference engine contains search guidance and inference components. The search guidance component selects which portion of the knowledge base is most important to try to apply at any point in the problem solving session. It may use general knowledge - base considerations, or it may make use of user-specified strategic rules (meta-knowledge). The inference component evaluates individual rules and interconnections among concepts in the KB in order to add to the WM. The WM is a store of the current problem data. The KB is the main depository for domain-specific heuristics. It is considered to be in four levels, each one built out of elements of the next lower level:

WM

KB

Concepts: - declarative representations of domain objects, with both abstract classes and concrete instances; complex interrelationships can usually be represented and used in making inferences and in constructing similarities. Usually this knowledge can be obtained from textbooks, and includes the basic terms of the

problem domain.

RULES :- Empirical associations linking: causes and effects; evidence and likely conclusions, situations and actions, etc. This level of knowledge is the main form that is obtained from the domain expert, and is based on experience. The knowledge is empirical (difficult to obtain from textbooks), and may have associated with it 'certainty factor' indicating degrees of belief in its applicability. Experts may not agree on knowledge at this level.

MODELS :- Collections of interrelated rules, usually associated with a particular problem hypothesis or overall diagnostic conclusion. Sometimes this represents a subsystem within a complex mechanical or natural structure. Rules within models interact much more strongly with each other than with rules in other models. This level of organisation is often achieved using contexts as an organisational device.

STRATEGIES :- Rules and procedures to aid in the use of the rest of the knowledge base eg guiding search and resolving conflicts when several equally plausible rules apply to a given situation.

Appendix G - Application Areas for ES.

Area	Key Aspects	Example
Diagnosis and repair; complex, data-dependent selections and interpretations	Fixed set of alternatives Question-answer dialogue Differential weighting	Medical diagnosis
Event-driven or data-driven procedures, detailing pre-specified plan	Intricate details Complex, data-dependent ordering	Computer configuration R1
Modelling and simulation of organisations and mechanisms	Declarative knowledge representation & inference techniques	Management systems
Design and planning; generative, goal-directed problem-solving	Open-ended space of alternatives Coordination of multiple experts	Molecular genetics Molgen

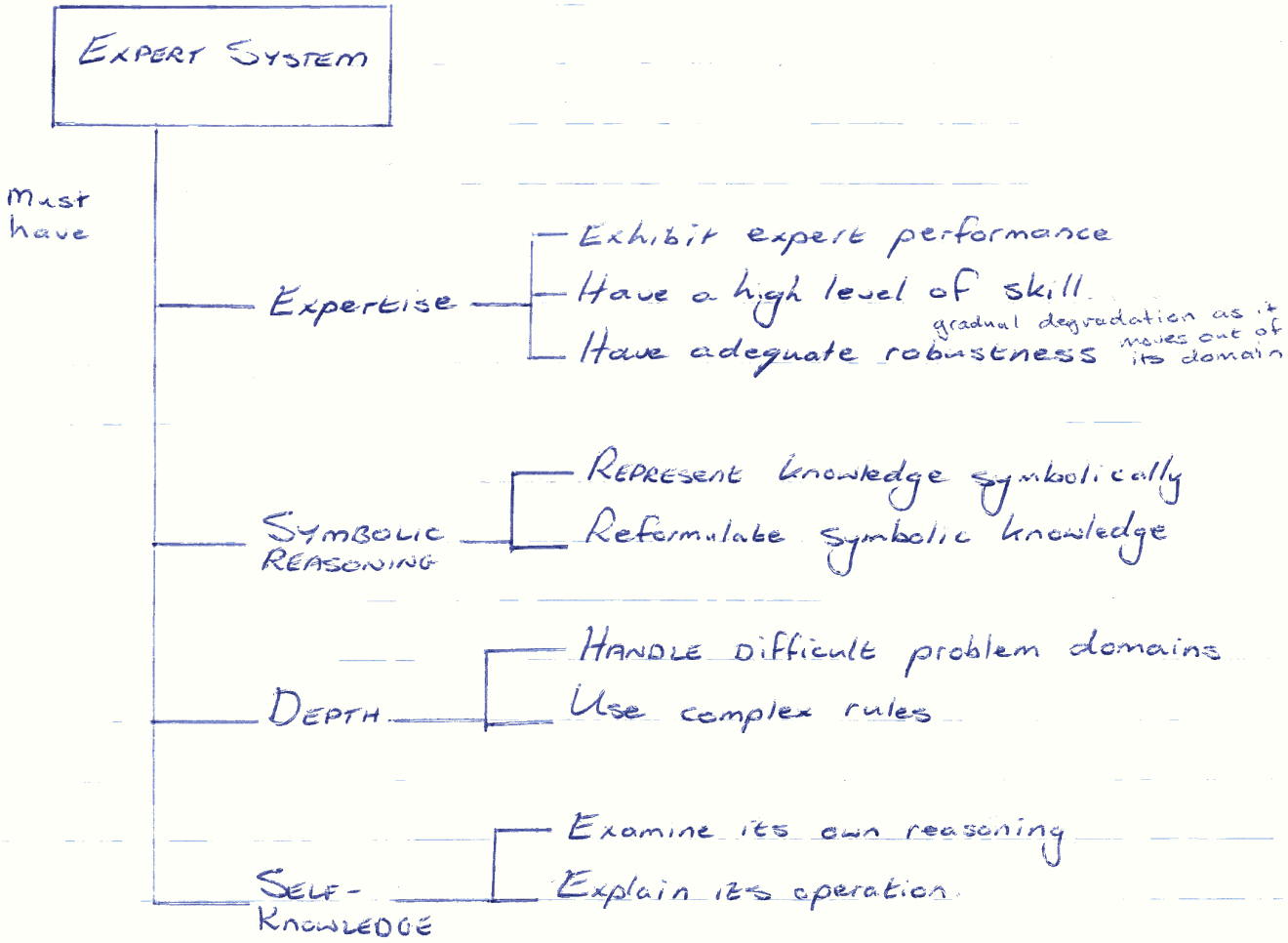
APPENDIX H - EXAMPLES OF EXPERT SYSTEMS

NAME	PURPOSE
MYCIN	Medical diagnosis
PUFF	" "
PIP	" "
CASNET	" "
INTERNIST	" "
SACON	Engineering diagnostics
PROSPECTOR	Geology diagnostics
DENDRAL	Chemistry
SECHS	"
SYNCHEM	"
EL	Circuit analysis
MOLGEN	Genetics
MECHO	Mechanics
PECOS	Programming
R1	Computer configuration
Su/x	Machine acoustics.
VM	Medical measurements
SOPHIE	Electronics tuition
GUIDON	Medical tuition
TEIRESIAS	Knowledge aquisition
EMYCIN	" "
EXPERT	" "
KAS	" "
ROSIE	Building expert systems
AGE	" " "
HEARSAY III	" " "
AL/X	" " "
SAGE	" " "
MICRO-EXPERT	" " "

I Comparison of Data Processing & Knowledge Engineering

Data Processing	Knowledge Engineering
Representation & use of data	Representation & use of knowledge
Algorithmic	Heuristic
Repetitive process	Inferential process
Effective manipulation of large databases	Effective manipulation of large knowledge bases.

Characteristics of an ES that distinguish it from a conventional program.



J ARCHITECTURES FOR KNOWLEDGE-BASED SYSTEMS

Experts are often those who know more facts and heuristics about a domain than lesser problem solvers. The task of building an expert system, therefore, is predominantly one of "teaching" a system enough of these facts and heuristics to enable it to perform competently in a particular problem-solving context.

views

Three basic methodologies - frames, rules and logic - have emerged to support the complex task of storing human knowledge in an expert system. The first is an object-oriented view of knowledge representation, whereby all knowledge is partitioned into discrete structures (frames) having individual properties (slots). Frames can be used to represent broad concepts, classes of objects, or individual instances or components of objects. They are joined together in an inheritance hierarchy that provides for the transmission of common properties among the frames without multiple specification of those properties.

frames

Advantages

Each method provides advantages in certain specific domains (logic where the domain can be readily axiomatised and where complete causal models are available, rules where most of the knowledge can be conveniently expressed as experiential heuristics, and frames where complex structural descriptions are necessary to adequately describe the domain) but the current view is one of synthesis rather than exclusivity. Logic & rule-based systems commonly incorporate frame-like structures to facilitate the representation of large amounts of factual information, and frame-based systems may allow both production rules and predicate calculus statements to be stored within and activated from frames to do inference.

Synthesis

J.1 # FRAME-BASED REPRESENTATION

Frame languages provide an easy means of describing the types of domain objects that the system must model. A frame provides a structured representation of an object or a class of objects. Constructs are available in a frame language for organising frames that represent classes into taxonomies. These constructs allow a knowledge-base designer to describe each class as a specialisation (subclass) of other more generic classes. Some of the advantages of frame languages are:

- they capture the way experts typically think about much of their knowledge
- they provide a concise structural representation of useful relations
- they support a concise definition-by-specialisation technique that is easy for most domain experts to use
- special purpose deduction algorithms have been developed that exploit the structural characteristics of frames to rapidly perform a set of inferences commonly needed in knowledge-system applications (eg constraints, etc).
- the taxonomic relationships among frames enable descriptive information to be shared among multiple frames (via inheritance) and the internal structure of frames enable semantic integrity constraints to be automatically maintained.

Unlike rule-based or logic-based systems, frames are not inherently declarative. However, a frame language can serve as a powerful foundation for a rule language. The frames provide a rich structural language for describing the objects referred to in the rules and a supporting layer of generic deductive capability about those

objects that does not need to be explicitly dealt with in the rules.

7.1.1 Components of a Frame-Based Representation Facility

7.1.1.1 Structural Features

Taxonomy Descriptions

Frames can be organised into taxonomies using two constructs that represent relationships between frames: member links, representing class membership, and subclass links, representing class containment or specialisation. These links provide two standard interpretations of the meaning of is-a links. Frames can incorporate sets of attributes descriptions called slots. A frame representing a class can contain prototype descriptions of members of the class as well as descriptions of the class as a whole. In the KEE system, for example, prototype descriptions are distinguished from other descriptive information by the use of two kinds of slots, own slots and member slots. Own slots can occur in any frame and are used to describe attributes of the object or class represented by the frame. Member slots can occur in frames that represent classes and are used to describe attributes of each member of the class, rather than the class itself. Frames representing classes may have slots whose values specify collections of subclasses that form disjoint decompositions or exhaustive decompositions of the class. The semantics of these decomposition slots is considered to be part of the definition of the frame language. Thus, domain-independent methods can be included in a frame system for reasoning about decompositions.

Attribute Descriptions.

Frames can include partial descriptions of attribute values, and help preserve the semantic integrity of a system's knowledge base by constraining the number and range of allowable attribute values.
Slots in most frame systems can have multiple and a set of properties (facets). Several frame systems have built-in facets for representing constraints on the number of possible values an attribute value can have and for indicating the classes to which each value must belong.

J.1.1.2 Behavioral Properties.

Although frame languages provide no specific facilities for declaratively describing behaviour, they do provide various ways of attaching procedural information expressed in some other language (eg LISP) to frames. This capability enables behavioral models of objects and expertise in an application domain to be built. It also provides a powerful form of object-oriented programming whereby objects represented by frames can respond to messages.

J.1.1.3 Reasoning Services

Frame-based representation extends the system's explicitly held set of beliefs to a larger, virtual set of beliefs by automatically performing a set of inferences as part of its assertion and retrieval operations. These inferences, based on the structural properties of frames and taxonomies,

can often play a major role in the overall reasoning of a knowledge system. Some of these methods perform inheritance; others use constraints such as value-class and cardinality specifications to determine whether a given item could be a value of a given slot.

INHERITANCE

The assertion and retrieval mechanisms for frame-based languages use the member links, sub-class links, and prototype descriptions of class members to augment the descriptive information in a frame. Any frame can have a member link to one or more class frames. A frame inherits the member slots of the class frames to which it has member links. Those inherited slots become own slots of the member frame, since they represent attributes of the member object itself.
Class frames can also have subclass links to one or more other class frames. Since every member of a subclass is also a member of the superclass, a subclass frame inherits the member slots of its superclass frames as additional member slots.

VALUE CLASS & CARDINALITY REASONING

A frame system considers value class and cardinality specifications as constraints on the legal values of a slot. The system provides constraint checking procedures for determining whether a given item is excluded from being a value of a slot. An item is excluded if the slot already has its maximum number of allowable values or if the item is not a member of the slot's value class. These procedures can be called directly by the user.

They are called by the system whenever a slot's values, value-class, or cardinality is changed.
The value-class constraint checking procedures include basic set theory operators and numerical intervals. The primitive test of whether a given item is in a class represented by a frame is performed by sending a message to the frame; thus each class in a KB can have its own membership test. The frame can respond yes, no, unknown. A default method is supplied that looks at explicit membership links and decomposition specifications.

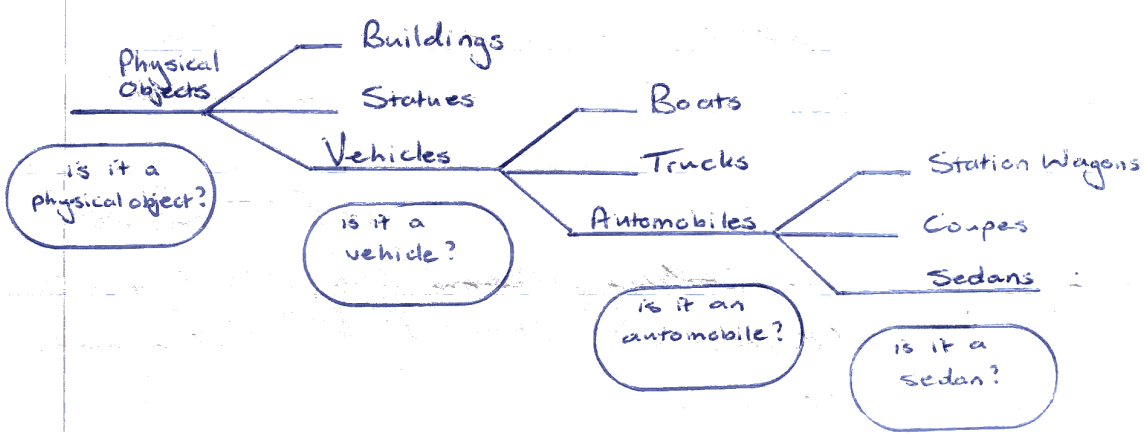
J.1.2 Frames as a Foundation for Production Rule Systems

Rules can be represented as frames; this allows them to be grouped into classes and include attributes.

J.1.3 Using Frames to Manage Rule-Based Reasoning.

The classification of rule frames allows them to be developed and maintained in an organised and modular fashion. Frames can also play an important role in managing a systems rule-based reasoning (see opposite). An effective means of solving diagnostic tasks is to use a class-subclass frame taxonomy containing prototype descriptions of class members as a discrimination net to successively refine the classification of a given object.

CLASSIFICATION BY SUCCESSIVE REFINEMENT



5.2 RULE-BASED SYSTEMS

All rule-based systems share certain properties:

- they incorporate practical human knowledge in conditional if-then rules
- their skill increases at a rate proportional to the enlargement of their KB's
- they can solve a wide range of possibly complex problems by selecting relevant rules and then combining the results in appropriate ways
- they adaptively determine the best sequence of rules to execute
- they explain their conclusions by retracing their actual lines of reasoning and translating the logic of each rule employed into natural language.

Because each rule in an RBS approximates an independent nugget of expertise, existing knowledge can be refined and new knowledge added for incremental increases in system performance, and systems are able to explain their reasoning, making their logic practically transparent and allowing them to satisfy the widely recognized need for understandability in computer systems.

The operating concept of RBS differs radically from von Neumann architectures. Intelligent problem solving with RBSs involves

an iterative cycle of

- identifying from experience the heuristic rules that bear on the problem at hand
- applying one of those rules to solve or simplify the problem.

The technology for building RBSs supports this cycle by providing a dynamic working memory for partial results, a device to identify relevant rules, and selective means for applying desirable rules.

7.2.1 AN OVERVIEW OF RULE-BASED SYSTEMS.

In general, an RBS consists of a knowledge base and an inference engine. The KB contains rules and facts — rules express conditionals, with an antecedent and consequent (action/conclusion) component. The rules always specify the actual behavior of the system when particular problem-solving are entered. In so doing, they perform a variety of distinctive functions:

- they define a parallel decomposition of state-transition behavior, thereby inducing a parallel decomposition of the overall system state that simplifies auditing & explanation. Every result can thus be traced to its antecedent data and intermediate rule-based inferences.
- They can simulate deduction and reasoning by expressing logical relationships (conditionals) and definitional equivalences.
- They can simulate subjective perception by relating signal data to high-level pattern classes.
- They can simulate subjective decision making by using conditional rules to express heuristics.

Several key techniques for organising RBSs have emerged. Rules can be used to express deductive knowledge, such as logical relationships, and thereby to support inference, verification and evaluation tasks. Conversely, they can be used to express goal-oriented knowledge that an RBS can apply in seeking solutions and cite in justifying its own goal-seeking behavior. Finally, rules can be used to express causal relationships, which an RBS can use to answer "what if?" questions or to determine possible causes for specified events.

Facts express assertions about properties, relations, propositions, etc. In contrast to rules, which the RBS interprets as imperatives, facts are usually static and inactive.
The RBS uses a working memory to store problem-solving state information. Ordinarily, the data in working memory adheres to the syntactic conventions of facts. Temporary assertions thus correspond to dynamic facts.

5.2.2 The RBS Niche in Computing

RBSs address a number of shortcomings in conventional programming technology, among them
- the nonperspicuity of programs
- the rapid changes in principles of operation that can occur during development
- the lack of user/expert participation in operations specification
- the lack of experimental development for computer-based competence
- the lack of expertise in exploiting computer capabilities.

Among the features that allow them to do this are:
- modular know-how
- KB's for storing rules & facts that directly determine decisions

- the capacity for incremental development with steady performance improvements
- explanation of results, lines of reasoning, and questions asked
- intelligibly encoded beliefs and problem-solving techniques
- inference chains assembled dynamically by built-in control procedures that can often perform efficient searches.

V.2.3 ## RBS ARCHITECTURE

An RBS is generally a complete computing system, which is to say that it can produce an output by applying memory and processing to an input. An architectural inventory of RBS technology would include the following basic components:

RULES :- data conforming to highly specialised grammars capable of using symbolic expressions to depict conditions & actions

INTERPRETERS :- the rule interpreter matches a rule component to working memory data. Generally, this requires pattern matching to find constants in working memory that match identical constants or unbound variables in rule patterns. The action of the rule is produced by another part of the rule interpreter. Actions are generally changes to working memory or external actions (eg I/O)

TRANSLATIONS :- Nearly all RBSs allow for multiple representation of rules. Typically, all rules are maintained in one prefered representation and translated as needed for other purposes.

EXPLANATIONS :- Explanations can be generated by translating the rules that caused a decision into natural language. This requires that a history of working memory changes and their causes be kept that can be searched as needed for explanations.

Although RBSs have been organised in a variety of ways, they all share a basic configuration - they are sets of decisions about what meanings to give rules, and how and when to interpret them. Two organisations are most common: stimulus-driven or forward-chaining, and goal-directed or backward-chaining. In FC, a rule is triggered when changes in the working memory data produce a situation that matches its antecedent component. In a BC system, the RBS begins with a goal and successively examines any rules with matching consequent components. These candidate rules are examined one at a time. The unmet conditions of the antecedent are extracted from each plausibly applicable rule, and these conditions are in turn defined as new goals. The back-chaining control procedure then shifts attention recursively towards the new goal. The effort terminates when the top goal has been reduced to a set of satisfied subgoals.

Thus RB's make heavy use of pattern-matching between rule-components and working memory. They also quickly identify rules that become relevant as working memory changes. This means that there must be a way to access rules by pattern-matched values. Most RBSs meet this need with software, although some current hardware efforts are attempting to improve performance for these tasks.