Non Monotonic Reasoning

Some Draft notes

D. M. Gabbay

Suppose we are given a database P containing declarative information about some subject matter. For example, P may contain all the rules and requirements for a college degree and all the relevant information about the teachers and students. Another example for P is the rules and regulations of the British Nationality Act. We want to query this database. If A is the query, we symbolically write:

P ? A

to show that we ask A of P.

An example of such a question is:

Can Mr. Smith become a British Citizen?

An expert Human Agent will look at all the rules and data of the database P, including the data relating to Mr. Smith, and then come out with an answer.

The answer is given after some serious deliberation which involves various logical principles of reasoning and search strategies in manipulating the data.

We want to analyse the logical principles involved and find an appropriate logical language in which to express these principles. Furthermore, we want to describe the principles in such a way as to enable us to replace the Human Agent (who answers the questions) by a computer programme (Expert System).

We can thus put our problem as follows:

Find a suitable logical language and construct a suitable logical framework, such that for databases P and queries A, we have:

(*) The machine gives the intuitively correct answer to the query A from the database P, in the sense that an expert human being would have given the same answer!

We are assuming here that there are some underlying general logical principles which are involved in getting an answer to a query from a database, which are independent of the subject matter.

The sense in which we perceive such principles to exist will be discussed later, after we describe in the following sections the types of logical principles involved. Only a closer look at what goes on during the process of evaluating (answering) a query can convince the reader, we believe, that such general principles exist.

Let us in, this section, give an analogy which will explain what we hope to construct.

Consider the main body of Indo-European languages. This body includes many languages with many similarities and differences among them. One connot write a grammar which is good for all of them, simply because they are too different. However, they do have the same <u>type</u> of grammar, as compared, for example, with Semitic languages, which are of a different type. These facts are well known. People express them by saying that it is easier for someone who knows French to study Italian or that it is difficult for Arabic speaking students to understand English grammar.

The Indo-European type of grammar has, e.g., πcasesπ, πverb declentionsπ, πmodalsπ etc. It is possible to construct an <u>abstract grammar</u> which allows in it all the phenomena found in this type of language, including all the types of irregularities and exceptions which exist.

Each particular grammar of a given language, e.g. French or Italian, can be πslottedπ into this abstract grammar.

The different expert systems correspond to the different languages.

The expert systems may reason differently. In fact, different expert systems may use reasoning rules which rely heavily on a particular knowledge representation, which is natural to the specific subject matter at hand. We believe, however, that different expert systems still use similar logical rules, despite a wide range of differences. A top British Petroleum executive may accept the post of general manager of I.T.V. He may have to learn the particular problems of the network etc. etc., but no one will say that he has to learn a new logic

We must clarify our use of the notion of expert system in this report. Th term expert systems does not have a clear cut meaning in the literature. There are various systems doing "expert work" like solving equations or solving specific problems in some particular area. These systems rely heavily on special tricks and shortcuts characteristic to their area of application to such an extent that no logical behaviour is recognisabl. It is not clear whether we should call such a system a system of reason in fact, one may claim that there is no logical reasoning involved at all in expert systems and each system uses its own ways of doing "the work".

Our perception of the notion of expert systems is slightly different. The system must reflect the logic of the human expert in the area of application. Changes in the human expert ways of thinking must be easily reflected in the expert system itself. We give two examples to illsutrate our point of view:

Suppose we want to replace a customs officer by our expert system which can decide whether to question and search an arriving passenger. After much consultation, a set of guidelines, rules and tests is devised which can replace this custom officer (we assume here that the law gives much discretion to the officer in the field). What we get is an expert system in our sense.

A clever observer may notice that no one was ever questioned (possibly because the rules were too restrictive for fear of complaints). One can thus devise the "expert system" which says "let everyone in, freely". This system is probably effective in 99.9% of the cases. It is not an expert system in our sense. It does not reflect the reasoning of the customs officer. A change in the human attitude cannot be reflected in the machine system.

we Seek correct representation in the computer system and not only any shortcut which can give correct results.

Our second example is exactly the opposite. Suppose the customs laws really want to say: search everyone on any charter flight. The airlines may object to a law saying: the above [illegible]
Thus the customs laws will probably be a complex of rules designed to achieve the same effect as the single rule above. A computerised version of the customs rules will not be considered an expert system in our sense (we might call it a rule based system) unless it does somehow "reflect" the true intentions of the law.

4b

If we discover in this case that the tendency to wear jeans and modern hairstyle can be used in 99.9% of the cases to decide whether to search or not (because that is how you can "recognise" a charter flight passenger) then in this case this may be acceptable as an expert system replacing the complex body of rules, since the "logic" is correct.

A change in the human attitude such as "don't search older people on charter flights" can be reflected in the expert system itself like "does the candidate have advanced hotel booking"

4C

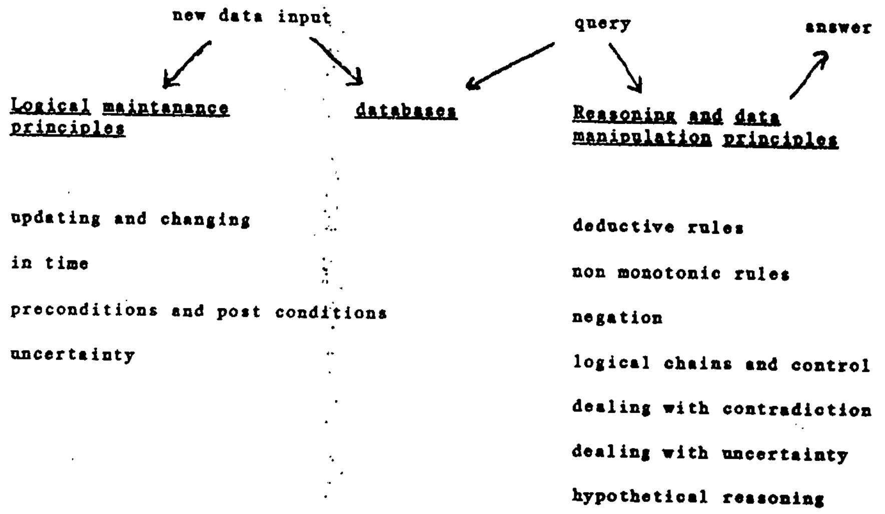
We want to construct an abstract logical framework which will describe all the logical possibilities of inference available to us and into which we can slot any particular expert system. The field of expert systems and non-monotonic reasoning is still in an unsatisfactory state mainly because we have not yet recognised the reasoning principles involved and the ways in which they can be used.

In fact, people are not aware of the strong arguments which show that it is intuitionistic rather than classical logic which underlies human reasoning. Thus all existing systems, we believe, are already at a serious disadvantage right at the start!!

The interest in our proposed research goes beyond computer science. The successful completion of this research will describe and analyse some of the principles involved in Human Reasoning, and thus be of fundumental and lasting value.

The sections below describe the various abstract logical principles and computational techniques involved in arriving at an answer to a query of a database. We discuss each component in a separate section and describe our research objectives at the end of the section.

The following is a schematic diagram showing the inter-relationship of the various components involved in giving answers to queries from databases.



2 Deductive rules Resolution:

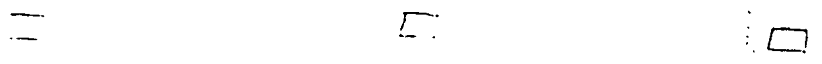
Some questions of a database can be answered immediately by direct search. For example, to the question:

Is London in the UK?

one can answer immediately by searching for the datum:

London is in the UK!

Which may be listed in the database. Other questions may be answered only using deductive reasoning and logic. No pure search of data is sufficient. The answer must be extracted from the database using rules. Here is a well known example (Robert Moore S.R.I.):



Block A	Block B	Block C
Colour green	colour not recorded	colour blue

If we ask: Is there a sequence of the blocks which are both green or both non green? the answer should be yes. However, no search, block by block, comparing colours, will give the answer. Logic is needed here!

Another example involving a well known logical rule is the following:

Suppose the following items are in the database:

(1) The lift can carry only one person (meaning the lift collapses if more than one person is in the lift).

(2, 3) John and Mary entered the lift at 1200 yesterday.

The database yields the conclusion:

(4) The lift collapsed at 1200 yesterday.

Let us take the following symbolic representation:

J = John entered the lift at 1200 yesterday.

M = Mary entered the lift at 1200 yesterday.

C = The lift collapsed at 1200 yesterday.

Then the database can be taken as:

(1*) J and M imply C

or in symbols:

$J \,\&\, M \rightarrow C$

(2*) J

(3*) M

The query is:

(4*) C

or in words:

Did the lift collapse at 1200 yesterday?

Any schoolboy will tell you that the answer to this question is yes. The logical rules involved are simple and well known. In this case we use the rule of Modus Ponens, namely:

(5): If A is known to imply B and A is given to us then deduce B.

In symbols:

(5*) $$\frac{[A \rightarrow B],\ [A]}{[B]}$$

Where the horizontal line "—" means "Deduce".

The problem involved here is not of <u>logic</u> but of practical computation. Given a <u>large</u> database P and a query C, what manipulations shall the computer do to obtain the answer, for the case where the only rules involved are deductive?

The question is similar to the solution of a chess problem, for example, mate in 3 moves. We know what to do from the logical (chess players) point of view. The problem is to write a good program to do it.

Programs for checking whether a query C follows from a database P by deductive rules alone do exist. They are called theorem proving Methods and the best know of these probably resolutio There are books written on the subject. A well known computer language 'Prolog' is based on Resolution Methods. Many logic based Expert systems use resolution methods as their starting point (others use production rules).

[illegible]

Unfortunately, existing Resolution Methods are based on the wrong logic. They give the wrong answers in some cases, in the sense that they do not give the answers a human would give in these cases. Naturally, programmers may be aware of the discrepancies. They try to compensate, thus patching and obscuring the logic and thus errors and their amendments multiply.

Let us see some examples of databases and queries, and see what a human would reply and what resolution would reply. It is easy to point out where the fault of resolution lies.

The problem is that resolution re-writes the database in a convenient form for computation. The re-writing is logically equivalent to the original according to classical predicate logic, but is not the same in the logic of human reasoning, namely, (we believe) in intuitionistic logic. Here is an example:

Consider the database we know already, namely:

(1*) (J & M) -> C

(2*) J

(3*) M

The above is written the way it is expressed in English. Resolution will re-write the above as.

(1*) (not J) or (not M) or C.

(2*) J

(3*) M

So instead of saying:

(1) If John and Mary entered the lift at 1200 yesterday then the lift collapsed at 1200 yesterday.

Resolution will say:

(1#) Either John or Mary was not at the lift at 1200 yesterday or the lift was collapsed at 1200 yesterday.

When we ask a query C from a database, Resolution will re-write the database and C as shown above, and tries to find out whether the answer to

the query ? C is yes by 'cancelling out' pairs of the form: (not x, x). In the example above we have:

(1#) (not J) or (not M) or C

(2#) J

(3#) M

The pairs

(J, not J)

(M, not M)

are cancelled out, and what remains is C. Thus the answer to C is yes, because C was not cancelled out.

So far, we got the correct answer. We got the same answer a human would give to the query C.

Let us now try a similar query and database and go wrong!

Let the database be

(1) J & M -> C

only.

Let the query be

[either (J -> C) or (M -> C)].

In words, the database says that if John and Mary were in the lift at 1200 yesterday then the lift collapsed.

The query essentially asks:

Did one of them collapse the lift on his own i.e.

Either if John (alone) entered the lift at 1200 yesterday then the lift collapsed or if Mary (alone) entered the lift at 1200 yesterday then the lift collapsed.

The above query will get the answer yes, which is wrong. One may need both John and Mary in the lift to have it collapse.

The answer we get in this case is not what a Human will answer.

Obviously we need to devise a new method of manipulating databases. We need methods based on the correct logic. We can devise resolution methods which operate on -> directly without re-writing it in terms of not and or.

Here is another example:

It may be true that

(a) If John loved Jane then John married Jane

and also true that

(b) If Terry loved Judith then Terry married Judith.

It does not follow from (a) and (b) above that:

(c) Either if John loved Jane then Terry married Judith or if Terry loved Judith then John married Jane.

No human will accept the above as a valid conclusion. Resolution however, would say yes. (c) does follow from (a) and (b).

A perhaps more striking example is:

(a) If John is in Paris then he is in France.

(b) If John is in London then he is in England.

(c) Either if John is in Paris then he is in England or if John is in London then he is in France.

3 Non monotonic rules:

Consider the following line of reasoning:

Suppose our data is a table of charter flights to New York of the major airlines. It has the form:

AIRLINES	Telephone numbers of main booking offices	List of charter flights:

Suppose we can "see" from the table that there are no charter flights to New York on Mondays. We mean here that we "see" that there are no flights at all, not that the table contains no flights (which is obvious). This is a use of a Default Rule. We reason that had there been such a flight, it would have been put in the table!!

Of course the table may be incomplete, but we also assume, knowing how anxious airlines are in advertising their flights, that the table is complete and there are no flights at all on Monday.

Let us now see how a deductive chain of reasoning may be constructed. Suppose the law says that a businessman may deduct as expenses the cost of a flight only if he has flown British Airways. If, however, no direct British Airways flight is available on the same day, then any other airline ticket may also be deducted.

Thus the database will contain the rule:

(1) If a ticket is British Airways then the amount is tax deductible.

(2) If British Airways has no flight to the same destination on the date of ticket then the amount of the ticket is tax deductible.

Let our database contain the above two rules and the table of charterflights. If one queries the database about whether a Monday charter flight ticket with TWA to New York is deductible, the answer should be yes. First we search the table and find no British Airways charter flight to N.Y. on Mondays. We use the Default Rule to deduce that no such flight exists. Then we use the result together with rule (2) above and give the answer yes; the cost of the ticket is indeed tax deductible.

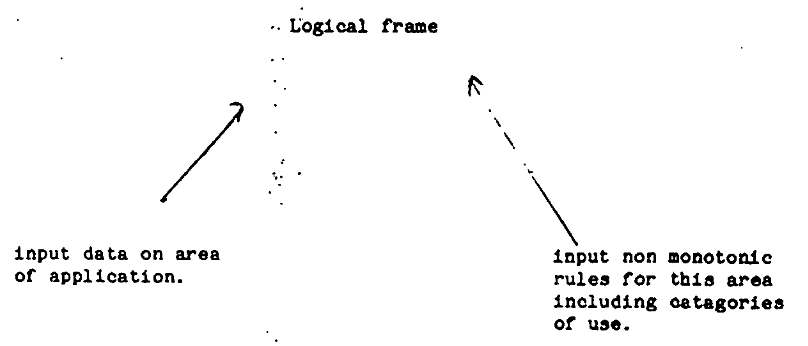
Notice that the Default Rule is a non-monotonic rule which depends on the current state of the database. It is possible that the data is wrong and a correction may come from British Airways that indeed there is a charter-flight to New York on Mondays. In this case the answer will change to no, the amount is not tax deductible. So in a way this rule (Default Rule) is a Heuristic rule, based on the assumption that there is no error of omission in the table.

The following is another example of possible reasoning from a table. Suppose the airline's telephone number is listed as 432111. This is only a 6 digit number, and it is obvious that a digit is missing. A human would reason that the number is probably 4321111, assuming that airlines always have easy to remember telephone numbers and that the digit "1" is likely to be missed. There is only one more possibility and that is that the number is 5432111.

The above use of reasoning is very instructive, in that it applies only to this particular database, namely, telephone numbers in London. We see that non-monotonic rules are characteristic to the database and the environment in which they apply whereas the deductive component is comprised of general logical rules applicable to any database.

What happens to our claim that there are general rules of reasoning for expert systems?

The answer is that these non-monotonic rules interact among themselves and interact with other components of the system in certain logical ways. These ways are general and independent of the environment. Thus if we code the ways in which rules chain, interact and are updated etc., as we intend to do in this research, then we will have an abstract logical framework which will take data and the particular non-monotonic rules of the environment as further data, as in the schematic diagram below:


Logical frame
input data on area of application.
input non monotonic rules for this area including catagories of use.

The advantage of this framework is that the rules can also be changed just like the data. There could be "grey areas" where depending how "strict" we are, the answer to a query could be "yes" or "no". The degree of "strictness" can be put in and the system will know how to respond.

We meet here two kinds of rules. Logical deductive rules and non-monotonic, Heuristic rules. Deductive rules give answers which never change in the face of an inflow of additional positive information. If non-monotonic rules are used to give an answer, then more detailed information may invalidate the answer. The additional information we are talking about is of the form of more compatible details. We are not dealing here with new information which replaces current data with new (possibly contradictory) data. We shall see that there is still a third kind of rule, discussed in section 5.

4 How to use negation:

We are dealing here with a logically difficult concept. When a human is asked a question ? A and answers no to the question, there are complex logical principles involved in arriving at the answer. The answer no does not mean the same in different contexts. There are several different negations involved and the final answer no is a result of some complex combinations of these negations. We shall indicate how we propose to analyse negation in this research, by listing the main types of negation.
The use of negation in expert systems is further complicated by the fact that a full logically correct treatment of negation causes a combinational explosion. Thus one cannot hope to work with negation on a computer witnout taking shortcuts, and it is our job to make sure that these shortcuts are logically sound. This is the first time (in the course of the description of this research so far) that we have to make "logical - concessions" owing to practical computational limitations. The challange is, of course, to improve performance while still retaining logical control. This is an area where there is the greatest temptation for "hacking" and act hoc heuristics, and where there is the greatest danger of logical errors.
We start by arousing the reader's suspicions of the notion of negation. Suppose we ask the Council whether we can get permission to do some alterations in our house roof. Suppose the answer we get is just the word no. This will never satisfy us. We would want to know why not (i.e. by what process was the answer no arrived at), and even further, we may want to consult other authorities.
The reader may think that there is a body of specialised knowledge involved here and by asking for details on "why not", we are trying to learn more. This is not the main reason. We claim that what we want to know is "what kind of no is involved". Let us take a simpler example. Suppose we ask a travel agent whether there is a charter flight to Timbuktu on Monday

January 1st? The agent says no. A rational being will ask further how come not?, and if one is clever, one would ask another travel agent. No one will be surprised if the other travel agent does find a flight to Timbuktu.
We have the following logical types of negation.

A: Negation as default

This type of negation follows the understanding that if the information is not listed positively then it is negative. Examples of this are abundant. Any list of the winners implies that the names not on the list are the losers. A contract specifying the allowed uses of a hired computer implies that any use not specifically mentioned is forbidden.

B: Negation as inconsistency:

We also answer no to questions because if we say yes then we get undesirable results. This is a dynamic sort of negation.
Taking up the example of the hired computer, one may have a general clause in the contract, allowing the use of the computer in any way desired, as long as the computer is not physically damaged. In this case many possible uses may be forbidden if they are thought to lead to physical damage. This use of negation assumes that there are things we do not want to have true and we say no to anything which may make them true.
Another example of this sort of negation is probably any general guidelines such as environment conservation laws. They do not specify what is allowed or forbidden to do but any single proposal is checked for its potential for environmental damage.
This negation is rather complicated logically and needs to be carefully studied. The problem is that what we do not want depends on what we already have. In other words, if P is the current database, (describing what is true now, before the next new information comes in) and if N is the set of data, which we do not want to become true, then N depends on P, and thus what we may negate now, we may not negate tomorrow.

For example, we may not want to work on Sundays, and thus will reject any contract which may include work on Sundays, but if our expenses increase substantially then we may be willing to work on Sundays.

C: Strong negation

Sometimes we have that some data is specifically negated. For example, we find written on medical bottles, "not for internal use" or posters like "Do not step on the grass" etc. This we call strong negation, because it does not depend on any additional assumptions or information and does not change in time.

D: Informative negation

This type of negation is very common. It is not really negation but a way of giving additional information, which is contrary to what one would expect. It thus depends on and assumes a context. When we say "the train did not leave on time", we are adding a positive piece of information, that the train was late. Had we not volunteered this information, the hearer would have assumed that the train had left on time. Thus we say not A to convey the information that contrary to what one may expect (namely that A is true) we have that A is not true.

E: Negation as failure

The notion of negation as failure is more a combinatorial computational notion than a logical one. It resembles default negation in the sense that we negate A if we fail to affirm A. The difference between the two is whether we perceive the notion of failure logically or computationally. We illustrate the difference through an example.

Suppose we go to a chemist and ask for some medicine. The clerk would go and look for it. If he cannot find it, he will say that he does not have it. We may accept this no at face value as negation by default. If one cannot find it then one does not have it. Suppose now that it is obvious to us that this particular clerk is new to the shop and is furthermore clearly incompetent. We have the feeling that the medicine is there and

the clerk simply does not know where to look. The no in this case is negation by failure - the failure of the search to produce an answer yes. It is a computational no.

In general when we have an expert system searching for an answer and it fails, it is computationally very practical and convenient to say no. This is the use of negation as failure. This use depends on the particular search methods. It may not be logically sound. Rephrasing our query a bit differently, may result in the success of the computation, because it may go along a different path. Thus the same question may get "yes" or "no" as answers, depending on the manner of asking.

Negation as failure has a serious conceptual disadvantage. We are saying no because we fail to say yes. So our no does not follow from some constructive (or destructive) active knowledge, but from lack of knowledge.

The other negations negate from some sort of knowledge. Even negation by default, uses the positive assumption that the database was carefully organised, and so if a datum is not there, then it must be negated.

Certainly strong negation, or negation as inconsistency, involve some positive action on our part, in the process of negating.

The lack of positive "action" in negation as failure surfaces when negation interacts with the quantifiers. Suppose we ask:

Is there someone not allowed to enter the Science Museum?

In symbols:

? $\exists x$ not $A(x)$

[$\exists x$ means there exists an x such that]

The problem is that we want to know not only that there is someone but also we want to get a name of such a person. We would like a more specific answer like John, e.g.

not A(John)

negation as failure cannot supply such a name, while other negations may be able to do that, for example, negation as inconsistency is a positive notion, it says no because it can "show" we get things we don't want to get. The process of "showing" will yield some names.

We saw that there are several notions of negation used in a complex interwoven way when dealing with data. The logic involved is yet to be analysed. Negation plays a central role in any expert system.
Its use is further complicated by the fact that computationally it causes a combinatorial explosion.

We hope to find a logically sound and correct notion of negation for use in expert systems. We will also try to find logically correct combinatorial shortcuts for practical implementation.

The logical analysis of negation, if possible, is of immense theoretical and practical value in many fields, not only computer science. It is inherently computationally complex. It causes combinatorial explosion owing to its logical nature and not because of poor implementation. Psychologists testing human reaction time to logical queries found that the presence of negation causes complexity to go up one level higher.

5 Logical chains and control structures in Logic:

In the previous three sections we have met three types of logical entities involved in the process of reasoning from a database. These were:

(a) The deductive rules, the underlying logic.

(b) The non-monotonic rules, which can be regarded as additional Heuristic rules, characteristic to the specific subject matter.

(c) The use of negative information.

The above indicated and described only the types of rules involved. If we want a working logical framework which can successfully extract information from a database, we must indicate how these rules are used successively, i.e. how to form chain deductions. We need to specify how to obtain more information from the database using some rules and then continue and use the other rules and the extra information just obtained to answer our query. This chaining is common to any process of reasoning.

So imagine a system with a database $\underline{P}$ and one step rules R1, R2, R3, ... of different types which can be used to extract more information from the database. If there are no restrictions and no special controls we can extract information in the following typical way:

$$\begin{tikzcd} \text{data } P_1 \arrow[r] & \begin{matrix}D1\\ D2\\ D3\\ \vdots\end{matrix} \arrow[r] & \begin{matrix}E1\\ E2\\ E3\\ \vdots\end{matrix} \arrow[r] & \cdots\cdots \end{tikzcd}$$

additional data obtained by applying any one step rule on P_1. Call the enlarged data P_2

additional data obtained by applying any one step rule on P_2. Call the enlarged data P_3

In practice the model above is too simple. What is not immediately apparent is the amount of control and common sense we exercise when we use chain reasoning in everyday life. This control must be logically analysed and the principles incorporated into the system.

There are several types of controls on the successive use of the rules. Some of them are of logical nature and some are specific to the subject matter. Here are some typical simple control restrictions:

We may restrict the use of a rule R1 only to certain situations. In the Nationality Act, for example, the rules of passing on citizenship to one's children depend on which rule was used in acquiring such citizenship. So we may use rule R1 to show that Mary's father is a British citizen. Rule R1 may say how to naturalise and we use it to show that Mary's father has naturalised. Rule R2 may say that Mary herself can become a British citizen if her father is a British citizen, provided the father's citizenship is not because of rule R1.

In other words Mary can become a citizen only if her father was a British citizen through birth in UK, but not through naturalization. (See {Kr84})

The above is a "logical" control, where the "logic" is that of the Nationality Act. We may give controls having to do with efficiency, e.g. when either rule R1 or R2 can be applied, we apply R1 first, because whatever we may get using R1 will make things more efficient for R2.

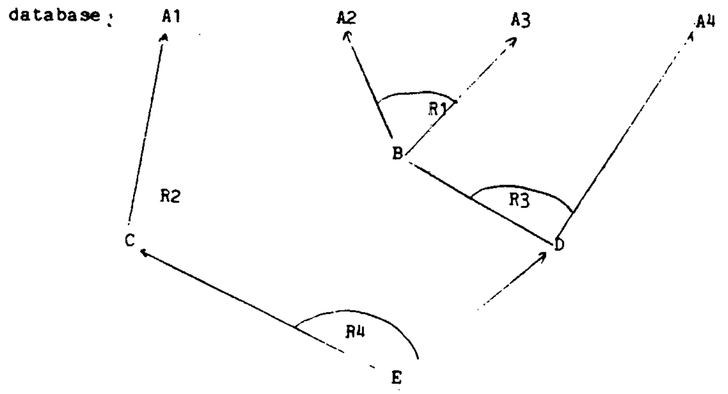
Some restrictions on the use of the rules are given by the logical necessity of trying to make the rules mean logically what they are supposed to mean. This is especially true with negation rules. Negation rules are always restricted and tightly controlled, because of the complex nature of negation.

It is not a matter of efficiency, but a matter of making the system work correctly. A typical restriction is the order in which the rules may be applied, and priorities among them.

Another example of Heuristic control may be that, when searching, for example, for a good cheap flight to the east coast of the USA, it is more efficient to look at major cities first and then airlines and flights rather than airlines first and only then look at which cities they fly to. Some of the above types of restrictions on the rules do not present any logical problems, beyond possibly the choice of a good framework in which these restrictions can be expressed. There are however more subtle problems which require further logical analysis.

The way the rules for negation interact with the other rules is mainly a problem of logic, but as we have seen in the last section, we may have to approximate the rules for negation, in order to avoid combinatorial explosion. This must be done correctly. Furthermore, the use of non monotonic rules together with negation is a major problem in any expert system. The way in which the problem presents itself is in the question of how to modify our thinking, when a conclusion A which follows from a database is found in reality to be false. Obviously something has gone wrong. But what do we exactly learn from this additional information? (A is false.

To be more specific, suppose we have the following sequence of deduction.



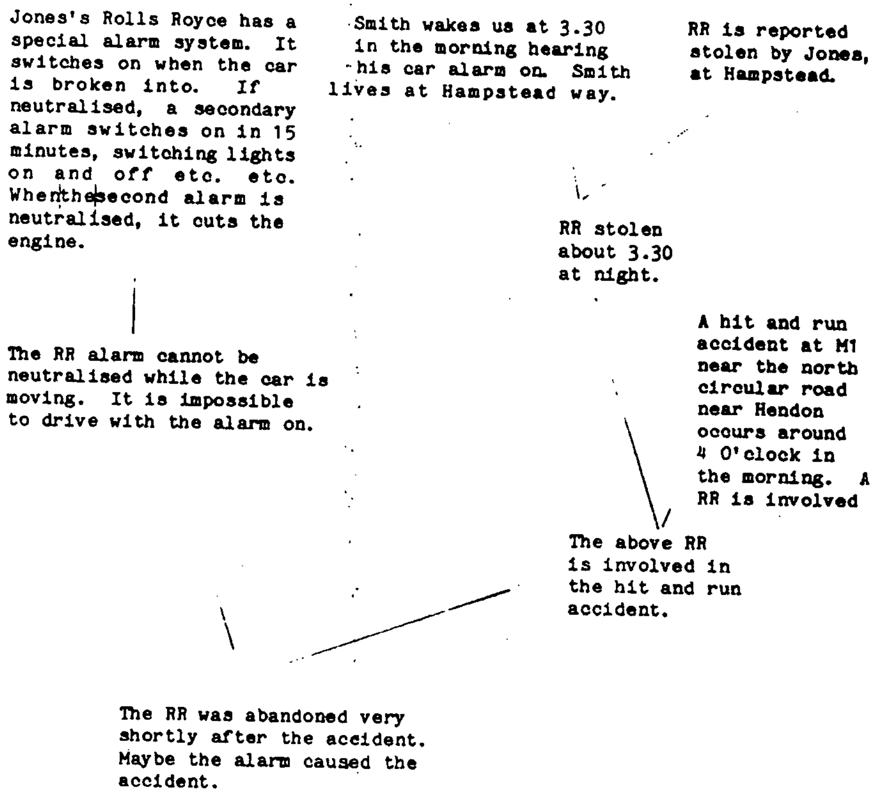
(1) E is arrived at by applying rule R4 to C and D.

(2) C was obtained from A1 of the database using R2.

(3) D was obtained from B and A4 using R3.

(4) A4 is in the database but B is not in the database. B was obtained from A2 and A3 using rule R1.

This is a typical situation. Suppose we check and find that E is false. How do we modify our deductions in view of this additional information? Consider the story of the yellow Roll's Royce (RR), which gives rise to the reasoning structure of the diagram.



Jones's Rolls Royce has a special alarm system. It switches on when the car is broken into. If neutralised, a secondary alarm switches on in 15 minutes, switching lights on and off etc. etc. When the second alarm is neutralised, it cuts the engine.

Smith wakes us at 3.30 in the morning hearing his car alarm on. Smith lives at Hampstead way.

RR is reported stolen by Jones, at Hampstead.

RR stolen about 3.30 at night.

The RR alarm cannot be neutralised while the car is moving. It is impossible to drive with the alarm on.

A hit and run accident at M1 near the north circular road near Hendon occurs around 4 O'clock in the morning. A RR is involved

The above RR is involved in the hit and run accident.

The RR was abandoned very shortly after the accident. Maybe the alarm caused the accident.

Notice that the only "observables" in this deduction is the database itself and the fact whether the abandoned RR is found or not.

<u>Fact A5</u>: No RR is found.

Our problem is now to try and rethink and examine our deduction, and more importantly, try to modify and improve our reasoning. At first glance we may think that this is a very specialised example, and that no general principles of reasoning can be involved. We may believe that each practical example has its own special characteristics and "logic". However, this is not the case. The police in this example will take some action which is typical to any reasoning situation.

(a) They will make sure the search for RR is exhaustive. In logical terms, this means that they verify that the negation (no RR is found) is negation of default and not negation by failure.

(b) They will re-check the weakest assumptions in the chain of reasoning. In this case they'll probably check how does Smith know that it was 3 O'clock? Was his alarm clock correct? How does he know it was his car he heard? Maybe it was the RR's first alarm.

(c) Maybe Jones knew about Smith's hearing an alarm. Maybe Jones got into his RR and triggered his alarm by mistake. He saw Smith come to check his car but kept away. Jones himself was involved in the hit and run accident and this is his way of getting away with it, by reporting his car stolen, forgetting, or not understanding, the nature of his secondary alarm.

(d) Is there another story in the database about Jones which aroused our suspicions?

(e) We make a note about Jones' credibility, for future reference!

There seems to be some general principles involved:

(1) With each item A in the database, we associate a "checking" rule which says:

When you doubt the truth of A, recheck items $B_1 \dots B_k$ to confirm A.

(2) Each rule R giving x from $y_1 \dots y_k$, is accompanied by an inverse rule of doubt, which says that if x is not observed, then this casts doubt about the rule R and about y1 ... yn, and some other data. Most importantly, this rule of doubt will recommend that we

try to observe other related facts and the rule will tell us how to act on the results.

(3) The system must remember for future use that R is a doubtful rule, and perhaps use a measure of "uncertainty" on rules and data.

The reader may have noticed that the above principles, (1) - (3) are different from the reasoning principles we have discussed so far. The difference is that they involve <u>Actions</u>! Take rule (2) for example. We start with a "static" database, containing a certain amount of data. On the basis of these data we reason that conclusion C follows. So far everyting is static, no action is taken. C however, can be observed, and tested, and so we go aghead and check C. We find that C is not true. We add <u>not</u> C to the database. We can now apply the non monotonic reasoning rules to the new database and get new conclusions. The situation could be no different from adding any new data to the database and applying the rules. But here we do more, we ask for more data, we ask to observe related facts and act upon them (maybe even modifying our rules).

There is no reason why we should allow <u>Actions</u> in rules only when things seem wrong. We can incorporate actions into rules right from the start, and have rules like:

<u>Car won't start</u>

check headlights

check electrical system

or more direct actions like:

<u>The carburetor is flooded</u>

Depress accelerator to floor while starting.

Rules like the above make the database and the reasoning process more active, more like a Human Agent. We thus modify our perception of non-monotonic rules to be as follows:

On the basis of A1, ... , An, check the additional data B1, ... , Bk and on the basis of what is found, case by case, deduce the following C1, ... , Cm

Here is an example:

On the basis of A check B1, B2. If B1 is true and B2 false deduce C1. If B1 is true and B2 is also true, deduce C2. Otherwise this rule is not applicable.

The rule may have only the following form:

If A check B and add the result to the database.

The above makes the inference machine interactive with the user. It behaves more like a Human Agent in the sense that it may answer a question with a question.

The example below, in figures 1 and 2, shows a practical case where "observable" items occur and where some items are "associated" with others, though with a certain degree of uncertainty.

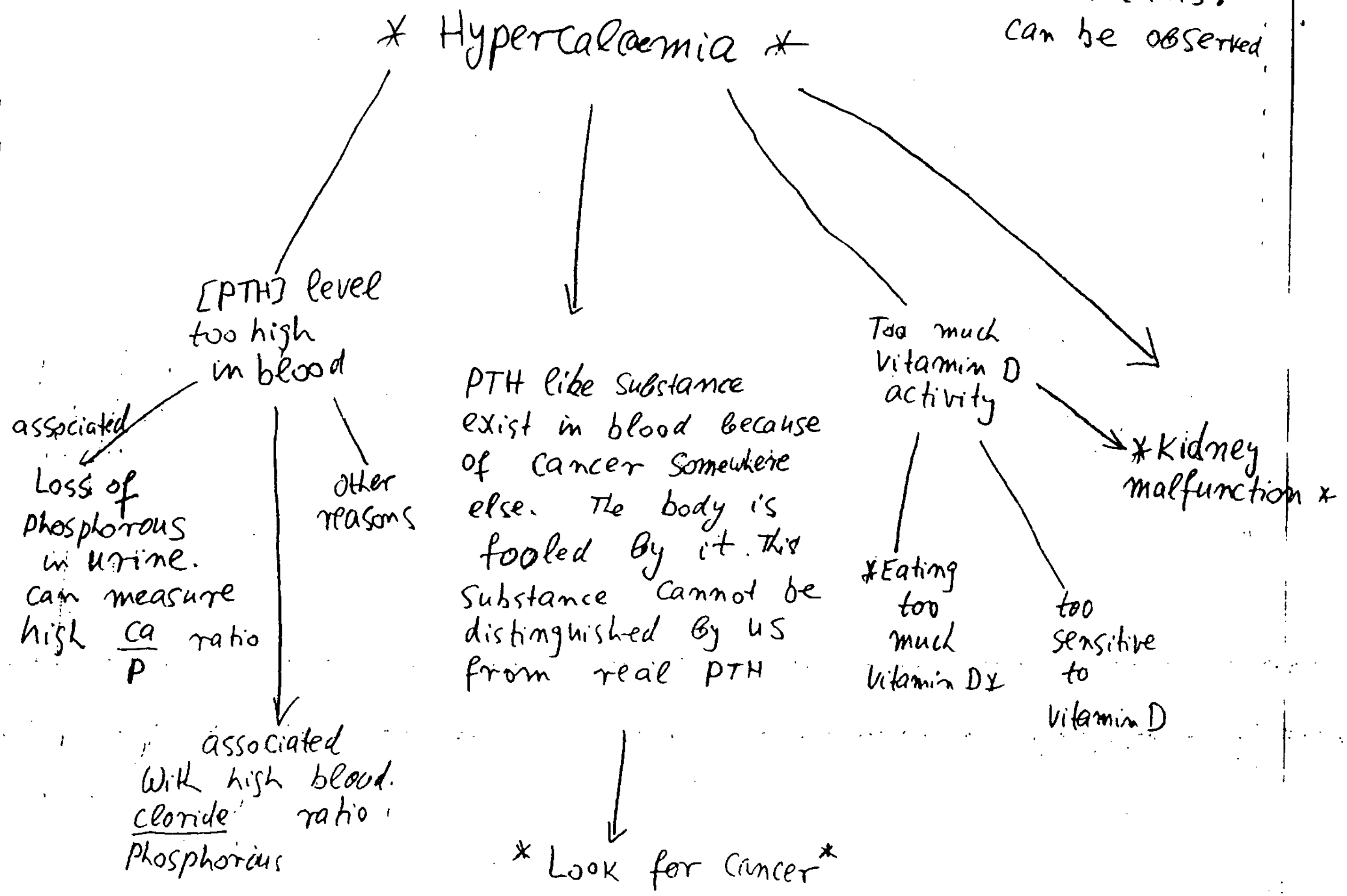
The analysis is courtesy of Dr. T. Cory and Dr. (med) N. Bradford.

notice that once figure 1 is given, it is almost automatic to build figure 2 from it. From the logical point of view, we claim, it may be possible to construct an expert system which can build automatically expert system based on objected oriented descriptions like figure 1. Furthermore the resulting expert system (i.e. figure 2) can be made self improving

FIGURE 2

"Associated" entails uncertainty.

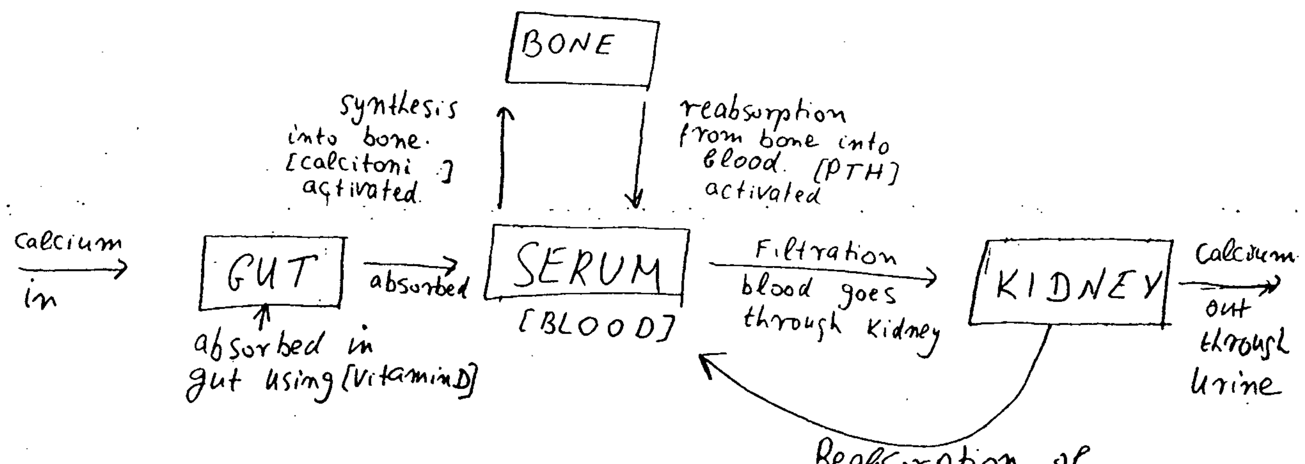
* means: can be observed



* Hypercalcaemia *
[PTH] level too high in blood
associated
Loss of phosphorous in urine. can measure high Ca/P ratio
associated with high blood cloride/Phosphorous ratio
other reasons
PTH like substance exist in blood because of cancer somewhere else. The body is fooled by it. This substance cannot be distinguished by us from real PTH
* Look for Cancer *
Too much Vitamin D activity
Eating too much Vitamin D
too sensitive to Vitamin D
*Kidney malfunction *

Courtesy of Drs. T. Cox / N. Bradford

FIGURE 1



BONE
synthesis into bone. [calcitoni] activated.
reabsorption from bone into blood. [PTH] activated
calcium in
GUT
absorbed in gut using [VitaminD]
absorbed
SERUM [BLOOD]
Filtration blood goes through kidney
KIDNEY
Calcium out through urine
Reabsorption of Calcium. [PTH] activated.

Healthy body allows for a narrow range of calcium in blood.
"Hypercalcaemia" means too much calcium in blood.

Courtesy of Drs. T. Cox / N. Bradford

Methods of chaining of rules and for rule control are in the core of any expert system.
Some rules and chaining of rules have to do with the specific subject matter, others deal with negation and non-monotonic updating.

We hope to find the logical features involved in chaining and control and build the correct natural framework for dealing with them. We will especially pay attention to how the framework reacts to uncertainty (section 9), as this is a major problem in expert systems.

Chaining and control rules can be very specific to the subject matter of the database. It is in this section, that our hypothesis of the existence of a background logic for all reasoning from expert systems will be tested. We believe that although the chaining and control rules may be very different in different systems, the updating and rethinking principles are very similar. Furthermore, there may be common principles having to do with handling uncertainty.
We also believe that we will find at least several large logical groupings of types of expert system, even if we will not be able to find a common denominator for all expert systems.
Present day expert systems seem to be successful without analysing the process of chain reasoning (beyond single step rules) at all. They rely on their ability to handle sheer bulk of information. In fact, some (e.g. Professor Feigenbaum) hold the view that chain reasoning is not of main importance. Their argument stresses the meagre results one gets from attempted Resolution reasoning in Artificial Intelligence. We have already discussed these methods and their limitation, and we propose to remedy that. We are looking for a framework which, if successful, will push expert systems to the next step up the ladder of evolution.

7: Hypothetical and counterfactual Reasoning

We explain our problem via an example. Suppose the government is interested in raising the educational level in elementary schools in socially deprived areas. The difficulty the government faces is that good school teachers are reluctant to teach in such schools. To counter that, the government offers some inducements and fringe benefits.
These include:
(a) Teachers in such schools do not have to have a full university degree to receive the full salary and benefits of degree holders. They should be good teachers with good experience. For example, missionary nuns with some experience will be considered as qualified teachers with an M.A. degree.
(b) Teachers with an M.A. degree who teach in such schools, will have their seniority increased by 10 years, and provided they teach a minimum of two years in such a school, they can also count this seniority towards their retirement. Thus a qualified teacher with a degree of M.A. and seniority 18 years can teach 2 years in such a school and retire with seniority and pension of 30 years.
These benefits were introduced to attract good teachers to such schools.
The above rules are reasonable in any environment with serious social problems. The rules are counterfactual. They say that under certain conditions, e.g. that Miss Smith teaches in a deprived children's school, the computer computes certain predicates such as the salary seniority of Miss Smith, not according to the real data in the database, but by pretending that the data were different.
Laws of this form occur in many other areas, for example in the British nationality Act. To decide if a child born in the UK is a British citizen, we look at the status of the father. If the father is dead at the time of birth, but had he been alive he could have been naturalised, then the law says that the child is a citizen.

Counterfactual Reasoning takes the following form:

<u>The database P</u>	<u>new datum</u>:	<u>query</u>:
datum B		
	datum C (to pretend to replace A)	G
datum A		

In the presence of B in the database, pretend A is not in the database, pretend C is in the database and then compute the answer to query G.
The problem we are faced with is that of logical control. If we pretend that A is not in the database, and we add C to the database we may have a contradiction. Yet, we intend to compute the goal G without being worried, and apparently we intuitively know how to steer away from trouble. To take the British Nationality Act as an example, to check whether the child is a citizen today, we check the status of the father. If the father died 3 years ago, we pretend the father has not died and check whether he would have been able to naturalise today. If the answer is yes we say that the child is a citizen. If not, we say the child is not a citizen.
It may be the case that in order for a person to naturalise, the person must have filed all his yearly tax reports. The father, being dead, filed no reports. Clearly the computer is not supposed to say, sorry, the father couldn't have naturalised because there is no record of any income tax reports. A human would know intuitively what is relevant and what is not. We see that there is a lot more to this counterfactual "pretending" than the simple addition and deletion of information, and our task is to figure out its logical properties!
To give a more extreme example, consider the case of Miss Smith, the nun. She has no degree, but does have some teaching experience. She is accepted as a teacher in this special school. She is 43 years old. The computer,

following the rules laid down before, considered Miss Smith as a qualified teacher with an M.A. degree, aged 43. Miss Smith taught for 2 years and expressed her wish to retire. The computer, when seeing the data of a 43 year old teacher with an M.A. (that is what the computer is supposed to see!) used the second rule, added 10 years to Miss Smith's seniority, and replied that indeed her age is 55 and she could retire!

8 Dealing with contradictory information, consistency and integrity of databases:

The Americans are very much worried these days about computer information and technology falling into the hands of the Soviets. As part of their security measures, they have instructed the C.I.A to keep detailed files on any computer scientist making contact with major British or American companies. For individuals in Britain, both the C.I.A and F.B.I co-operate in their files and databases concerning these individuals.

In the case of Professor X, an unfortunate discrepency has occurred. The F.B.I had his address as Stanford University, California, and the C.I.A had his address as Imperial College, London. When the two databases were joined together, the union contained contradictory information. The computer, using classical logic resolution (the same inadequate tool of reasoning discussed in section 2), inferred that Professor X was none other than Public Enemy No 1, because in classical logic a contradiction implies the truth of any statement.

There are several points of principle involved here. First, classical logic does not deal with contradictions correctly. Although it is logically correct to infer from the above contradiction that Professor X is Public Enemy No 1, it is clearly an incorrect step in terms of human reasoning. More importantly, Professor X's address is completely irrelevant to the question of whether he is Public Enemy No 1. or not.

Intuitionistic logic, the logic we propose to use in the solution of the problems mentioned in the previous sections, is equally useless for our problems in this section. Intuitionistic logic would make the same inference as classical logic and also conclude that Professor X is Public Enemy No 1.

A second point of principle involved is simply the question of what to do with the contradiction itself. What do we do when we have two contradictory items of information in the database. Do we choose one of

them? How do we make the choice? Do we leave them in and find a way "around" them?

A third point is more practical. The C.I.A agent may investigate Professor X and find the charge the computer made to be ridiculous. They may suspect that there is a contradiction in the database but how to find it? Generally the contradiction may involve several steps of reasoning and may not be as blatant as in the case of Professor X. We may have several simple and innocent looking data items and some very reasonable rules which together give the wrong answers, but no single item is to blame. How do we debug our system in this case?

The problem of dealing with contradictions is a difficult one. We believe a solution for better handling of contradictions can be found by looking closely at the ways humans deal with them. Upon reflection we arrived at the following principles to be taken as a first attempt at a solution.

(1) We do not share the view that contradictions in a database are a "bad" thing, and should be avoided at all costs. Contradictory information seems to be part of our lives, and sometimes we even prefer ambiguities and irreconcilable views. We must therefore seek logical principles which will allow for contradictions in the same way that we humans allow for them, and possibly even make contradictions useful.

(2) Humans seem to intuitively grasp that some information is more relevant than other information. The notion of relevance should be developed and used.

(3) There seems to be a hierarchy of rules involved here. Rules of the form "When contradictory information is received about A - do B" seemed to be used constantly. These are meta-rules, i.e. rules about rules. Full exploitation of these rules require our database language to be able to talk about itself. This requirement we have met also in previous sections, and we shall develop such a language.

(4) Never throw anything out of the database. Always keep an open mind, that although A is rejected now (because of new information contradicting it), it may become useful again. Perhaps some uncertainty values may be attached to all data items. We will have to check this point of view.

(5) Despite everything we do, although we may give various heuristic rules dealing with contradictory information, there will always remain the possibility of two contradictory items of equal reliability and equal relevance and equal standing concerning which there will be no way of deciding which of the two items is the correct one. In this case, we can only wait and be suspicious of any computation involving them. The problem of contradiction is further discussed in part 3 of appendix 2 of this report.

We described the problems associated with contradictory information in a database. By contradictory we mean not anly logically contradictory but that the database does not satisfy some conditions it is supposed to satisfy. The main problem is how to trace and eliminate the causes of contradictions. Our view is different from the current view that contradictions are "bad". We think that they are an essential part of life and perhaps can even be made useful.

A system for maintaining consistency of database based mainly on the notions of relevence, and hierarchy of rules, allowing for cotradictions to be present in the system and enabling the user to make use of contradictions.

Contradictions are a "useful thing " to have!

9 Dealing with uncertainty

Uncertainty arises in databases and reasoning with databases through the following reasons:

A: Lack of confidence in the data and rules

We may be uncertain about the data and the rules governing them. In a medical database for example, results of medical tests may be only partially conclusive, and so one may find it useful to enter a datum with a certainty number between 0 and 1. Alternatively, one may assign a confidence interval for the datum. Thus A may be entered in the database as:

A: 0.61

or as:

A: [0.4, 0.8]

Another area of uncertainty of this type, is uncertainty in the reasoning rules, especially the non monotonic ones. It may be that from A we can deduce B only with confidence 0.75 or confidence interval [0.4, 0.7], so the rules will be written as:

From A deduce B: 0.75

or as:

From A deduce B: [0.4, 0.7].

Examples (Kulikowski)

(1) If the starter is making odd noises the probability of a bad starter is 0.75,

or equivalently:

If the starter is making odd noises the probability of a good starter is 0.25.

(2) If a car has a bad starter then the starter will make odd noises in 0.87 of the cases.

The above probabilities may be given by a car mechanic.

We can have another rule:

(3) The probability of a bad starter (when a car won't start) is 0.02 (before looking at any specific symptoms).

The 2% was arrived at by looking at how many cars are at repair shops with starter problems relative to the total number of cars in repair shops.

Here is a fourth rule, directly measured, to complete the picture.

(4) The probability of a normal starter making odd noises is 0.07.

There are problems with such models:

(1) It is not clear what these numbers mean. How does one get them, and how does one update them?

For instance, in the above example, if we use rules (2), (3), (4) and use Bayes's formula to compute probability (bad starter gives odd noises), we get 0.202 and not 0.25 as predicted by the expert.

We see here that the probabilities or uncertainty numbers people give do not always match.

(2) One has to figure out the correct intuitive way in which the uncertainty numbers propagate through the system. If Ai have uncertainty xi and a deduction is used through rules Rj whose uncertainty is yj, to give an answer to a query ?G, then what is the uncertainty in G ?

Does it depend on the computation path?

Does it depend on the order the rules are used?

(3) If G is now measured and found false or is known to be true with a high confidence number, how do we update our confidence numbers and rules? Can we let the system "learn" automatically?

Systems used in practice usually fail on the updating and chaining of numbers. Odd results arise. Different computation paths get different numbers, to the extent that the numbers become meaningless.

Expert Systems have had great success with one step predictions. That is data Ai is entered with certainty factors and one step rules of the form

C if D1 and ... and Dk

are also entered with a certainty number. The system can predict, on the basis of the Ai which C is most likely.

Since the use of confidence numbers seems problematic, researchers developed special logics, called fuzzy logics, or many valued logics. These logics can be described and studied in a normal way like any other logic. They offer more hope of integration into our general framework. It is possible to construct a fuzzy intuitionistic like logic and use it fruitfully.

<u>B</u> <u>Uncertainty arising from inconsistency</u>

We have met with this phenomenon before, in the section on inconsistency. If we have conflicting information then although we may decide to ignore one item and thus avoid contradiction, we still bear in mind that we may be wrong and keep a watchful eye for more evidence. We do not have to associate numbers with data and rules, but just mark them as doubtful or very doubtful. It is important to figure out how this system interacts with updating and how it can be integrated with part A of this section. In other words, what is the meaning of "contradiction" and doubt, in a system with confidence numbers.

<u>C</u> <u>Uncertainty due to lack of information or a simplified model</u>

This is a simple case where the real model is causal but uncertainty is introduced because we do not know about it. If C is caused by A and B then we can write the rules:

C if A and B.

If we don't know about A we will have to write:

C if B: 0.6

where 0.6 is the frequency with which A appears.

It may be of interest to examine the thesis that all cases of uncertainty can be explained away by hidden unknown factors.

To explain what we want to do, we first look at an example.

Consider the axiom system with the axioms and rules listed below:

<u>Axioms</u>

(1) $A \to (B \to A)$

(2) $(A \to B) \to ((B \to C) \to (A \to C))$

(3) $(\neg B \to \neg A) \to (A \to B)$

(4) $((A \to \neg A) \to A) \to A$ ($\neg$ is <u>not</u>)

<u>Rule</u> modus ponens

$$\frac{A,\ A \to B}{B}$$

The above axiom system forces the logic to be a fuzzy logic, with 3 values. True, false and medium.

It is the well known logic L3 of Lukasiewitz. Similarly one can write axioms and rules to make the logic fuzzy with an infinite number of values. Classical logic can be obtained by varying the axioms. Intuitionistic logic can also be obtained by varying the axioms. See the end of Appendix 1.

The following can be shown (D. Scott):

Many valued logics (i.e. fuzzy logic) are just like any other logic, and can be described using axioms and rules in a framework which is not fuzzy at all but has the two values, truth and falsity.

The implications of the above to our plans are considerable. We do not need uncertainty numbers in order to describe uncertainty. We can achieve the effect by choosing the correct reasoning rules!

We don't expect to do everything by reasoning rules. We believe that reasoning rules and modalities like <u>certain</u>, <u>in some doubt</u>, <u>very doubtful</u>, <u>false</u> can achieve the desired effect.