EXPERT SYSTEMS FOR STRUCTURAL DESIGN.

C.J.BURGOYNE

Department of Civil Engineering,
Imperial College.

Presented at
Construction Industry Computer Conference
Barbican Centre, London.
11th February 1986.



EXPERT SYSTEMS FOR STRUCTURAL DESIGN.

C.J.BURGOYNE

DEPARTMENT OF CIVIL ENGINEERING, IMPERIAL COLLEGE

Over the last few years, we have become accustomed to the increasing use of computers in the design office. Very few offices now do not have at least some micro-computers installed, and most are used for something more than merely word-processing, running software which assists engineers in the design process. Many also run software which has become known by the the acronym CAD (for Computer Aided Design).

In the field of structural mechanics we are still a long way from true CAD software, and it is the intention in this paper to consider why this should be so, how things may change in the future, and what we can do now to achieve the desired results.

Summaries of existing work on expert systems in the civil engineering field have recently been given elsewhere (1,3), and there are a number of good references on the general principles of expert systems (6,10). Some of the implications for engineers have also been discussed (17).

Existing expert systems are constrained by the capacity of machines and languages available, and also by the level of development of Artificial Intelligence. Together, these have meant that the sort of problems that have been successfully tackled tend to be fairly trivial, but we must expect the limits to be pushed rapidly back to enable us to write systems that are of genuine value.

The examples from computer programs and expert systems given in the paper deliberately do not correspond to any particular syntax; they will be given in a form as close as practicable to natural language so that the principles involved are made clear.

## CONSTRAINTS OF CONVENTIONAL LANGUAGES

The original computers adopted an architecture due to von Neumann which has proved very durable and very difficult to improve on. The central processor (CPU) works its way through a sequence of operations in a predefined way. It can make choices, which allow it to follow different paths through the program, but only those which the programmer has foreseen. This process

fits in very well with the way we have traditionally tackled numerical problems by hand, and is thus fairly easy to program.

Conventional computer languages match this procedural form of operation. FORTRAN, PASCAL and BASIC, which are the languages most commonly used for engineering and scientific work, need to have all routes through the program predefined by the author. This leads to very complicated collections of IF .... THEN .... ELSE statements, often with conditions based on checking flags which have been set elsewhere within the program (Figure 1). There are also problems caused by the fact that the state of the variables changes with time, so that a particular statement can mean different things, depending on when it is called. Verification of such programs, to ensure that the desired route is taken through the program, and that the desired results are achieved, is very difficult, and often not carried out systematically.

Attempts have been made to speed up the CPU by taking as many operations as possible away from it, so that disc access, printer access, and display are handled by specialist chips that allow the CPU to get on with the main program. Increasingly, computers have floating point processors which can carry out arithmetic faster than the CPU, but these work under the control of the CPU, and not independently of it (Figure 2).

Parallel processing, which provides multiple conventional processors, and thus the possibility at least of the execution of multiple statements simultaneously, has not yet been fully developed because of the problems of ensuring that all processors are working with the same set of data. Parallel processing does not fit in with our own way of working, and its use is restricted to specialist tasks with very predictable data changes and much repetition, such as the solution of very large sets of equations, or numerical simulation of large systems, such as weather forecasting.

The computer programs that have been written for existing conventional machines have been written to mirror the normal procedural mode of operation, and are thus constrained to follow standard algorithms. We have very efficient programs to solve large sets of simultaneous equations, find optimal solutions to problems, and to analyse complicated structures. Large amounts of research effort have gone into producing efficient programs that do the work in the minimum possible time. But these programs do not confront the real problem facing the engineer, that of design.

It is important to be able to draw the distinction



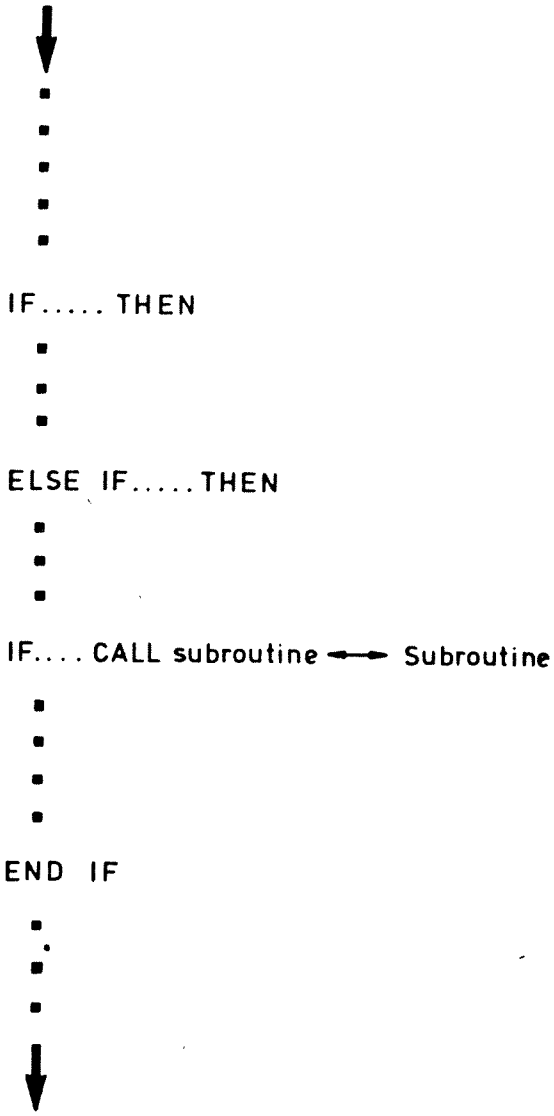


Fig 1. Flow control in a procedural program

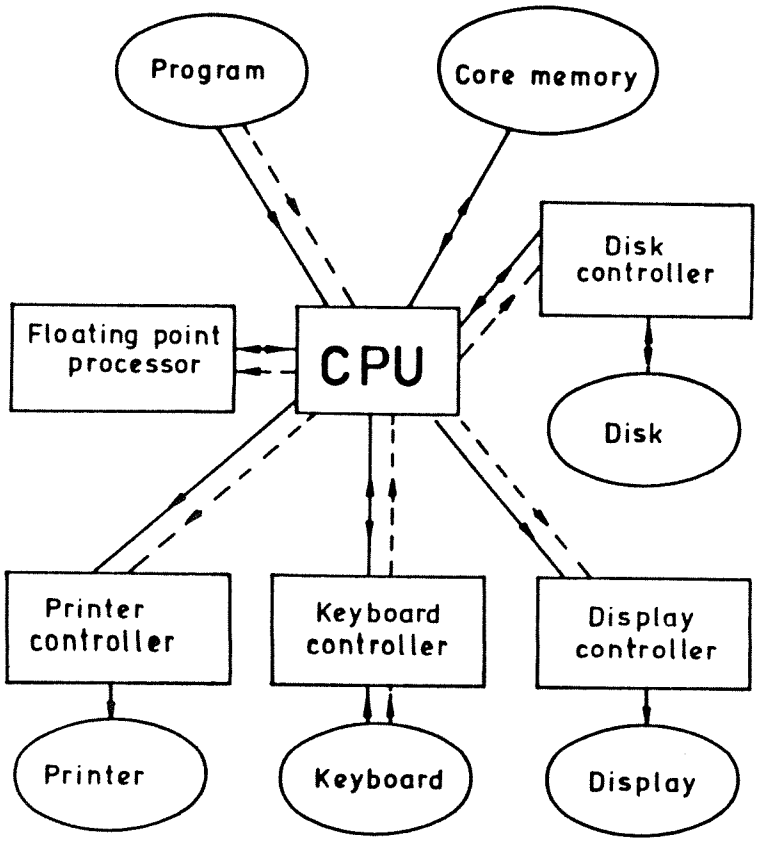


Fig 2 Schematic layout of von Neumann computer



between design and analysis. When we are taught, or teach, structural mechanics we usually implicitly refer to the analysis problem. "What is the response of this structure to that set of loads?" There is normally a unique solution to the problem and this can often be obtained by considering the structure to be linearly elastic, and solving a set of simultaneous equations derived from a space frame or finite element model. Even if the structure or material are non-linear, there are well-established techniques for finding the solution.

The design problem on the other hand, is much less amenable to solution; "What structure do I need which will carry this set of loads with a satisfactory response?" There is, in general, no unique solution. Harris (12) and others (9) recently discussed the problem of whether design could be taught. The general conclusion was that it could not, although students could be given an insight into the design process by taking part in project work. The reason it cannot be taught is that it is very difficult to determine an algorithm for general design problems; it is too much of a 'black art'. That conclusion is directly relevant to our present discussion.

Although there may be many valid solutions to the design problem, there are clearly some which are 'better' than others. But can we define 'best' in a meaningful way? We could price everything and produce a minimum-cost design, but the costing process is itself highly subjective, with many contractors not willing to release true rates and many special cases caused by such factors as the amount of repetition, which can affect formwork and other costs, and the sequencing of operations. Once again, these considerations are directly relevant to our discussion, since if we cannot define 'best', then design using optimisation techniques is at best suspect and at worst, a waste of time.

## EXISTING 'COMPUTER AIDED DESIGN' SOFTWARE.

We have so-called CAD software, but this acronym is a much abused catch-all that means different things to different people. It encompasses, inter alia, computer aided visualisation, computer aided analysis, computer aided manufacture and computer aided simulation. We need to be clear about which we are referring to in any particular case.

In structural design, we often use programs which stand between the user and the main analysis package. These front-end programs allow the user to inspect or modify the input data, without having to know the exact syntax required by the main analysis program, and to display,

in whatever form is most suitable, a selection of the results from which further choices may be made about the design (Figure 3). The program is acting very like the wordprocessor used to type this paper; it allows selective changes to be made to the input data without having to retype all the numbers: It also shares another similarity, in that it has no intelligence. The wordprocessor did not write the paper, and the computer aided analysis (CAA) program cannot design the structure.

The reason that existing programs cannot design structures is because of the multiplicity of choices that have to be made during the design process. These choices interact in a very complex way, and given the fact that with conventional algorithmic languages the programmer must have thought out all the possible routes through the program when writing it, it is not surprising that programs written in such languages either do not work, or are restricted to a very trivial set of problems.

## DECLARATIVE RULE-BASED LANGUAGES.

New programming techniques are now becoming available which should dramatially improve the way in which programs can be written to simplify the design process. This work comes out of the theoretical work on Artificial Intelligence (AI) that has been underway for some years (14).

The work centres on the way we represent knowledge, and the way we can draw inferences from it. The basic building block is the rule, which represents one piece of information; this may be used to represent either a fixed piece of data, the relationship between a number of pieces of data, or the means by which something can be calculated. One important property of such rule-based systems, provided that the rules are purely declarative and have no procedural interdependence, is that the order in which the rules are placed in the rule base is immaterial.

To be useful, rule bases need to be provided with mechanisms for drawing conclusions from the built-in knowledge; this is known as the inference mechanism. We thus replace the procedural programming idea of

Program = Algorithm + Data

by the declarative form

Program = Inference + Knowledge.

Systems capable of dealing with data in this way have



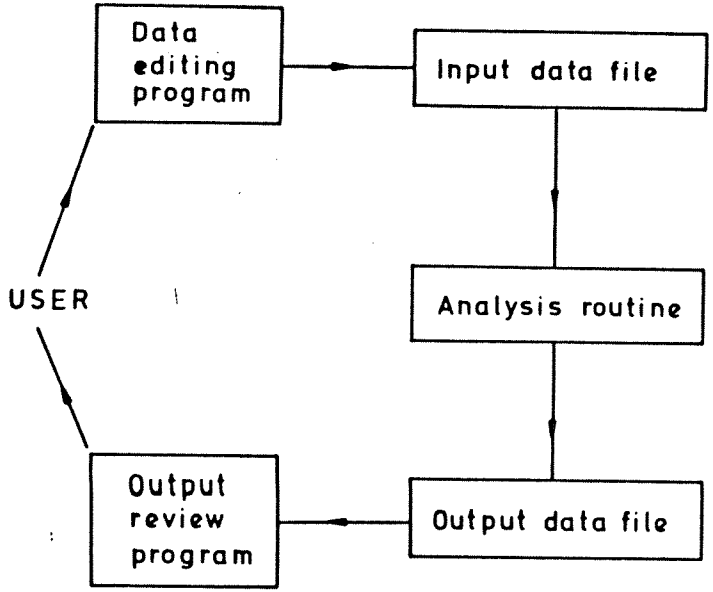


Fig 3 Existing "CAD" programs

been in existence for over 20 years. It is possible to write a program in a procedural language to interrogate a data file containing a rule base, but it is much more conveniently done using one of the declarative languages, such as LISP or Prolog.

There is some debate about which languages are best suited to expert systems work; indeed, the case for declarative languages has not been conclusively proved, but such discussions are not relevant in our present context.

LISP was devised by Mccarthy in 1960 for LISt Processing, but has become the favoured tool of AI workers in North America. It does not offer in built inference mechanisms; these have to provided by the user, but this gives it a degree of flexibilty not available in some other languages.

Prolog (for Programming in Logic, was developed by Colmerauer at Marseille in 1972, and has been chosen as the fundamental language of the Japanese 5th generation computer initiative. It is widely used outside North America for AI and Expert Systems research (8), but does not as yet have such a wide user base as LISP.

The basic form for the representation of a rule in Prolog is the Horn clause.

A is true if B is true and C is true and . . . .

where 'true' here means 'implied by the users rule base'.

The inference mechanisms are contained within Prolog, and it is possible to ask questions in the form

For which set of data is it true that . . . ?

and

Is it true that . . . . ?

Prolog is imperfect when handling negation, where not all the data in known, which can cause problems in rules of the form

A is true if B is not true

but it can be used to handle rules in either direction. For example, a single rule which expresses that x is the sum of y and z can be used to evaluate either x, or y, or z according to the particular circumstances.



## EXPERT SYSTEM SHELLS

Expert system shells are programs which are put around the rule based package to improve the interface with the user.

They assist the user in the construction and maintenance of the rule base, allowing rules to be added, modified and deleted as required. While in normal use, they provide improved facilities for prompting the user for data, checking the validity of replies, and perhaps most importantly, providing justification of the conclusions the program has come to by tracing back through the line of reasoning the program has used. The shells can also provide links between the rule-based language and routines written in other languages, which, as we shall see later, will be important for true design systems (Figure 4).

Expert system shells can themselves contain rules which provide a natural language interface between the main rule-base and the user. By modifying the rules in the shell itself, but keeping the same rules in the expert system, it will be possible to use the same expert system to answer queries in English, French, Spanish etc.

Despite the importance of the shells to expert systems, we do not need to concern ourselves at the moment with the details of the shells. Detailed work is reported elsewhere(4,5,11).

## THE HUMAN DESIGN PROCESS

Before we can consider automating the design process, we must consider how the engineer performs design by hand. We have already seen that we do not have general algorithms for design; there is no set sequence of operations we can perform to produce the 'correct' solution.

In practice, designs are produced iteratively. Initially, all variables are unknown; member layout, member dimensions, reinforcement, prestress, joint details all have to be fixed.

Although the design process will vary from structure to structure, the basic principles are encompassed in the following strategy (Figure 5). Based on his experience, the designer will choose the member layout; he will then perform some calculations based on a very simplified structural analysis to determine the bending moments and shear forces in the principal members. These, together with simple calculations of member strengths, will allow the choice of member dimensions. Subsequently, a more accurate analysis will allow the

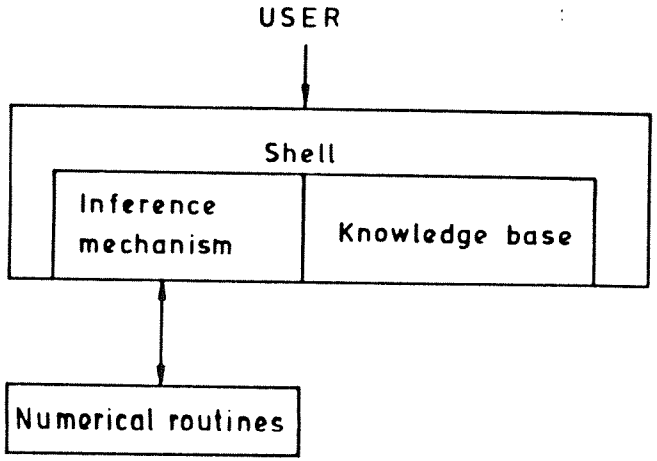


Fig 4 Simplified layout of expert system



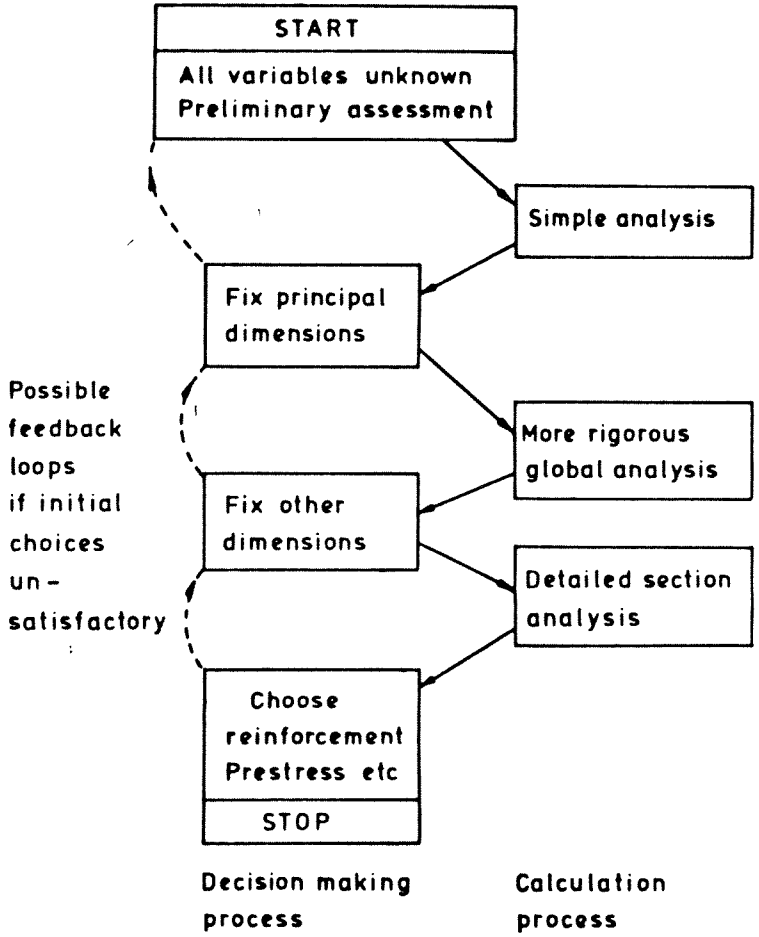


Fig 5 Simplified manual design process

determination of the stress-resultants everywhere, which will allow the sections to be detailed (either reinforcement or prestress in the case of concrete members, or stiffeners in the case of steel structures).

This strategy is highly simplified, but it illustrates the basic principle. We have elements of choice, alternating with elements of analysis. The choices are based on the engineer's own experience, on the results of previous analyses, on information he can look up in reference books, and on his knowledge of what he has to do to comply with the codes of practice.

The analyses that are performed are of increasing sophistication. The initial calculations may be at the 'back of an envelope' level, to determine orders of magnitude. Subsequent calculations may involve more rigour, such as continuous beam theory, or frame analysis, while the final analyses may involve comprehensive finite element analysis, at least of detailed areas, such as around joints, prestressing anchorages, or other stress-concentration regions.

The whole strategy is based around the avoidance of wasted effort. Carrying out the simplified analyses to determine the overall section dimensions usually means that subsequent analyses can be carried out on the assumption that the results will be satisfactory. The final calculations are done to demonstrate to others (client, certifying authority, contractor etc) that the solution is valid. Only rarely will the results prove to be unsatisfactory, and this is most likely to occur at an intermediate stage, so that the whole process does not have to be repeated.

If the designer is inexperienced, cannot think logically, or is trying to produce a design which is 'sailing too close to the wind', then more of the analyses will fail. With luck, the engineer can determine what particular aspect of his proposed design caused the failure, and can modify the design; the structure can then be reanalysed. In the worst case, we move from design by a controlled strategy to 'design by repeated analysis'. The intermediate analyses are wasted, since they do not refer to the structure in its final form, and design by this method is to be avoided if possible.

One aspect of design that is often overlooked in computer based systems is that intermediate calculations frequently do not involve the complete structure. Analyses are often performed of the partially complete structure to identify trapped moments caused by the erection procedure. In many cases, influence lines are required to allow the correct choice of loading to determine maximum effects;



ah analysis has to be carried out with a cut or hinge at the point being considered. Similarly, special analyses of the incomplete or modified structure are often needed to be able to determine the effects of self-straining loads, such as prestress, creep or residual stresses.

Most packages allow the combining of stresses from different load cases, when applied to structures with the same geometry, but very few allow such combinations to be made for structures with different geometry.

Before we consider how this process may be automated, it is worth considering and abandoning one other option for computer assisted design; optimisation.

## THE FALLACY OF DESIGN BY OPTIMISATION

One technique that has been around for some time and that has been seriously considered for design purposes (7) is the use of optimisation techniques. The structure geometry is defined in terms of a number of parameters; in the case of a bridge, they may be spans, member depth at pier and mid span, flange and web sizes, etc. A penalty function is identified which will be minimised for the optimum solution. In civil engineering, this is usually cost (despite the drawbacks referred to earlier), but can be weight (more common in the aerospace industry), or construction time.

Constraints are identified which the structure must satisfy; these would frequently include stress, deflection and strength as governed by the codes of practice, but may also include practical considerations, which govern whether the structure can be built.

An initial estimate of structural form is made, and the structure modified using linear programming, or other optimisation technique, until the best solution is found which satisfies all the constraints.

However, optimisation techniques do not work well with discontinuous constraints (13), which are quite common in real problems. For example, while it is quite simple to get a solution which specifies the optimum area of reinforcement in a section, it is much more difficult to find the optimum solution which uses an integral number of bars of one of the specifed standard sizes.

The principal drawback of such techniques is the number of analyses required. It is not unusual for several hundred analyses to be performed, and they will all need to be of the same level of sophistication as the

most rigorous analysis performed in a hand design. By comparison, an experienced designer would not expect to perform more than 2 or 3 rigorous analyses. Optimisation is 'design by repeated analysis' taken to its illogical conclusion.

## DESIGN USING EXPERT SYSTEMS.

If we wish to perform design using expert systems, we shall try to mimic the process performed by human designers. That may sound a simple task, but it is one that involves great complexity, because we are not looking merely at what the designer does, but at why he does it that way.

Most engineers can explain what they do; indeed it is one of the fundamental parts of his job to be able to justify his final design to others. But this justification is usually in the form that

"this structure will be satisfactory under these loads because I can show that .....",

(which is analogous to our definition of the analysis problem) rather than

"I chose this structure because .....",

which is what we need to know to be able to automate the design process.

There will thus need to be considerable work to get behind the apparent process of design to arrive at the underlying strategy, and it is very likely that different engineers will adopt different approaches to solving the same problem. This work will need to be carried out by engineers who are familiar with the design concepts involved, but are also familiar with the requirements of programming in rule-based languages. There are plenty of people around with the first skill, and a few with the second, but people with both are very rare at the moment.

To make these ideas work, it will be necessary to get through the idea that design is a 'black art', and this should have a beneficial effect on the design process itself. It is likely that the strictly logical work needed to encapsulate the design process in a codifiable form will allow more rational discussions to be held about design. Thus, we should expect to see, within the next few years, a number of technical papers outlining the design process for various structural systems. There will be those, who in contributing to discussions on the papers claim that 'this is all obvious'; obvious to them perhaps, but probably not obvious to everyone else. The time is now ripe for



extensive study of the design process.

What form will the rule base for design take? The rules will naturally fall into three distinct categories, and may be provided, for any particular design, from a number of different sources; a process which is quite feasible in a declarative system, but is very difficult to do in a program based on procedural programming techniques (Figure 6).

One aim of determining the rule base is that as far as possible, the same rules should be used, irrespective of the form of the problem being tackled. In many cases, it should be possible to arrange that the same rules are used for the design of a totally new section, the analysis of an existing section, or the strengthening of an existing section.

The first two categories of rules should prove to be non-controversial. They will relate to statements of fact and definition which are valid in all circumstances. There may well be considerable discussion about the third set, however, which will form the kernel of the design process. The basic categories will be:-

(i) All fixed sets of data, such as standard section properties, tables of available reinforcing bar sizes and tables of strengths of different types of steel. These rules could be provided from standard sets of manufacturer's data (section properties), from codes of practice (safety factors and cover requirements), or from data supplied by the client (loading and spacing requirements).

(ii) All definitions which relate together the various quantities, and which are valid in all circumstances. In this category would be included such items as the calculation of stiffnesses from member dimensions, calculation of resistance moments from member dimensions, calculations of applied moments from applied loads, and rules which contain the clauses of the relevant codes of practice.

These definitions should not differ from engineer to engineer, although they may differ from country to country because of the different codes in use. We may disagree about the rules within the codes, and it may well be that the discipline enforced by the use of computer systems will lead to a rationalisation of the codes, but this should not prejudice us against the system itself.

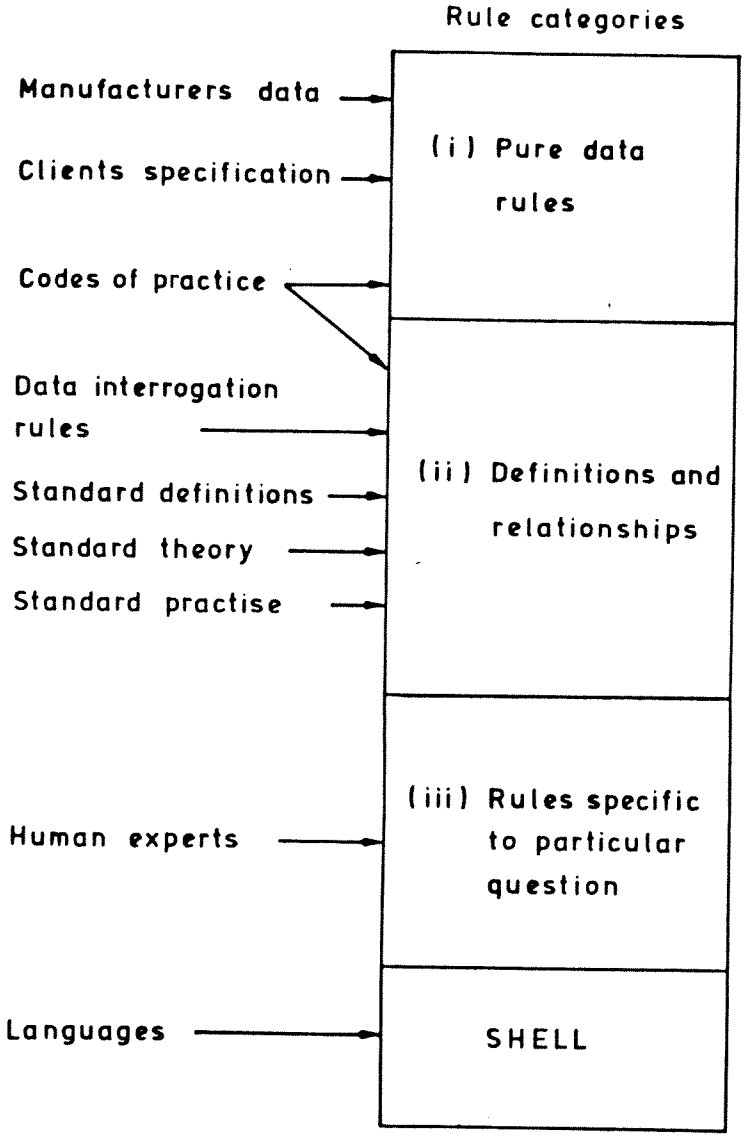


Fig 6 Assembly of rules from different sources



As with the first category, these rules may be derived from many different sources. Some will come from practical considerations about what it is reasonable to specify, or what can be built (eg overall depth should be a multiple of 50mm, or rules relating to how much steel can be fitted into a given cross-section). Others will come from codes of practice (for example, rules which relate to the loading). Here we see one of the great strengths of rule-based systems; the fact that rules can be provided in any order, and from any source, means that we can replace a rule which says

"The loading is x " derived from BS5400,

by a rule which says

"The loading is y " derived from the AASHTO code,

secure in the knowledge that we are not causing problems elsewhere in the program.

This group of rules will also include rules which allow the abstraction of data from the first, purely data, set of rules; for the sake of convenience, however, it is likely that the data extraction rules would be provided as part of the data base itself.

(iii) The final category includes only those rules which are specific to the question being asked of the system; thus, this set will include one set of rules for tackling analysis type problems, and a different set of rules for tackling design problems.

It is to provide these rules that we have to formalise the design process; it is these rules whose validity will need to be discussed and agreed before expert systems become widely accepted; it is on determining the correct form of these rules where the majority of the research and encoding effort must be directed.

If the rules in this third category are written suitably, then it will be possible in, say, a bridge design package, to answer diverse queries such as

"What bridge do I need to carry 3 traffic lanes across spans of 20, 30 and 25 metres"

"Will bridge x carry 30 units of HB load"

and

"How long a span can I use this section for if the loading is RL loading to BS5400"

The system will, of course, need to be provided with rules which allow it to answer these queries, but it is likely that with relatively few rules specific to the particular problem, and rather more contained in categories (i) and (ii) outlined above, it will be possible to answer all these queries and more from a single expert system.

## LINKS WITH NUMERICAL SUBROUTINES

There is no doubt that, at least at the moment, there is no faster way of solving sets of simultaneous equations or analysing a given structure, than by using a well-tried algorithm written in a compiled and optimised language, such as FORTRAN.

Our expert system, running in a rule-based manner, will need to be able to call on analysis subroutines to perform the numerical calculations at reasonable speed. The rule base will decide on what structure needs to be analysed; the numerical subroutine will analyse the structure, and the results will be interpreted by the rule base.

There will need to be interfaces provided between the inference part of the program, and the numerical part of the program, and vice-versa, to allow the interchange of data. Most machines will support multi-tasking operating systems, so it is quite feasible for one package to generate a data file, which is then used by the other package. More direct pipelining techniques are also possible.

## IMPRECISION

Nothing has been said hitherto in this paper on the use of imprecise reasoning in structural design, but imprecision underlies many of the fundamental principles of the subject: Hence our factors of safety, which are quite large because we do not know accurately the magnitude of some of the quantities we have to use.

Expert systems, particularly those written in Prolog, may allow us to incorporate imprecision into our design process directly (2), rather than indirectly, as it is now done. It is not suggested that all design could benefit from this approach, but in certain structural applications where an understanding of the probabilities of failure is required, it would



certainly be of benefit.

Imprecision can be incorporated directly into our reasoning in the following way. Instead of the precise statement

A is true if B is true and C is true

we can introduce a measure of support for each of our statements (we shall not, here, attempt to define what we mean by support, but it could be probability or one of the measures of support adopted in fuzzy logic); this allows us to write an imprecise version of the same rule

A is true with support x if
B is true with support y and
C is true with support z and
y combined with z gives x

The last clause, which has been deliberately left vague, is used to define some relationship between the support quantities x, y, and z. It may be a clause which is specific to the problem in hand, but more likely, it will be a clause written in terms of a standard theory, such as probability or fuzzy logic. Expert systems have already been written in this way, for example the MYCIN project looking at medical diagnosis (15), and much work is both underway and remains to be done before practical applications can be made in our fields. Not the least of these is the problem of obtaining the basic data for the values of support in the first place.

## COMPUTING REQUIREMENTS OF EXPERT SYSTEMS

Expert systems are very heavy consumers of computer power, both as regards CP time and memory. Existing systems of course, which are still, strictly speaking, experimental, run on the current generation of von Neumann computers; these, as we have seen already, are ideally suited to procedural programs.

New computer architectures are however being proposed, under the umbrella title 5th generation computers, which, while still capable of running procedural programs will have architectures much more suited to running rule based packages as well (16). The efficiency of the inference mechanisms will be increased and the speed and memory available in the computers themselves will continue to increase. One of the elements of recent defence initiatives, such as the SDI, will be the production of extremely fast computers capable of making decisions in real time about which targets to attack. Although it is widely believed that the increase in computing power required for that

particular application will not be possible, sufficient improvement will be made to have a significant effect on expert systems for civil applications.

Declarative formulations, unlike procedural programs, can also make sensible use of parallel processing, so that improvments in speed can be made by providing multiple processors.

For example, if A can be true in either of two ways

A is true if B is true

A is true if C is true

then it is perfectly feasible to get one processor checking whether B is true, while another is dedicated to testing whether C is true.

## CONCLUSIONS

The new approaches to computing opened up by the fifth generation computer architectures, intelligent expert systems and logic programming offer far more than a better way of performing the same calculations that we currently undertake.

They give us the opportunity, for the first time, of being able to produce true computer aided design software, as design is understood in the field of structural mechanics.

The techniques that are being offered will call for a rational reappraisal of the existing human design processes, that should have benefit, not just for CAD applications, but more widely in structural engineering.

We are just at the beginning of a very important change in the way design is carried out.

## REFERENCES


1. Adeli H. Knowledge-based expert systems in structural engineering. Proc 2nd Int. Conf on Civil and Structural Engineering Computation London 1985.

2. Alim S. Fuzzy expert systems. PhD thesis. University of London 1985

3. Alim S and Munro J. Prolog based expert systems in Civil Engineering. Submitted for publication. Proc. Inst Civil Eng.





4. Allwood R J, Stewart D J and Trimble E G. Some experiences from evaluating expert system shell programs and some potential applications. Proc 2nd Int. Conf on Civil and Structural Engineering Computation London 1985.

5. Allwood R J, Stewart D J, Hinde C and Negus B. Evaluation of expert system shell programs for construction industry applications. Dept of Civil Engineering, University of Loughborough, 1985.

6. Alty J L and Coombs M J. Expert Systems. Concepts and examples. National Computing Centre 1984

7. Bond D. Optimisation of prestressed concrete structures. Paper in 'Developments in prestressed concrete - 2'. Ed Sawko. Applied Science 1978

8. Clark K L and McCabe F G. PROLOG: a language for implementing expert systems. In 'Machine Intelligence' Vol 10. 455-470. Ellis Horwood 1982

9. Cowan J. Design education based on an expressed statement of the design process. Proc. Inst. Civ. Eng. Vol 70. Nov 1981 743-754

10. Forsyth R (Ed). Expert Systems, Principles and case studies. Chapman and Hall Computing 1984

11. Hammond P. A PROLOG shell for logic based expert systems. Proc. Conf. on Expert Systems, British Computer Society, Cambridge 1983

12. Harris A J. Can design be taught. Proc. Inst. Civ. Eng. Vol 68. Aug 1980 409-416. See also discussion, Vol 70. May 1981 343-354

13. Hu T C. Integer programming and network flows. Addison Wesley 1969

14. Rich E. Artificial Intelligence. Mcgraw Hill 1983

15. Shortliffe E H. Computer based medical consultations. MYCIN. Elsevier 1976.

16. Simons G L. Towards fifth-generation computers. National Computing Centre 1983

17. Taffs D. Expert or knowledge based systems: consequences for the industry. Conf. Impact of Computer Technology on the Construction Industry. London 1984.