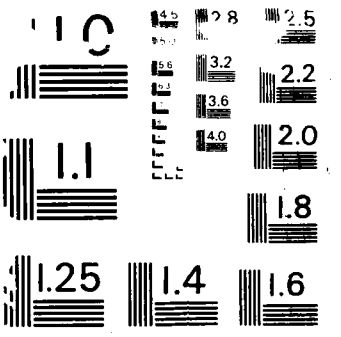
AD-A191 083 REPRESENTATION AND DECISION MECHANISMS IN ARTIFICIAL INTELLIGENCE(U) DUKE UNIV DURHAM NC DEPT OF COMPUTER SCIENCE D W LOVELAND ET AL 28 DEC 87 ARO-21213 7-MA 1/1

UNCLASSIFIED DAAG29-84-K-0072 F/G 12/9 NL



Representation and Decision Mechanisms in Artificial Intelligence

Final Report

Donald W. Loveland
Alan W. Biermann

December 28, 1987

U.S. Army Research Office

ARO Grant DAAG29-84-K-0072

Duke University
Durham, NC 27706

THE VIEW, OPINIONS, AND/OR FINDINGS CONTAINED IN THIS REPORT ARE THOSE OF THE AUTHOR(S) AND SHOULD NOT BE CONSTRUED AS AN OFFICIAL DEPARTMENT OF THE ARMY POSITION, POLICY, OR DECISION, UNLESS SO DESIGNATED BY OTHER DOCUMENTATION.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) ARO 21213.7-MA
6a. NAME OF PERFORMING ORGANIZATION Department of Computer Science Duke University 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION U. S. Army Research Office
6c. ADDRESS (*City, State, and ZIP Code*) Durham, North Carolina, 27706	7b. ADDRESS (*City, State, and ZIP Code*) P. O. Box 12211 Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING/SPONSORING ORGANIZATION U. S. Army Research Office 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DAAG29 - 84 - K - 0072
8c. ADDRESS (*City, State, and ZIP Code*) P. O. Box 12211 Research Triangle Park, NC 27709-2211	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (*Include Security Classification*)
"Representation and Decision Mechanisms in Artificial Intelligence"

12. PERSONAL AUTHOR(S)
D.W. Loveland and A.W. Biermann

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Final Report	FROM 5/84 TO 9/87	87/12/28	11

16. SUPPLEMENTARY NOTATION
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
Representation, learning, adaptive systems, Prolog, testing procedures, expert systems .

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

During the period of this grant, six subprojects were investigated and five subprojects have reports and/or publications at present. The sixth area is the subject of a Ph.D. dissertation expected to be completed in 1988. The areas include: Automatic selection of most efficient programs, determination of confidence factors in expert systems, analysis of errors that learning machines make, real time program synthesis through graph factorization, an extension to Prolog, and finding efficient test-and-treatment procedures.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (*Include Area Code*) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

FINAL REPORT

ARO Grant DAAG29-84-K-0072

This is the final report for Army Research Office (ARO) Grant DAAG29-84-K-0072, for the period May 14, 1984 to September 30, 1987. The funding provided partial support for two senior investigators, Donald W. Loveland and Alan. W. Biermann, and partial support for four students which resulted in three M.S. degrees awarded and an expected Ph.D. degree. (A Senior Programmer was also employed during a critical stage of one project.) This funding provided for a many-faceted exploration of new concepts in the development of learning or adaptive machines and their application to practical system design. It also allowed investigation of a theory of efficient testing procedures with extension to the integration of treatments. It provided funding that permitted examination of the power of the logic programming language **Prolog** and the first pursuit of a system to extend the underlying logic of Prolog while retaining much of the efficiency that has allowed it to be so useful. The funded research in outcome can be viewed as six distinct lines of research which:

(1) show how adaptive software systems can be built to automatically select the best program for doing a task,

(2) show for learning machines general relationships between the sizes of the classes of learnable behaviors and the error rates and learning rates,

(3) give a method for automatically discovering the confidence factors for the knowledge based production rules in expert systems,

(4) develop a synthesis technique for real time computer programs based on the concept of the factorization of a program behavior graph into its control structure and data structure graphs.

(5) initiate the extension of Prolog to permit disjunctive clauses and the classical negation to be used, and

(6) continue the investigation of the test-and-treatment problem where treatments are integrated into a theory of designing efficient (low cost) test procedures.

Each of these lines of pursuit has realized results of its own so it is hard to highlight some results over others. Some results can be useful near-term, such as the determination of confidence factors for expert systems, while others are long term, such as studying the tradeoffs between fast learning machines and large-domain learning machines. This is a particularly interesting contrast because both results deal with automating adaptive behavior; the expert system project recognizes the presently unique ability of the human expert to devise the governing rules relevant to the task domain and "settles" for adjusting the weights for the rule inference strengths relative to the inference network in which the rules are employed. This is itself a non-trivial task, generally too slow to be done by simple algorithms on arbitrary rule sets. Our longer range study just mentioned is particularly important because while many researchers have proposed learning machines of various types, few have discussed general properties that

all learning machines must obey. Here, we show relationships between the size of the learnable class and the learning and error rates.

Another area where we feel that this grant has promoted important work is the extension of the logic programming language Prolog. While "extensions to Prolog" have been known since before Prolog was actually defined, they generally have not allowed a natural programming viewpoint plus a declarative (logical) viewpoint simultaneously, and also have been too slow in processing time to be useful over nontrivial problems. By not trying to extend too far beyond the domain Prolog now processes we feel that we maintain the key properties of Prolog in the extended domain. Although much research and attention to implementation concerns still lie ahead, the basic structure of the extension is now clear.

The following paragraphs describe these six topic areas and papers are attached that include full presentation of results. (Note: Papers noted as *attached to report* have been appended to an original copy sent to our ARO monitor. Other copies of this final report do not have the reports appended; these reports have been deposited with the ARO in standard manner and are available through the ARO document library.)

Adaptive Software: Automatic Selection of the Most Efficient Program. A common problem for practical programmers regards which of several alternative solution methods should be used in a given situation. The programmer may know of several alternative data representations and not know which to employ; the programmer may also know of more than one way to code the algorithm once the data structure is selected. Furthermore, the various alternatives may lead to widely varying performance levels so one should make an informed decision.

Yet the problem of which to use is not easily solved. The programmer may need to study the application carefully, to read relevant literature on the issue, and to run simulations of the computation to discover the most efficient implementation. On the other hand economic considerations require that the decision be made quickly so that the coding can be done promptly and attention can be turned to other issues.

This project proposes a pragmatic methodology for addressing this dilemma. If several alternative ways exist to solve a problem, the suggestion is that all of them be coded and loaded simultaneously into the application along with a monitor that will collect data on their relative performances and eventually select the most desirable one. The advantages of this approach are that

(1) the best selection will be reliably made,

(2) the decision will be based upon actual execution of existing code on the actual data and machine for the application, and

(3) the programmer's time required for analysis will be minimized.

A methodology for designing such a monitor is given in the attachment [5] where it is assumed that two programs P_1 and P_2 are available for doing a given computation which must be executed L times. On each execution, the cost of running program P_i, will be assumed to be c_{ij} with probability p_{ij}, $i=1,2$, and $j=1,2,...,m$, respectively. However, the p_{ij}'s are unknown, so it is not known which program will have lower

expected cost.

It is desired to have a monitor in control which will, during the first few of the L computations, select which program to run, note its execution time and decide what to do next. It may select either program to run on the next computation or it may decide to run one of the programs for the rest of the L executions. The paper [5] gives an optimum strategy for the monitor which achieves minimum total expected cost for the L computations assuming the cost of one monitor decision is g.

On The Errors That Learning Machines Will Make. Associated with each learning system there is a class of learnable behaviors. If the target behavior to be acquired is in the learnable class, it will be learned perfectly. If it is outside that class, the machine will only be able to acquire a behavior that approximates the target and it will always make errors. It is desirable for a learning machine to have a large learnable class to maximize the chances of acquiring the unknown behavior and to minimize the expected error when only an approximation is possible. However, it is also desirable to have a small learnable class so that learning can be achieved rapidly. Thus the design of learning machines involves selecting a position on the spectrum: minimum error and slow learning time versus larger error and faster learning time.

Several types of learning systems are examined from the point of view of the above parameters including signature tables, linear system models, and conjunctive normal form expression based systems. These studies lead to the concept of an "optimum" machine which spreads its learnable behaviors across the behavior space in a manner to minimize the expected error. Two approximations to such optimum machines are presented and their behaviors are compared to the more traditional learning machines. The details of this study appear in [2].

Determining Confidence Factors for Expert Systems. Expert systems have been used successfully in recent years to solve a variety of application problems related to medical diagnosis, chemical spectroscopy, geological analysis, and other domains. Unfortunately, immense problems arise in the construction of such systems preventing their wider use. In fact, one or more computer scientists or "knowledge engineers" must spend many hours over a period of months or years interviewing a specialist in the application domain and coding the reasoning processes sufficiently so that a machine can carry it out. This project aims at partially automating the process of such expert system construction.

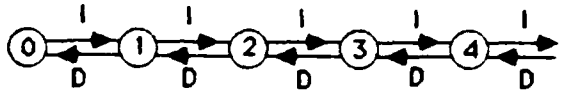
Typical expert systems are constructed with a set of production rules. Usually these rules have two components, the logical operator which defines the manner in which new information is derived from known facts and the confidence factor which assigns a degree of certainty to each conclusion. Our assumption is that the expert must provide the logical operators but we have developed a methodology for automatically computing the confidence factors. The methodology requires only that samples of input facts and their associated conclusions with confidence levels be given. From, these examples one can compute the appropriate confidence factors for the rules used to derive the conclusions from the input facts.

Specifically, we present in [10] algorithms to instantiate confidence factors in expert systems given that the logical operators are already instantiated. We show that nk carefully selected input-output behaviors are sufficient to characterize a system with n inputs and a finite set of k confidence factors. If ar oracle is allowed, we present an algorithm to synthesize the confidence factors in $O(n^2)$ time with nk calls to the oracle.

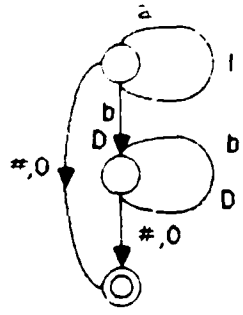
If the confidence factors are real numbers, then $2n$ input-output behaviors are sufficient to characterize the system, and we can find the correct confidence factors in $O(n)$ time with $2n$ calls to the oracle.

We also consider the case where we must satisfy an arbitrary set of input-output behaviors by instantiating the confidence factors. We present an algorithm to solve the problem in this case, but also show that this problem is NP-complete by transforming the satisfiability problem to it.

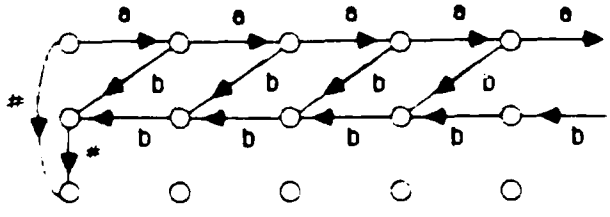
Real Time Program Synthesis Through Graph Factorization. This research begins with an observation related to real time programs: if the infinite state graph representing a data structure and the finite state graph representing its control structure are combined with a properly defined cross product operation, the result will be a graph representing all of its possible behaviors. For example, a counter can be represented by the graph



where I is the increment instruction and D is the decrement instruction. If we wish to program an acceptor for the language $a^i b^i \#$, $i = 0,1,2,3,\ldots$ then a proper control structure which uses the above counter is



where a (or b) means read a (or b) from the input, I and D mean increment and decrement the counter and # ,0 means read a # from the input and a 0 on the counter. The cross product of these two graphs is the "behavior graph", a representation of all possible behaviors the system can have.



In this graph, the upper leftmost node corresponds to the zero state of the counter and the initial state of the control. The state to its right corresponds to the 1 state of the counter and the initial state of the control, and so forth.

These observations lead one to a number of questions related to learning theory. Suppose one is given a behavior graph such as the one above and is asked to algorithmically discover the data structure and control structure required to achieve that behavior. This corresponds to having an oracle available to answer questions about a target behavior and to the automatic discovery of the data structure and control structure required to achieve that behavior in general. The observations also suggest a methodology for doing the synthesis. It is through graph factorization.

The research questions for the project thus become:

(1) Is there a way to diagnose which data structure from a dictionary of available data structures (counters, stack, queue, etc.) should be used in the synthesis?

(2) Find an algorithm which, given the behavior graph for a computation and the data structure, will factor the behavior graph to find the control structure.

(3) Given the fact that general algorithms for the problem (2) appear to require exponential time to complete a computation, find subclasses of real time programs that can be synthesized in polynomial time.

This project is ongoing and forms the doctoral dissertation project for Mr. Amr Fahmy. A data structure diagnosis requires a bound on the number of states in the control structure and uses the rate of growth of the behavior graph as a function of distance from the initial node to estimate the needed data structure. Several synthesis (graph factorization) algorithms have been developed, tested, and their properties analyzed. One powerful synthesis strategy uses the technique of identifying all sets of nodes in the behavior graph that have similar topological environments and merging these nodes to yield the control structure. Attempts are being made to characterize a class of programs which have such a low cost factorization scheme. Detailed results will be presented in late 1988.

Near-Horn Prolog. Somewhat over a year ago we suddenly saw an approach to a problem that has been of interest to us for some time. The area of concern is logic programming and the potential importance of the idea, if successful in realization and effect on the area, warranted redirection of our research efforts to this question. To a large extent this redirection of effort has occurred for one of the investigators of this grant (Loveland). Our concern is with minimal extensions to Prolog, the primary logic programming language in use today. Our intent is to increase the scope of the language while preserving as strongly as possible the speed of execution, a big factor in the present success of the programming language from a pragmatic point of view. We now have a precise formulation of our new approach and have presented this first formulation to the Logic Programming community in [9], a copy of which is attached. There is much left to do but the basic idea seems to have validity.

Our approach in this work is quite pragmatic, building on the experience of the success Prolog has had in the world of computing in spite of many drawbacks (but

obviously because of many advantages). The idea is to give up some theoretical virtues to preserve a practical virtue that has proven its importance. Namely, we promote a complete first-order proof procedure that is not in general as powerful as some that already exist but appears to be much better in a certain domain close to the domain Prolog functions on. We now clarify this. Prolog deals with a subset of logic known as Horn clause logic, basically a positive implication logic. It is safe to say that Prolog's success is due to two things, the amazing number of useful problems that can be formulated within the Horn clause logic and the speed with which these problems can be executed. However, disjunctive facts and conclusions (e.g., P(a) OR P(b)) cannot be handled by Prolog. Moreover, negative information is not naturally processed within the Horn clause logic and this capability is needed, very often for database systems. Presently the device of "negation as failure" is used. The disadvantage of negation as failure is that it is built on the principle that anything not provable is false, and its advantage is that its execution is achievable within the Prolog framework (but for ground statements only) and so benefits from the speed of execution that Prolog possesses. Complete first-order theorem provers have proven to be too slow to be useful in the "real world" to drive logic programming languages.

The new procedure is called *nH-Prolog* for near-Horn Prolog. The relevance of the term *near-Horn* is that the procedure is particularly attractive for processing clause sets with few negations or disjunctive conclusions. This is because the speed of processing is a function of the "distance" of the clause set from a Horn set. The ideal is that the processing speed fall off only in proportion to the distance from a Horn set, figured by the number of negations in a Prolog-type input format, because many practical problems have relatively few negations in an otherwise legal Horn clause set. The nH-Prolog system has this characteristic. The nH-Prolog procedure is indeed a complete first-order proof procedure as mentioned above (when certain obvious expediences of Prolog are altered). Although almost surely slower than some known theorem provers on "badly" non-Horn clause sets, such as is characteristic of logic puzzles, our first implementation shows that indeed we realize excellent speed on the near-Horn clause sets that we have tried. A conference paper discussing the first implementation is now being written [11] and, when completed, will be forwarded to ARO because some of the ARO funding partially supported the implementation effort.

We are encouraged by the results to date and work continues on the design and implementation of nH-Prolog. There are many design and implementation questions yet to address, which we hope further funding will allow us to pursue, but we are still of the opinion that our efforts are well directed in pursuing this Prolog extension.

Testing procedures. For the past several years we have studied the design and analysis of efficient testing procedures in relatively unstructured environments. The problem that we have been studying recently is the integration of tests and treatments. This has resulted in a summary conference presentation [7] and a completed M.S. thesis [6], the former attached and the latter submitted to the ARO in standard reporting manner. A second problem involves methods for quick detection of multiple objects given a set of tests for doing that job. Both problems concern finding low cost decision trees where cost is an expected cost using given costs for tests and *a priori* estimates

regarding the probability that an object is a (the) desired object. We will briefly discuss the problem that received most attention and then comment on the second problem.

We are studying a problem that we call the *test and treatment problem*, so-named because of the integration of treatments within the testing problem itself. A related problem studied by many people has been called the *diagnosis* problem. We state it here in the form studied by computer scientists although a more general form has been intensely studied by statisticians. A doctor-patient model is used for concreteness of presentation and because of its intuitive simplicity. Suppose that a patient appears at a doctor's office with an unknown disease. By inspection the doctor can reduce the diagnosis problem to a set of n diseases and give some weight as to the likelyhood of each candidate diagnosis being the correct diagnosis. For simplicity we assume that there is only one disease to discover. (The multiple-disease problem is the alternate problem we mentioned above.) Refining this initial diagnosis requires non-trivial tests and so the doctor wishes to assess the order in which the tests are done. The tests have costs associated with them; the costs reflect not only economic costs but patient risk, discomfort, etc. The problem is to produce the lowest possible (optimal) cost test procedure, using expected cost as the cost measure. For the case when all tests have unit cost computer scientists have determined a number of facts about this problem already. We grossly oversimplify here to give a quick general picture of the situation. If every test is available then there is a fast algorithm for finding the optimal test procedure, and it is essentially the Huffman coding procedure. When some tests are unavailable for use (the *incomplete test case)* then the problem is NP-hard, i.e. all procedures for finding the optimal decision tree may take exponential computation time in the length of the problem presentation. Because in practice it is impossible to find the optimal tree in the incomplete test case we seek good but fast approximations. Statisticians use the maximum likelyhood estimator, which degenerates to the binary splitting algorithm when much of the probabilistic considerations (such as unreliable tests) are dropped to permit deeper analysis of performance. Analysis shows that the binary spitting algorithm can be extremely bad in the general incomplete test case, although if the full class of single-disease tests exist then the performance greatly improves but is still not too good. (Some of these results were ours and published prior to receiving this ARO grant.) Even when all the *a priori* probabilities are equal the binary splitting algorithm can be quite poor.

While working on the diagnosis problem just described we realized that for most real-life situations the problem missed a key point. One does not often wish to diagnose to isolation for the true disease just to know the disease; rather, one seeks that intermediate knowledge in order to treat the patient. But often it is cheaper to prescribe a treatment before testing to isolation. The classic example is the doctor's refrain, "take two aspirin and call me in the morning". We seek a unified theory of test and treatment and have been at work on this problem for several years. Surprisingly, we have not found previous work in the literature and asking statisticians about the problem has not yielded any analytical work from that camp either. Recently, there have been suggestions that the area of decision support theory has related material but the actual connections are unclear.

Because of the superposing of treatment possibilities on top of the testing problem the test-and-treatment problem is a much more difficult problem in several ways. Whereas the diagnosis problem has an $O(n(\log n))$ running time for the complete test problem, the test-and-treatment problem is NP-hard even for the complete test-and-treatment problem. Also the problem is much more complex to analyze. We chose to work on an important special case to understand the mechanisms at work, and have some interesting results for that special case. More importantly, we have learned much about the behavior of test-and-treatment decision trees. We chose to take the special cost case of unit cost for all tests (which copies the cost simplification used in the diagnosis problem) and have the cost of the treatment proportional to the power of the treatment, where the proportionality constant is a parameter. Here treatment *power* is the sum of the *a priori* probabilities of the objects (diseases) that are successfully treated by that treatment. (All treatments are reliable and unambiguous in that the object either is or is not successfully treated.) Moreover, we chose the complete test-and-treatment case with equal *a priori* weights and with a simple approximation algorithm, one that yields a complete binary tree of arbitrary level. The general problem and the specific results are outlined in a talk given at the Fourth Army Conference [7]; the report is attached. Proofs are omitted from the talk.

Why was the above cost model chosen? Preliminary investigation strongly suggested that if the treatments have widely differing cost/power ratios then an optimal or near-optimal procedure is found by merely selecting treatments by ordering them low-to-high with respect to the cost/power ratio. The difficulty arises when the cost/power ratios are equal or nearly so. Indeed, this much is quite intuitive. What is not intuitive is to how to proceed when the cost/power ratios are the same, which became the basis for our first study. Again, we emphasize that the study named above was chosen for its simplicity to allow us to understand the problems we faced.

The M.S. thesis of Paul Lanzkron contains some analytical results along the line just described. However, it also contains some experimental results where the problem statement was free of the constraints we needed to obtain analytic results. In particular, we let the test and treatment costs be arbitrary and the *a priori* probabilities also be arbitrary. We chose several approximation algorithms which our analytic work had led us to believe would yield reasonably good approximations in many cases. What was surprising was how good they were given the simplicity of the algorithms relative to the difficulty of characterizing the optimal solution. We hope to publish these results in a journal in the biomedical computing area so that the results become known outside the domain of academic computer science.

Study of the multiple faulty object problem has been initiated, and an appropriate model defined. A very restricted subproblem has been chosen for study but no definite results have been obtained yet, primarily because our energies were diverted to the test-and-treatment problem. This diversion occurred partly for the pragmatic reasons that we had early success with establishing some key results regarding test-and-treatment problems and then a student, Paul Lanzkron, developed interest in this problem and so attention remained focussed there. Moreover, the multiple faulty object problem seems most meaningful when the *a priori* probabilities are statistically independent. Thus the model is quite different in detail and we felt that we wished to pursue the test-and-

treatment problem that people seemed to be ignoring in spite of its importance. The multiple test problem is an important one and we hope to pursue it in the not-too-distant future. Our results to date thus are very preliminary and no publication in this subarea is seen in the near future.

PARTICIPATING PERSONNEL

Student	*Degrees Earned*
Amr Fahmy	M.A., Ph.D.(in progress)
Barry Koster	
Paul Lanzkron	M.S.
Albert Nigrin	M.A.

Senior Programmer

K.C. Gilbert

Senior Investigators

Alan W. Biermann

Donald W. Loveland

PUBLICATIONS AND REPORTS supported by this grant.

[1] Biermann, A.W. Fundamental Mechanisms in Machine Learning and Inductive Inference. In *Fundamentals of Artificial Intelligence*, Eds. W. Bibel and Ph. Jorrand, Springer-Verlag, 1985.

[2] Biermann, A.W., Gilbert, K.C., Fahmy, A., and Koster, B. On the Errors That Learning Machines Will Make. Report (submitted for publication), November, 1986. (Abstract appears in the transactions of the Fourth Army Conference on Applied Mathematics and Computing. Cornell University, May, 1986).

[3] Biermann, A.W. Fundamental Mechanisms in Machine Learning and Inductive Inference, Part 2. In *Proceedings of the Advanced Conference on Artificial Intelligence*, Ed. R. Nossum, Springer-Verlag, 1987.

[4] Fahmy, A.F. A Self Modifying Monitor for Algorithm Selection. Thesis, Duke University, June 1984.

[5] Fahmy, A.F., and Biermann, A.W. Adaptive Software: Automatic Selection of the Most Efficient Alternative Program. Report (submitted for publication), January, 1986.

[6] Lanzkron, P.J. Results on the test-and-treatment problem. M.S. Thesis, Computer Science Dept., Duke Univ., Durham, NC, May, 1987, 137pp.

[7] Loveland, D.W. Introducing treatments into test procedures. *Proc. of the Fourth Army Conf. on Appl. Math and Computing*. Ithaca NY, May, 1986.

[8] Loveland, D.W. Automated theorem proving: mapping logic into AI. *Proc of the Int'l. Sympos. on Methodologies for Intell. Systems*. Knoxville, TE., Oct. 1986.

[9] Loveland, D.W. Near-Horn Prolog. *Proc. of the Fourth Int'l. Conf on Logic Programming*. Melbourne, May, 1987.

[10] Nigrin, A.L. Determining Confidence Factors for Expert Systems. Report (submitted for publication), Sept. 1987.

[11] Smith, B.T. and D.W. Loveland. An implementation of nH-Prolog. In preparation.

END
DATE
FILMED
5-88
DTIC