HOW EXPERT SHOULD AN EXPERT SYSTEM BE?

Roger T. Hartley

Man-Computer Studies Group,
Computer Science Department,
Brunei University, U.K.[1]

ABSTRACT

A computer system which aids computer engineers in fault diagnosis is described. The system, called CRIB (Computer Retrieval Incidence Bank) is shown to fit into the class of pattern-directed inference systems. Emphasis is placed on the "before" and "after" phases of system generation and it is shown why, to be called an expert system, these phases are important. The forms of knowledge used in CRIB are shown to be adequate for diagnosis and yet possess little of the structural or functional knowledge of more advanced expert systems. Summaries are given of the three phases of implementation: elicitation, implementation of knowledge structures, validation and improvement. The idea of an expert system as a "model of competence" is mentioned and the transferrance of the system architecture to software diagnosis, using the same model, is described. There are short discussions of system performance and the nature of expert systems.

1. Introduction

This paper, whilst also being a vehicle for presenting the work on CRIB carried out between 197*4 and 1979. attempts to investigate the "before" and "after" of an expert system. Most of the papers written on and around the field of knowledge engineering systems have concentrated on the structure of the program itself with short discussions of how the program came into being and how the program performs "in the field". However, an expert system is a non-academic exercise (not to say commercial) and the twin needs therefore to get the detail right beforehand and the performance right afterwards, are equally important as the system design.

Four phases of the route to a viable expert system may be distinguished. They are not dissimilar to the staging of the development of any commercial computer system. They are:

1. Elicitation for the first approximation knowledge base.
V. System design and generation.
3. Continuous validation (debugging and fine-tuning).
4. Improvement through experience.

Notice that we have not pre-empted our discussion of elicitation by specifying the source of the initial knowledge for the system. In order to discuss this point we should like to make a distinction in the people who come into contact with the system, i.e. between experts and practitioners. We consider this a useful one especially where the system can be used for education and dissemination of knowledge as can a standard enquiry data-base. Practitioners are those people who use the system in the course of their every-day works experts are the handful of people who define what the work is and how it should be carried out.

2. The "Before" and "After" of an Expert System.

2.1. Elicitation: from experts or practitioners?

The dramatic rise to academic consciousness of the expert system has come about partially because of the cult of the "expert" and his largely undis- uted place in our society. However, an expert is only an expert relative to the practitioners in the field. Moreover, experts are always a tiny minority of the whole group. We would therefore like to ask the question "Is an expert necessarily better than a large number of practitioners?" The answer to the question must surely depend to some extent on the sort of knowledge involved. If knowledge is fragmentary, consisting largely of facts which are additive in nature then the answer is "no". Practitioners acting separately (or at least only loosely connected) have different experience and therefore different knowledge. The expert has only his own experience to rely on and, whereas he might have a far wider experience than any one practitioner, he cannot match the aggregated experience of a large group of them. If, on the other hand, the knowledge needed is structural or systemic in nature, then the answer to our question is "yes", the expert is better. In fact, it is often the case that experts are expert because of their ability to see the structure of the domain and not merely its content.

In this paper we shall not discuss whether knowledge is of one form or the other. There is some discussion both in A.I. [11], [13] and.

• From September, 1981 at Department of Computer Science, Kansas State University, USA.

perhaps more relevantly, in philosophy (for example see [9], [15] for general texts) which we refer the reader to. The only point which must be made here is that the sort of knowledge which one must have to be effective in a particular domain is certainly dependent on the activity or activities within that domain. For example, the knowledge required to be a good bricklayer is mainly additive, consisting of facts such as how to chop a brick in half; what proportions of sand and cement make good mortar; how to build a corner. On the other hand, the knowledge required by the builder employing the bricklayer is organisational in the main. He must know how to time the work of subcontractors where the work of one depends on another finishing; how to balance incoming and outgoing payments so as not to go bankrupt; when to quote a high price for a job he cannot find time to do.

It thus appears to be dependent on the domain whether knowledge should be first elicited from experts or from a community of practitioners. Can we then say something about the areas in which expert systems have gained a foothold and generalise this to future *or* projected applications? All the early expert systems (see [8]) concentrated on academic or scientific areas of knowledge. The knowledge involved was aimed at finding and making explicit that which was previously implicit; it involved a high degree of intellectual skill and thus clearly fitted into the "expert elicitation" category. It is true that they also used a large body of factual knowledge, but this was decidedly subordinate to the knowledge directly concerned with "expertise" in the area. The facts were taken for granted and formed a significant but largely static part of the system. If what we have said before has any validity, there should, then, be areas of human activity in which the notion *of* an expert is not inappropriate and where the expertise depends in part on a large and ever-changing body of facts. Of course the expert in such a field will have the structuring skills of the builder in our example above, but the aggregated practitioners will have more factual knowledge and thus should be the source of the initial knowledge-base. We believe that computer fault dignosis is *one* such area (the area in which CRIB operates) and that there are others of a similar nature (almost any skilled trade, but also such diverse fields as navigation, production scheduling, or information services).

Since most knowledge based systems split knowledge into two, namely static knowledge (facts) and methods of inference, organisation of the techniques of elicitation around these categories can help both the interviewer and the interviewee. When working with an individual (the expert), both sorts of knowledge tend to come together quite smoothly, but working with an aggregate is different. Firstly, elicited facts are only additive if inconsistency and contradiction are allowable. An expert system, however, cannot afford to show these qualities to the user. It must have a way of checking its self-consistency and reporting errors when they occur. Some sort of semantic network is the usual way of doing this although the problems of building networks with sufficent expressive power are well known [5], [13]. If these problems are to be avoided, then there must necessarily be a diminution in the generality of knowledge which the system can maintain. In CRIB the sorts of knowledge which it stores are reduced to a minimum without reducing its efficiency as a diagnostic aid. These simple facts (see section 3.2) are easy to elicit using a variety of techniques; no semantic network is then necessary.

Turning to methods of inference, it is apparent to us that little has been done in developing elicitation techniques from experts. Most expert systems using production system methodology necessarily force all inference procedures into the uniform situation -> action pair. The elicitation is thus characterised by the question "what rules do you use?" rather than the more general "how do you make inferences?". Elicitation of methods of inference from an aggregate of practitioners involves a second level of system design since some form of induction must be used to capture commonalities amongst the members of the aggregate. This second level difficulty merely adds to the already tricky process of elicitation and does not seem worthwhile if an acceptable alternative is available. This alternative is for the system designer to supply methods of inference based on theoretical notions of what it is to reason well in the domain under consideration. A study prior to CRIB (called project DEEMEN, see [4]) suggested that a hierarchical model of the broken machine be adopted to guide a step-by-step search for the faulty part. The idea here is that the designer supplies a model of competence which the system then uses as the universal method of making inferences and guiding the user. Les Johnson [11] makes the point that most A.I. systems are in fact models of competence. For our purpose this description is correct since we have not attempted to induce methods of diagnosis from the engineers we worked with.

2.2. Validation and Improvement

If, as we have said, an expert system is a model of competence, then we must be prepared to change the model in the light of experience by users. The same problem occurs in this "after generation" phase as with the "before" of elicitation. It is mainly practitioners who use the system and it is their experience with it that can help to validate the model. But can this experience also be used to improve the model? In general we can say that the system users come to two different sorts of conclusion after gaining experience with it. Either the model is inadequate as a model of competence (i.e. the system is "incompetent") or it is failing to cope with a wide enough range *of* new situations and methods of working. This leads to the distinction between validation and improvement. An expert system which is improving in the more fundamental sense must be gaining new knowledge; validation can be carried out by restructuring existing knowledge. The same distinction also applies when considering

the changes of a system in time. Continuous validation through day-to-day use by practitioners can cope with small perturbations, like tuning a radio set for better reception, but new knowledge usually comes from an expert (tuning to a "better" station).

It is appropriate to ask whether improvement can come not from experts, but from an aggregate of practitioners in the same way as elicitation can when the form of knowledge is right. Even with a level of interpretation (presumably by an expert-system expert) it is hard to see how this could come about. Practitioners, by the nature of their job, do not normally generate new knowledge. If they do it is not of the structural sort, but is more likely to be additive - the sort of knowledge which an expert system should be able to handle by the "fine-tuning" process of continuous validation. Thus an expert sytem must have interaction with a domain expert to improve its effectiveness. With most systems this means modifications to data-base structures and/or procedures - modes of interaction not allowable to a practitioner.

3. CRIB: A Computer Engineer's Diagnostic Aid

3.1. Overview.

The initial aims for CRIB were strictly commercial: to reduce maintainence costs in two areas - training of new engineers and existing engineers on new equpiment; increasing productivity by reducing the average time per fault investigation. With constraints like these it was clear that a highly flexible, user-friendly system was needed. Ultimately, it was decided to aim CRIB at the CAFS data-base processor (see section 3.4 with its wealth of associated software packages and analysis techniques and, more importantly, its high effieciency as a pattern-directed data-base system. However, early versions used standard file techniques for database management. The CAFS version effected a 20-30 fold improvement in speed, using the same logical structure.

The whole CRIB system consists of three programs:

P1. The main program which carries out the executive function (see section 3.4) is called DIAGNOSE. It interfaces with the user (the field engineer) through a simple Jargon-English translation package. This allows the engineer to communicate with CRIB in a fairly natural way to describe symptoms, inform of actions carried out and control housekeeping functions such as the investigation log.

P2. Some degree of validation and fine tuning is done by ADAPT which examines the logs of previous investigations and updates times of actions performed, and investigation times after a complete match with a symptom group (see section 3.2). It also handles sub-groups (see section 3.4.1).

P3. Major restructuring of the data-base and the insertion of new knowledge are done through EXPERT. As mentioned before, this is a specialised access to the data-base and is done with the help of the system designer.

3.2 Forms of Knowledge for Computer Diagnosis

Two questions were posed at the beginning of analysis for system design. They were: What sort of knowledge does an engineer need to find faults on broken computers? and How does the engineer use this knowledge to find and cure the fault? The DEEMEN project told us that most field engineers know little about the correct functioning of the machine at the electronic level but a lot at the level of interfaces between modules. We therefore concentrated on the notion of a replaceable or repairable part (we called it a sub-unit) and the interfaces between them. The normal concept of "a fault" can then be dropped in favour of the location of the fault within a sub-unit. It follows that the only knowledge needed about the structure of the machine is the hierarchy of sub-units which expresses the relationship between them. Such a hierarchy can be drawn as a tree whose leaves are the replaceable/repairable modules.

Support for this came from the DEEMEN reports. In these, an easy-to-follow routine for diagnosis was suggested in an attempt to give engineers a fail-safe method of proceeding when all else fails. This procedure has the acronym TOAST and each letter refers to the following steps which when cycled successively produce a spiral of diagnosis, hopefully homing in on the fault by travelling down the hierarchy.

Test:	carry out an appropriate test on the machine
Observe:	observe and record the results
Analyse:	analyse the results and in the light of this:
Split:	split the faulty sub-system into faulty and non-faulty parts
Test:	generate an appropriate test for the faulty sub-system

TOAST provides support for CRIB as a model of competence in computer diagnosis. TOAST also needs knowledge of the relationship, through the Analyse phase, between the results of tests carried out by the engineer and the form that the next test should take. The analysis is simply represented as an association of symptoms and faulty sub-unit. This begs two questions: Do symptoms uniquely identify faulty sub-units? and How do we know which pairs to put in the data-base? The second question is concerned with elicitation and will be left for section 3.3. The experts we worked with told us that the answer to the first is "no": some symptoms are common to many faults and so more information is needed. We did not want to include knowledge of machine function and so the other information which we added to the pair was all of the other symptoms (perhaps ten or fifteen) which had been observed before the current one. Does a group of symptoms taken together uniquely identify a faulty sub-unit? If an engineer only observes these symptoms and finds the fault, then his analytic skills (if any) are reflected in the choice of actions which yield the symptoms in the group. We do not then need to represent the analytic skills explicitly

- they are represented implicitly through the symptom groupings.

However, some symptoms are only observed as a result of lengthy and intricate operations. The knowledge needed here is which symptoms are better looked for than others. Since CRIB has no functional knowledge it must base its assessment on attributes of actions which potentially yield symptoms in the various groupings. Attributes such as the time it takes to perform an action; how many groups contain it; and how long the investigation might take after it has been observed are only heuristically adequate in this case. However, since the final aim is to find all of the symptoms in any one group the heuristic element is only concerned with means and not ends.

The knowledge needed for a model of competence in computer diagnosis can be summarised thus:

KI. A hierarchy of sub-units representing the structure of the machine as a collection of repairable or replaceable parts.
K2. A number of symptom group/sub-unit pairings which represent successful fault investigations.
K3. Attributes of symptom-related actions which are heuristically adequate to determine the next action.
KM. Procedures for operating a modified TOAST cycle in order to progress down the hierarchy to the faulty sub-unit.

3.3. Elicitation of Knowledge in CRIB.

The hierarchy of sub-units (KI) was formulated in discussion with people who train field engineers. The main aim was to provide a comprehensive breakdown of the chosen "patient" machine (ICL 2903) to the level of replaceable or repairable modules. The bulk of the K2 knowledge came from an interpretation of a set of fault-finding guides for the 2903. These had been produced by an expert engineer using his experience of general fault-finding and applying this to what was then a brand new machine.

With hindsight, our first stage elicitation should have come directly from an expert (there were no practitioners at the time with any direct experience). Information could have been gained from system engineers during a series of interviews about specific fault investigations. The data could then have been cast in a form suitable for CRIB without engaging in the sort of interpretation demanded by the guides. We cannot emphasise strongly enough that the success or failure of an expert system can largely rest on this initial stage of elicitation.

3.4 System Design and Generation.

The description of the internal workings of CRIB given here follows the analysis given by Hays-Roth et al. [10]. Other more straightforward accounts may be found elsewhere [1], [2]. The categories of knowledge K1 to K4 have many of the characteristics of what they call a Pattern-Directed Inference System (PDIS). The group/sub-unit pairings can be seen as expressing the following inference:

R1. _if symptoms S1 to N have been observed then assume the fault is in sub-unit U.

The procedures in K4 will clearly have a matching function, in part; the process of looking for a suitable split to make (as in TOAST) will involve matching the current observed-symptom set with each of the pairings in the data-base. The symptom groups are patterns and the pairings are inferences.

In general, a PDIS consists of a large, constantly changing body of data and a relatively small, fixed set of procedures which operate on them. These procedures taken together form the executive of the PDIS. Hays-Roth et al. discuss PDIS executives in four parts: selection, matching, scheduling and execution. It is possible to discuss CRIB in these terms.

Selection: filtering data before the expensive matching process. The CRIB user is able to direct the program to look only for matches amongst symptom groups paired with any chosen sub-unit and not, as is normal, with all of the sub-units below the currently assumed faulty one. This can improve search efficiency and CRIB can then quickly inform the engineer whether he has located the fault correctly *or* not.

Matching: carrying out the matching between observed symptoms, input by the user, and symptom groups in the data-base. This function is carried out by the CAFS data-base processor. Enquiries, in the form of Boolean expressions of attributes known to the data-base, are presented to CAFS as a microcode program. The device then returns selected parts of all records which satisfy the request to the mainframe via a DMA link. Information about the CAFS side of the CRIB implementation are contained in [2], Technical information about CAFS can be found in [3].

Scheduling: three outcomes are possible after matching has been completed. The first is that there is one and only one match with a rule in the data-base. In this case scheduling is not needed. The other two are more difficult. Either there is more than one match or only partial ones. What to do next is the subject of meta-rules which describe how to proceed in situations of imperfect matching. The meta-rules in CRIB for handling partial matches are aimed at suggesting actions to the engineer which will yield symptoms to complete the match. The rules *are:*

MR1. Choose a symptom which is the only remaining one in a group. (All the others have been observed already).
MR2. Choose the symptom with the highest membership in partially-matched groups. (This is the symptom, which if it were observed, would give the engineer the most information).
MR3. Resolve conflicts in MR1 and MR2 by taking

the symptom with the shortest time factor.

In fact, because there are many situations, especially early on in an investigation, when even the rules given above do not yield a unique choice, the rules are applied five times (eliminating the previous choice each time) and five symptoms are chosen, giving the engineer an element of choice in his actions.

3.4. 1. Sub-groups.

Although TOAST represents a good model of diagnostic competence, engineers do not always like to work in this rigorous fashion. In particular, some of them are skilled at recognising the emergence of patterns of symptoms before all of the symptoms to confirm a fault have been observed. This heuristic knowledge can be captured in a set of assumption rules of the form:

R2. *if* symptoms S1 to N are present
then assume that SN+1 to N+M are present too (but not observed)

The symptom set (the sub-group) on the left-hand side of the rule is a sub-set of the whole symptom set (the key-group) which implies a faulty sub-unit. The ideal is still to match a whole key-group but a match with a sub-group is considered heuristically adequate to make progress in the investigation. The program ADAPT checks for confirmation when a sub-group is used successfully and can also reject those which are disconfirmed by failure. More importantly, ADAPT "discovers" new sub-groups by applying the following meta-rule:

MR4. if action suggestions resulting from a partial match result in the completion of that match
then insert the symptoms making up the partial match into the data-base as a possible sub-group.

3.1.2. Implementation of rules and meta-rules.

Meta-rules MR1-4 are implemented conventionally through fixed procedures. Since they form the analytic part of the model of competence, there is no need to represent them explicitly. However, rules R1 and R2, when instantiated by particular symptom groupings, need a flexible representation which allows addition and modification of rules as the system grows and changes. Thus all rules in the CRIB data-base are represented as relations between symptom groups and either sub-units as in the case of key-groups (R1), or other groups as with sub-groups (R2). CAFS is an ideal vehicle for supporting such relations.

3.5 Validation, Improvement and Performance.

It was possible to validate CRIB's competence assuming that the correspondence between symptoms and the actions designed to reveal them was accurate and that the fault-finding guide data was not too far from the truth. A relatively inexpert user was asked to be completely guided by CRIB and 'carry out' the actions suggested by the system. If all was well, we would have expected CRIB to suggest, as a final action, the replacement or repair of a terminal sub-unit. This was the case in a large number of 'investigations' carried out (some 200 in all). However, in some cases, especially where the chain of suggested actions was long, the really relevant actions were not displayed soon enough. The reason for this was pinpointed in the scheduling of actions after a partial match (see MR2 in section 3.4. When little information is available to CRIB to partition the symptom groups it tends to spread its net too wide in the search for possibilities. However, by using the filtering facility described in section 3.4, it was possible to resolve at least some of these difficulties. Full detail can be found in [2], where several 'real' faults discovered by an expert engineer are described.

3.5.1. Assessment of performance.

CRIB can certainly help an engineer to find faults on broken computers where these faults have been successfully remedied before. If an engineer does not spot a fault as a known one (whether or not he is using a system such as CRIB), he must fall back upon basic knowledge both about the particular machine he is working on (possibly functional knowledge) and about diagnosis in general. In this case, CRIB, with its complete lack of explicit functional knowledge, cannot help. However, the unknown symptom group will, of course, contain many symptoms which are known and are present in other groups. It is thus possible that the engineer, with a little thought on his part, can locate the true fault. This process has been observed when using CRIB in an unstructured way; ignoring suggestions, putting in unsolicited symptoms etc. Further discussion and a proposed solution to the 'new faults' problem are in [1],

4. SOFTCRIB: An Exercise in Generalization.

If the structures and mechanisms within CRIB can be describd independently from the content of the data which drives it, then the description will be of a production system capable of wider application than computer diagnosis. This line of thinking follows the generation of PUFF [8] and PROSPECTOR [7] and GUIDON [6] from the system architecture of MYCIN (the EMYCIN or Essential-MYCIN concept).

The next most obvious application for CRIB was diagnosis of software faults in a similar fashion to hardware faults. Within ICL, a group a 'dump-crackers' had already gathered data on faults exhibited by a new large-scale operating system called VME/K. Most of the concepts in CRIB carried across to the software domain, albeit with re-interpretation. A hierarchy of attributes describe the different levels of detail within the operating system. These attributes were chosen to partition bugs in a reasonable way and the whole system of attributes forms a paradigmatic representation of the operating system's path-

ology. Attributes which are immediately subordinate to a common attribute can then be characterised as answers to a single question, e.g. What type of dump? is subordinate to What software version?

The remaining correspondences were straightforward. Symptoms remained as symptoms, and key groups of symptoms corresponded to a trace of symptoms recorded on the way to fixing a bug. Actions were classified as either patches or questions which when answered by the user yielded further symptoms of that particular bug. Other less important features of SOFTCRIB concerning interaction with and use of the new system can be found in [1]. Once again the power of a hierarchically guided diagnostic methodology has been demonstrated to be adequate to the task. Moreover, the production system architecture of CRIB enabled the change of function merely through a change of data-base content and a simple re-interpretation of parts of the data structures.

5. Conclusions; What is an Expert System?.

Many workers in knowledge engineering and expert systems have attempted to define the field from several different angles. It has been said that an expert system should be able to explain itself to the user (as does MYCIN for example) just as a human expert can. Claims have also been made for 'natural language' interfacing; being interactive in the form of a dialogue; having incremental data-bases and being adaptive and heuristic in operation. All of these attributes are highly desirable and even necessary in some cases. However, they are all reduced to being incidental window-dressing unless the system does actually possess expertise in its field. Hence the emphasis in this paper on the "before" and "after" phases. Elicitation of expertise is crucial to the initial success of the system; adaptability and a capability for improvement are crucial to its continued success.

The question posed in the title of the paper can now be seen as relevant to tying "before" to "after". It is of little use to attempt a full-scale representation of expertise in the elicitation phase (knowledge engineering is not cognitive modelling) if the resulting system is difficult to modify and hence to improve. It is equally bad to program elaborate schemes for fine-tuning the data-base when its content does not properly represent the desired expertise. The system should thus only capture that part of the expertise which is sure to be useful to all practitioners, and which can be readily modified through straightforward improvement schemes. CRIB partly meets these demands, but its authors now see it as a prototype for something better in the future. Certainly production system architecture has many desirable features, but expert system architectures should vary according to the forms of knowledge needed to capture a particular sort of expertise. Elicitation techniques geared towards these different forms of knowledge are thus vital.

ACKNOWLEDGEMENTS

Initially, it was Andrew Bond, in the team led by Frank George who suggested a system like CRIB. Les Rabbitts of ICL engineering training provided some of the early ideas. Thanks are due to Bob Beakley and Ted Newman of ACTP and to Gerry Piper and Tony James of ICL. The biggest thankyou, however, goes to Tom Addis of ICL RADC for his hard work, encouragement and insights too numerous to mention.

REFERENCES

[I] Addis, T.R. Towards an 'Expert' Diagnostic System. ICL Technical Journal, May:79-105, 1980.

[2] Addis, T.R. and Hartley, R.T. A Fault Finding Aid Using a Content Addressable File Store ICL Research and Advanced Development Centre, Stevenage, U.K. TN 79/3. 1979.

[3] Babb, E. Implementing a Relational Database by means of Specialised Hardware. ACM Transactions on Database Systems 4(1):1-29, 1979.

[4] Bureau of Information Science in cooperation with the Deparment of Trade and Industry. Contract K78B1/314 Computer Fault Finding Methods. Technical Report, May 1973.

[5] Brachman, R.J. What's in a Concept: Structural Foundations of Semantic Networks. International Journal of Man-Machine Studies 9:127-152, 1977.

[6] Clancey, W.J. Tutoring Rules for Guiding a Case Method Dialogue. International Journal of Man-Machine Studies 11:25-49, 1979.

[7] Duda, Richard; Gaschnig, John and Hart, Peter Model Design in the PROSPECTOR Consultant System for Mineral Exploration in Expert Sytems in the Microelectronic Age edited by Donald Michie. Edinburgh University Press, 1979.

[8] Feigenbaum, E.A. Themes and Case Studies of Knowledge Engineering in Expert Systems in the Microelectronic Age edited by Donald Michie. Edinburgh Unversity Press, 1979.

[9] Griffiths A.P. (ed) Knowledge and Belief (Oxford readings in philosophy), OUP, 1967.

[10] Hays-Roth, Frederick; Waterman, D.A. and Lenat, Douglas B. Principles of Pattern-Directed Inference Systems in Pattern-Directed Inference Systems edited by D.A. Waterman and Frederick Hays-Roth. Academic Press, 1978.

[II] Johnson, Leslie Practical Reasoning Man-Computer Studies Group, Brunei University. M-CSG/TR/1, 1979.

[12] McCarthy, John. Epistemologlcal Problems of Artificial Intelligence. Proceedings of 5th. International Joint Conference on Artificial Intelligence. MIT, 1977.

[13] Schubert, L.K. Extending the Power of Semantic Networks Artificial Intelligence 7(2):163-198, 1976.

[14] Sloman, Aaron. The Computer Revolution in Philosophy Harvester press, 1978.

[15] Woozley A.D. Theory of Knowledge: an Introduction, Hutchinson, 1969.