ELECTRICAL DESIGN
A PROBLEM FOR ARTIFICIAL INTELLIGENCE RESEARCH

Gerald Jay Sueeaan
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139, USA

Abstract:

This raport outlines the problem of intelligent failure recovery in • problem-solver for electrical design. We want our problem solver to learn as much as it can from its mistakes. Thus we cast the engineering design process in terms of Problem Solving by Debugging Almost-Right Plans, a paradigm for automatic problem solving based on the belief that creation and removal of "bugs" is an unavoidable part of the process of solving a complex problem. The process of localization and removal of bugs called for by the PSBOARP theory requires a*n* approach to engineering analysis in which every result has a justification which describes the exact set of assumptions it depends upon. We have developed a program based on Analysis by Propagation of Constraints which can explain the basis of its deductions. In addition to being useful to a PSBDARP designer, these justifications are used in Dependency-Directed Backtracking to limit the combinatorial search in the analysis routines.

Although the research we will describe is explicitly ebout electrical circuits, we believe that similar principles and methods ere employed by other kinds of engineers, including computer programmers.

Introduction:

Engineers combine, analyze, debug, and explain structures in the course of design. They decide how simple structures may be combined to achieve particular goals. They can predict the behavior of complex structures by combining the behaviors of the substructures out of which they were formed. This analysis is critical for debugging plausible designs which do not quite work, for constraining the possible design decisions, and for ruling out unfeasible plans. Finally, an engineer must be able to explain the devices which he has designed. An explanation is often a description of how the behavior of the composite device can be attributed to the combined behaviors of its parts. The ability to explain is crucial to analysis and design. It is much easier to analyze a system if we know the intended operation of the parts.

This paper outlines our project to construct an electrical circuit designer program as part of an effort to understand the fundamental mechanisms involved in reasoning about complex, deliberately constructed systems. Parts of this program already exist, other parts are being developed and others ere still in the planning stage. Essential ideas from the recent theses of Allen Brown on the localization of failures in radio circuits <Brown 1975> and Drew McDermott on a rule-based system of hierarchical design <McDermott 1976> are being incorporated into this project.

A Theory of the Engineering Design Process:

Innumerable hours can be spent tracking down a "bug" in a computer program, an electronic device, or a mathematical proof. At such times it may seem that a bug is at best a nuisance and at worst a disaster. We believe that many bugs are just manifestations of powerful strategies of creative thinking — that creation and removal of bugs are necessary steps in the normal process of solving a complex problem. Following the work of Poly a <Polya 1962>, recent research <Fahlman 1973> <Sussman 1973> <Gold$tein 1974> predicated on this belief has resulted in the development of a paradigm for problem solving which we cell Problem Solving by Debugging Almost-Right Plans (PSBDARP). We believe that the PSBDARP theory is a good foundation for building expert problem-solving systems for such diverse kinds of engineering as circuit design and computer programming.

The PSBDARP Theory:

Figure 1 displays the structure of a PSBDARP problem solver. When the problem solver is given a problem it first checks its Answer Library to determine if there is an answer available whose pattern of applicability matches the problem

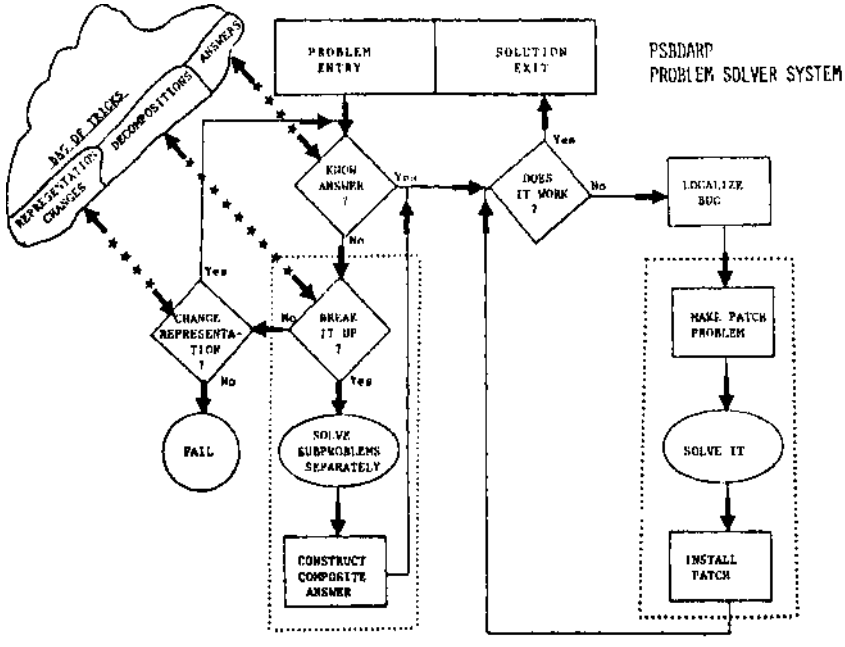


FIGURE 1

statement. If so, the proposed answer is tested make sure that it really works and if it passes the test it is returned as the answer to the problem posed. But suppose the answer is not immediately available. The problem solver next examines a set of problem decompositions to see if any are appropriate for breaking the problem into more manageable chunks. If so, the problem solver remembers the decomposition rule chosen and recursively calls itself to solve each subproblem separately. If this is possible, the solutions returned *are* combined according to the decomposition rule used to break the problem up and the resulting proposal is sent off to be tested. If there are no decomposition rules available which match the problem statement, the problem solver next checks to see if there *are* any changes of representation which can be applied to the problem statement to put it into a form more amenable to solution. If so, the problem is considered in terms of the alternate representation. If no representation changes are appropriate, the problem solver has failed on this problem and reports its failure. A failure may cause backtracking and search.

Suppose that a composite solution is eventually proposed and tested. If it is found to work it is returned as the answer, but often the proposal has a bug. A bug may manifest by a contradiction among the constraints of the modules which are the solutions to the subproblems. The composite solution is also analyzed to see if it actually achieves the goal. If there is a bug the next step is to localize the cause of the failure. Since the solution is a composite of correct solutions to subproblems, the bug must be the result of some unanticipated interaction between the parts of the proposed solution. In any case the problem solver must construct a subproblem whose solution would fix the bug. This problem is then solved (by a recursive call to the problem solver) and the resulting patch is installed in the proposed solution. The corrected solution must then be retested against the original criteria.

Why are there bugs?

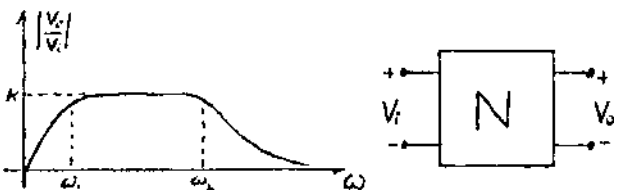
One might imagine a problem-solver based on Figure 1 which produced only correct solutions to problems -- that is, one in which the question "Does it work?" is always answered "Yes". The. problem with this idea is that a crucial part of the problem-solving strategy is the decomposition of problems into presumably independent subproblems. There is no guarantee that this is possible in general, but even when it is not possible, there are often general strategies for approximating a solution to a problem by composing the solutions to almost independent subproblems. Often one can make progress on the solution to a hard problem by considering the solution of a simplified version of the problem which is similar in some essential aspect to the original one but which differs from it in detail. But even in those cases where a decomposition into completely independent subproblems is possible, it is not always feasible. In order to be sure that the solutions to the subproblems *are* really independent it is necessary to understand the problem and the possible implementations of subsolutions so completely that one must effectively solve the entire problem before choosing the correct decomposition. This compromises the decomposition strategy. Another difficulty is that in order to allow "perfect" solutions, the decompositions and possible answers to problems must be specified more precisely. This leads to a proliferation of stored answers and decompositions which differ in only some minor aspect and thus hide the power of generalization.

Superficially, PSBDARP is a kind of Means-Ends analysis <Ernst & Newell 1969> but it is not profitable to merge the concepts. In Means-Ends analysis the problem solver considers the current state of the problem solution and the goal being approached and attempts to apply an operator which will move the solution in the direction of the goal. Often the operator will decompose the problem Into subproblems which can be achieved separately, to be combined into a solution of the overall problem. At this point the PSBDARP philosophy diverges. In Means-Ends analysis the process Is now iterated. In debugging, the original goal may be ignored because many bugs manifest in terms of destructive interactions among the solutions of subprobiems.

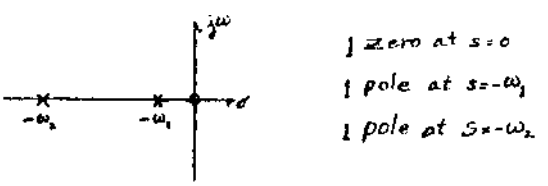
In PSBDARP there is a specific phase of the solution process where debugging knowledge is applied. This knowledge is relatively domain independent and is concerned with notions of causality, teleology and simultaneous constraints. The debugging phase is far more concerned with the structure of the plan produced by the decomposition phase than it is with the goal that evoked the plan. For example, localization of a bug in an electrical circuit or in a computer program may involve such strategies as "tracing" — examination of the conditions at various module boundaries to determine how the expected conditions compare with those that actually occur.

An Example of Synthesis:

Consider a concrete problem of engineering synthesis. Suppose we want an electrical network with a specified system function; perhaps a network having a voltage-transfer ratio whose magnitude varies with frequency as follows:



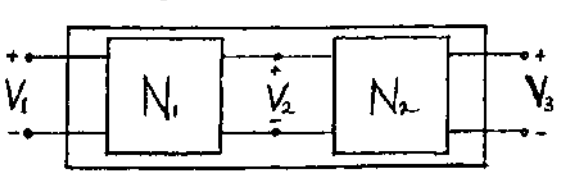
We recognize that the encoding of these requirements is not obvious — part of the research is to determine appropriate languages for such description. The designer first checks the bag of tricks for a plan fragment (an answer or decomposition) whose pattern of applicability matches the goal. (An expert engineer would probably have an answer on tap for so simple a problem.) "Matches" is a rather complex idea — features must be extracted such as the "flat response" between wl and w2, the fall off at frequencies above w2 and below w1, the positions of the "elbows", etc. In this case we assume that the designer does not have a plan fragment for synthesis of the required network, so it has to look for a transformation of the problem. (In McDermott's terminology, we enter the "Rephrasing Protocol".) In this case, there is a good transformation available ~ from a magnitude graph to a pole - zero plot. (This is an algorithmic transformation which requires careful measurement of the parameters of the magnitude plot.)
We get:



This representation is the same as the assumption that the system function has the algebraic form:

$$\frac{V_o}{V_i} = K\,\frac{s}{(s+\omega_1)(s+\omega_2)}$$

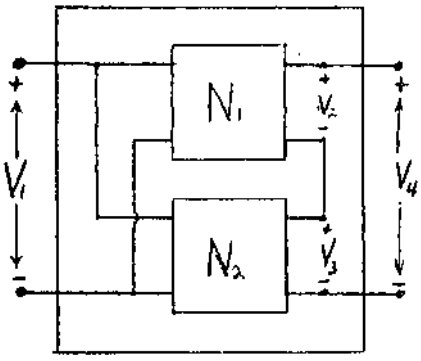
This algebraic expression matches the pattern of applicability of one of our plan fragments -- the cascade plan:



Its pattern of applicability is:

$$H(s) = \frac{V_3}{V_1} = \frac{V_3}{V_2} \cdot \frac{V_2}{V_1} = H_1(s) \cdot H_2(s)$$

Notice that this plan fragment it more generally applicable than just in the domain of electrical circuits. It is appropriate to any domain in which the "stuff" being manipulated (in this case signal represented as voltages) "flows* from process to process. Note that this plan fragment assumes that the flow is unilateral; we will see that this causes a bug! The system hes other generalized decomposition rules as well. For example, if the system function hed been e sum of terms, we could use a different plan for decomposition:



$$V_4 = V_2 + V_3$$

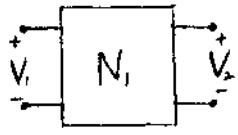
$$\frac{V_4}{V_1} = \frac{V_2}{V_1} + \frac{V_3}{V_1}$$

$$H = H_1 + H_2$$

In fact, if our "bag of tricks" includes some techniques of elgebreic manipulation we can turn our product into a sum by a partial-fraction expansion.

We next try to expand and instantiate the plan fragment's parts. We are forced to solve 2 subproblems:

$$\frac{V_2}{V_1} = \frac{s}{s+\omega_1}$$

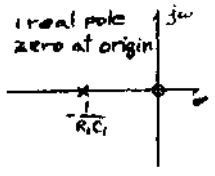


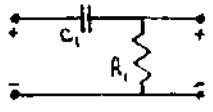
$$\frac{V_3}{V_2} = \frac{1}{s+\omega_2}$$

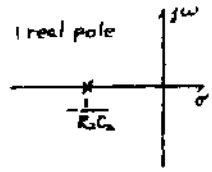
(How will our system Know to break it up this way? — It won't particularly, in that this depends upon how the algebraic expression matcher works. But in any case, if one decomposition fails to be realizable we can expect to back up and try another.)

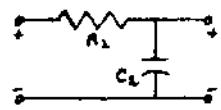
We also see that we have conveniently forgotten about K, which was deferred for later because it is not a constraint. We know that it is always easy to make a scaler if the problem specification does not require that the result be passive.

Next, we recursively attempt to instantiate the subproblems. In this case, we have (at least) two matches in the answer librery. The voltage transfer ratios with one real pole or with one reel pole and a zero at the origin can be reelized as resistor-capacitor voltage dividers.

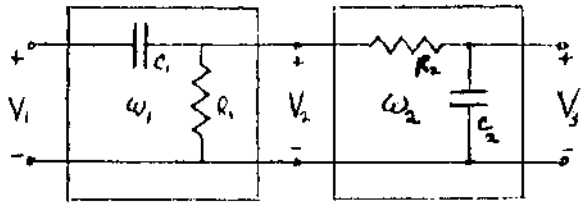








Thus we expand our plan to get:

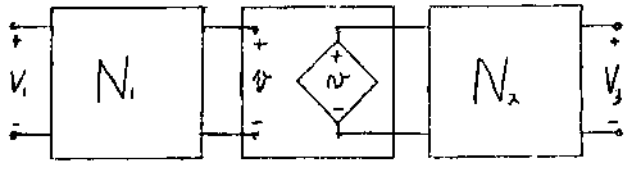


The problem solver must now check the proposed result. Here circuit analysis is employed. If the problem were one of program design, we might run the proposed program in CAREFUL mode <Sussman 1973>; or we might try to Meta-evaluate the program <Hewitt ft Smith 1975>, or even prove that the program is correct.

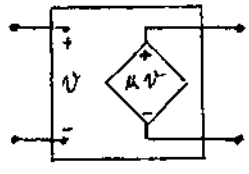
***

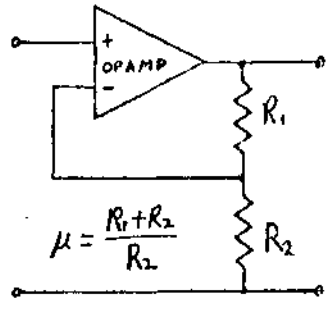
In this case analysis discovers a bug. Our analysis of NI as a voltage divider was contingent on the assumption that no current would be drawn from the midpoint of the divider. Our analysis of N2 was contingent on it drawing a significant current. These assumptions are contradicted by connecting the output of NI to the input of N2. This contradiction is apparent from local evidence in the structure of the proposed solution independent of the goal of the overall circuit. We have caught *ttr* unanticipated destructive interaction between the subproblem solution modules. This particular kind of bug, called loading is common and should be considered whenever ports are connected together.

The statement of this bug — that we have a port voltage which we cant draw a current from, connected to a port which wants to draw a current at that voltage -- can easily be turned into a statement of a problem whose solution is an appropriate patch for this bug. Wishful thinking tells us that if we hed e voltage-controlled voltage source inserted between NI end N2 everything would be OK.

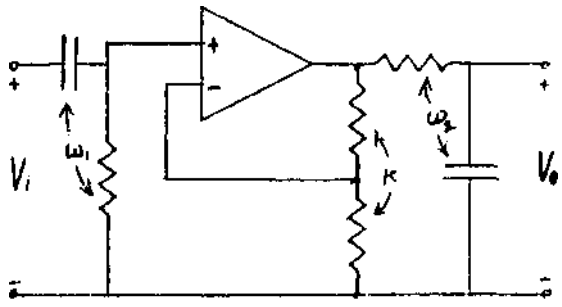


This matches the scaler plan from the bag of tricks:





We now have e good place to absorb our constant K, thus solving the problem given:



Suppose we were using pure means-ends analysis rather then a debugging strategy. At point *** above we would then analyze our current circuit and compere it with the result we expected (the goat). In this case, analysis leads to the following result:

$$\frac{V_2}{V_1} = \frac{R_1C_1 s}{R_1C_1R_2C_2s^2 + (R_1C_1 + R_1C_2 + R_2C_2)s + 1}$$

But this is not the result we were hoping for. We expected:

$$\frac{V_2}{V_1} = \frac{R_1C_1 s}{R_1C_1R_2C_2s^2 + (R_1C_1 + R_2C_2)s + 1}$$

The difference between the answer we got and the answer we wanted is just that there is an extra term (R_1C_2)in the second coefficient of the denominator. Why is this term present? Allen Brown <Brown 1975> has developed methods for localizing bugs using causal and tepeological reasoning about a circuit but it is certainly a more difficult problem to pursue this path than the one we have.

The Propagation Theory of Engineering Analysis:

It is important that the bug localization process have access to the assumptions on which the bug manifestation is based. This depends upon analysis being able to explain its answers. In this section we will describe progress that has been made on an analysis program that reasons qualitatively about circuits and can explain its results.

As part of the development of the PSBDARP circuit designer, we have developed EL, a new Kind of electrical network enalysis program <Sussman ft Staliman 1975>. The literature is full of powerful and useful circuit analysis systems which implement the formal methods. What is novel about this program it its rule-based approach to network analysis and its consequent ability to explain the basis of its deductions.

EL is implemented in ARS (Antecedent Reasoning System), a problem-solving language which implements rules as demons with multiple patterns of invocation monitoring an associative data base <Stallman & Sussman 1976>. It performs ell deductions in an antecedent manner, threading the deduced facts with Justifications which mention the antecedent facts used end the rule of inference applied. These justifications may be examined by the user to gain insight into the operation of the system of rules as they epply to a problem. The same justifications are employed by the system to determine the currently active data-base context for reasoning in hypothetical situations. Justifications are also used in the analysis of blind alleys to extract information which will limit future search.

EL is a set of ARS rules for electronic circuit analysis. This set of rules encodes familiar approximations to physical laws such as Kirchoff's laws and Ohm's law as well as models for more complex devices such as transistors. Facts, which may be given or deduced, represent data such as the circuit topology, device parameters, voltages and currents. The antecedent reasoning of ARS gives analysis by EL a "cetch-es-catch-can" flavor suggestive of the behavior of a circuit expert. The justifications prepared by ARS allow an EL user to examine the basis of its conclusions. This is useful in understanding the operation of the circuit as well as in debugging the EL rules. For example, a device parameter not mentioned in the derivation of a voltage value has no part in determining that value. If a user changes some part of the circuit specification (a device parameter or an imposed voltage or current), only those facts depending on the changed fact need be "forgotten" and re-deduced, so small changes in the circuit may need only a small amount of new analysis. Finally, the search-limiting combinatorial methods supplied by ARS lead to efficient analysis of circuits with piecewise-linear models.

The style of analysis performed by EL, which we call the method of propagation of constraints, requires the introduction and manipulation of some symbolic quantities. Though the system has routines for symbolic algebra, they can handle only linear relationships. Nonlinear devices such as transistors are represented by piecewise-linear models that cannot be used symbolically; they can be applied only after one has guessed a particular operating region for each nonlinear device in the circuit. Trial and error can find the right regions, but this method of assumed states is potentially combinatorially explosive. ARS supplies dependency-directed backtracking, a scheme which limits the search as follows: The system notes a contradiction when it attempts to solve an impossible algebraic relationship, or when discovers that a device's operating point is not within the possible range for its assumed region. The antecedents of the contradictory facts are scanned to find which nonlinear device state guesses (more generally, which backtrackable choicepoints) are relevant) ARS *nenr* tries that combination of guesses again. A short list of relevant choicepoints eliminates from consideration a large number of combinations of answers to all the other (irrelevant) choices. The fact that the set of assumptions leading to the contradiction is inconsistent is summarized and recorded with antecedents being that part of the support of the contradiction which are independent of the assumptions. These summaries are examined whenever a choice has to be made, thus preventing rechoosing of an inconsistent set of assumptions. Thus the justifications (or dependency records) are used to extract and retain more information from each contradiction than a chronological backtracking system. A chronological backtracking system would often have to try many more combinations, each time wasting much labor rediscovering the original contradiction.

Jon Doyle is now engaged in further research on the uses of dependency information in the control of reasoning <Doyle 1976>. *1* De Kleer, Jon Doyle, Guy Steele and this author have been developing an even more powerful rule-besed language we call AMORD in which the EL rules can be expressed in e more hierarchical form.

History of this Project and Relation to other work:

The PSBDARP theory of design has many antecedents. The idea of successive refinement of plans appears as a key dogma of "Structured Programming" <Dijkstra 1970> <Wirth 1971> <Dahl et el 1972>, although it also appears in the Artificial Intelligence problem-solving literature. The idea of relaxation of a hierarchy of constraints comes from <Freeman ft Newell 1971> There are also versions of GPS <Ernst ft Newell 1969> which were purported to do reasoning in a hierarchy of

abstraction spaces. ABSTRIPS <Sacerdoti 1973> showed how refinement of abstract plans could be used to guide a problem solver past problems which would otherwise be combinatorially explosive. Recently the NOAH system <Sacerdoti 1975> has developed this idea to great depth. The major difference we have with successive refinement is our emphasis on engineering analysis and debugging. We are sure that it is impossible to build systems which can deal with complex real world problems without making and removing bugs. PSBOARP is a descendent of the HACKER <Sussman 1973> and MYCROFT <Goldstein 1974> debugging systems. One might consider that GPS already embodied the idea of debugging in that one may take a problem solver with a debugging strategy to be a special case of a problem solver which first attempts to eliminate the main difference between the given and the goal and then reevaluates the situation after a step. This is true in principle, but GPS was *never* used for debugging. Poly a <Polya 1945> had developed a theory of problem solving which included debugging but GPS only captured the reduction and rephrasing aspects of Polya's theory.

Recently Allen L Brown Jr. finished a PhD thesis at MIT <Brown 1975> which explored the use of causal and teleologies! reasoning in the troubleshooting of complex electrical systems. In this thesis Brown developed a set of linguistic conventions for the representation of the plan of a complex, hierarchically-structured system. Brown's methods depend on, and inspired the construction of the EL analysis system <Sussman & Stallman 1975> which uses constraint propagation and can explain how a result depends on assumptions. Brown needed analysis by propagation of constraints to predict the consequences of a hypothesized fault in a component. These consequences are compared with the measured values as a test of the fault theory. The explanations are critical in determining the faulty assumptions. Johan de Kleer also uses this technique in his debugging program INTER <de Kleer 1976>. A related process of relaxation of symbolic constraints has been applied to the labelling of line drawings of visual scenes <Muffman 1970>. A beautiful exposition of this technique can be found in <Waltz 1972>. Some theoretical analysis of this technique appears in <Freuder 1976>.

TOPLE <McDermott 1974> was an early attempt to record the interactions among deductions for the purpose of deciding what is currently believed to be true. McDermott used this information to help decide which of several assumptions must be thrown out in order to keep a consistent data base when a new fact conflicted with existing ones. MYCIN <Shortliffe 1974> <Davis 1976> use dependency information to produce explanations but do not use it for any control purposes. The SRI Computer Based Consultant <Fikes 1975> makes use of dependencies to determine the logical support of facts in a manner similar to ARS but does not use them to control search.

Two other recent research efforts at MIT have developed these ideas further. Drew V. McDermott finished a PhD thesis <Mc Dermott 1976> concerning the design of such systems. Drew developed a rule-based language, called NASL, in which it is possible to express strategies, tactics, and advice for design. He used this language to encode some general design strategies and some specific strategies for the design of electrical systems. Howie Shrobe and Charles Rich <Rich & Shrobe 1976> designed and mostly implemented a system which "understands" a limited class of LISP programs. They have developed set of linguistic constructs for attaching a set of structured comments to a program which relate it to its plan. These plans look *very* much like the plans of Brown. They have also developed a system which reasons about the program in terms of its plan, and which can check that a program in fact implements its goals. In effect their system employs the teleological information in the plan about the parts of the program being checked, as an outline for the verification of the program.

Finally, there have been many books about the strategies of the design process (for example <Alexander 1964>, <Asimow 1962>, <Buhl 1962>, <Glegg 1973>) but these are mostly simple advice about how to avoid overlooking a good approach when working out a hard problem. They offer little help in how to propose new solutions to new problems. Artificial intelligence researchers have been interested in the design process as a model of creativity. Computer Science in general has been interested in design because of the "complexity barriers" apparent in the design of large systems. Herbert Simon <Simon 1969> wrote a delightful and insightful book which relates computer science to general engineering design and to cognitive theory. At Carnegie-Mellon University, for example, Grason <Grason 1970> wrote a PhD thesis on the relaxation of architectural constraints, and Hanley <Hanley 1968> wrote a PhD thesis on computer-aided design of computer instruction sets.

One would expect the CAD literature to deal extensively with systems to save an engineer time and effort. A survey of this literature (see <Kuo & Magnuson 1969> <Furman 1970> <Dertouzos 1972> <Vlietstra & Wielinga 1973> <Rosenbrock 1974>) shows that the thrust of CAD development has been in the development of interactive graphics packages, libraries of special purpose programs, and mathematically sophisticated programs aimed at analysis or optimization of synthesis. Only a small amount of work has been done in the field of synthetic reasoning, and then only in restricted domains where algorithms are available to solve a small class of problems. Such approaches have been partially successful in the problem of printed-circuit layout (e.g. <Fletcher 1974>) and filter design (e.g. <Chohan A Fidler 1974>). Director <Director 1974> describes a circuit design program which assumes a full-graph impedance network and then optimizes the network for the behavior desired. In the process, many component values become zero and *are* thus discarded? The CAD literature is almost completely ignorant of non-numerical techniques (except <Powers 1973>) and would benefit from an infusion of these new ideas. Besides our work, the TROPIC system <Latombe 1976> is one other application of artificial intelligence ideas to CAD. John S. Brown <Brown & Burton 1975> (at Bolt, Beranek and Newman, Cambridge, Massachusetts) has been applying both artificial intelligence ideas and CAD ideas to the problem of computer-aided instruction of circuit debugging skills for technicians. We expect that the work of his group will contribute to the understanding of AI issues in CAD.

Conclusions:

A major problem confronting builders of automatic problem-solving systems is that of the combinatorial explosion of search-spaces. One way to attack this problem is to build systems that effectively use the results of failures to reduce the search space — that learn from their exploration of blind alleys. In simple cases, as in analysis of circuits, various automatic techniques such as the dependency-directed backtracking of ARS can go a long way toward controlling the search. In more complex situations, as in design, it is necessary to constrain the search as rapidly as possible — even if that sometimes overconstrains the problem and causes a bug. At least we can hope that the debugging problem is easier than the search problem. In either case it is necessary to build problem solvers so that they can remember and explain their reasoning. Both dependency-directed backtracking and problem solving by debugging almost-right plans depend on the ability to manipulate the justification of a conclusion as well as the ability to deduce it.

Saving justifications for the intermediate results of a computation has other merits. It is very difficult to debug programs containing large amounts of knowledge. The complexity of the interactions between the "chunks" of knowledge makes it difficult to ascertain what is to blame when a bug manifests itself. A program which can explain the reasons for its beliefs is more convincing when right, and it is easier to debug when wrong.

Acknowledgement:

1 would like to thank the many people who have contributed to this project. Richard (RMS) Stallman, Drew McDermott, Allen Brown, Guy (QUUX) Steele, Jon Doyle, Johan de Kleer, Marilyn Matz and Gerald Roylance have worked with me on it. Chuck Rich, Howie Shrobe, Scott Fahlman, Marvin Minsky, Seymour Papert, Louis Braida, Paul Penfield, Kurt Vanlehn, Richard Fikes, John Allen, David Marr, Pat Winston, and Earl Sacerdoti provided good advice and important ideas.

Bibliography:

Alexander, C. (1964) Notes on the Synthesis of Form, Cambridge: Harvard University Press.

Asimow, M. (1962) Introduction to Design, Englewood Cliffs, NJ: Prentice-Hall, Inc.

Brown, Allen (1975) Qualitative Knowledge, Causal Reasoning, and the Localization of Failures, Cambridge: unpublished MIT Ph.D. thesis.

Brown, John Seely and Richard R. Burton (1975) "Multiple Representations of Knowledge for Tutorial Reasoning" in Daniel Bobrow and Allan Collins (Eds.), Representation and Understanding: Studies in Cognitive Science, New York: Academic Press.

Buhl, H.R. (1962) Creative Engineering Design, Ames, Iowa: The Iowa State University Press.

Chohtn, V.C and IK. Fidler (1974) "Computer Aided Design of Filters for Data Transmission Using Frequency Modulation" Proceedings of the International Conference on Computer Aided Design, 1974.

Dahl, OJ., E.W. Dijkstra and CAR. Hoare (1972) Structured Programming, London: Academic Press.

Davis, Randall (1976) Applications of Meta Level Knowledge to the Construction, Maintenance, and Use of Large Knowledge Bases, Stanford, CA: Stanford University Artificial Intelligence Laboratory Memo 283.

de Kleer, Johan (1976) Local Methods for Localization of Faults in Electronic Circuits, Cambridge: MIT Artificial Intelligence Laboratory Memo 394.

Dertouzos, Michael (1972) "CIRCAL-2: General-Purpose On-Line Circuit Design", Proceedings of the IEEE, Vol. 60, pp. 39-48, Jan. 1972.

Dijkttra, Edsgar (1970) "Structured Programming" in Software Engineering Techniques, JLN. Buxton and B. Randell (Eds.), NATO Scientific Affairs Division, Bruseels, Belgium, 1970.

Director, S.W. (1974) Towards Automatic Design of Integrated Circuits", in William R. Spiders (Ed.) Basic Questions of Design Theory, New York: American Elsevier Publishing Company, Inc., p. 303.

Doyle, Jon (1976) The Use of Dependency Relationships in the Control of Reasoning, Cambridge: MIT Artificial Intelligence Laboratory Working Paper133.

Ernst, George W. and Allen Newell (1969) GPS: A Case Study in Generality end Problem-Solving, New York: Academic Press.

Fahlman, Scott (1973) A Planning System for Robot Construction Tasks, Cambridge: MIT Master's Thesis.

Fikes, Richard (1975) Deductive Retrieval Mechanisms for State Description Models, Menlo Park, CA: Stanford Research Institute Artificial Intelligence Center Technical Note 106.

Fletcher, A.J. (1974) "EUREKA - A System for the Automatic Layout of Single-Sided Printed Circuit Boards," Proceedings of the ZIInternational Conference on Computer Aided Design, 1974.

Freeman, P. and Allen Newell (1971) "A Model for Functional Reasoning in Design," Proceedings of International Joint Conference on Artificial Intelligence II, p. 621.

Freuder, Eugene (1976) Synthesizing Constraint Expressions, Cambridge: MIT Artificial Intelligence Lab Memo 360.

Furman, TA (1970) (Ed.) The Use of Computers in Engineering Design, London: English Universities Press.

Glegg, Gordon Lindsay (1973) The Science of Design, Cambridge, Eng.: Cambridge University Press.

Goldstein, Ira (1974) "Summary of MYCROFT: A System for Understanding Simple Picture Programs," Artificial Intelligence, Vol. 6, No. 3, Fall, 1975, pp. 249-288.

Grason, Jason (1970) Methods for the Computer-Implemented Solution of a Class of "Floor Plan" Design Problems, unpublished Ph.D. dissertation, Pittsburgh: Carnegie-Mellon University.

Hanley, Frederick (1968) Using a Computer to Design Computer Instruction Sets, Pittsburgh: Carnegie-Mellon University Computer Science Department Ph.D. thesis.

Hewitt, Carl and Brian Smith (1975) Towards a Programming Apprentice, Cambridge: MIT Artificial Intelligence Laboratory Working Paper 90.

Huffman, David (1970) "Impossible Objects as Nonsense Sentences," in Machine Intelligence 6, Edinburgh, UK.: Edinburgh University Press (1970).

Kuo, F.F. and W.G. Magnuson (1969) (Eds.) Computer-Oriented Circuit Design, Englewood Cliffs, NJ: Prentice-Hall, Inc.

Latombe, Jean-Claude (1976) Artificial Intelligence in Computer-Aided Design: The Tropic System, Menlo Park, CA: Stanford Research Institute Artificial Intelligence Center Technical Note 125.

McDermott, Drew (1974) Assimilation of New Information by a Natural Language Understanding System, Cambridge: MIT Department of Electrical Engineering and Computer Science Master's Thesis.

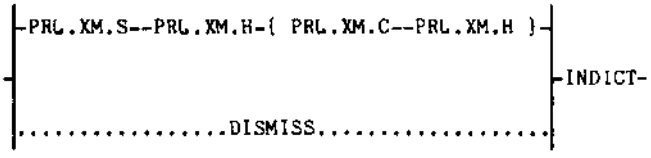
McDermott, Drew (1976) Flexibility and Efficiency in a Computer Program for Designing Circuits, Cambridge: MIT Department of Electrical Engineering and Computer Science Ph.D. Thesis.

Polyt, George (1945) How to Solve It, Princeton, NJ: Princeton University Press,

Polya, George (1962) Mathematical Discovery: on Understanding, Learning, and Teaching Problem Solving, Vols. I and II. New York: John Wiley and Sons, Inc.

3. The STATUS transition, if any (e.g., "PRL.XM.S" can occur only if the defendant is in the "Magistrate's Proceedings" status, and the event will not cause a status change),
4. The formal parameters necessary to specify the data contents of the event (e.g., "PRL.XM.S" requires that the scheduled date of the examination be supplied),
5. The Speedy Trial time accounting actions necessary as a result of posting the event,
6. Any special preconditions that must be satisfied in order for the event to be posted,
7. Any special semantic actions that must be performed upon posting the event,
8. The text that is to be printed on a docket sheet when this event is reported.

In addition to the event sequencing constraints represented by the status diagram, arbitrary sequencing relationships may be specified to control the order in which events may be posted within any given status. These relationships are represented graphically as SEQUENCE DIAGRAMS and extend the finite state nature of the STATUS DIAGRAM. In addition to strict event sequences, SEQUENCE DIAGRAMS may denote forks and joins, optional sequences, and parallel paths. The state of any given subject with respect to the progress of a case may be described as the STATUS containing the subject- plus a vector of "program counter" values indicating the subject's position along each possible parallel path in the SEQUENCE DIAGRAM which further defines the STATUS.



Event Sequence Diagram for "Magistrate's Proceedings" Figure 2.

Figure 2 depicts a simplified event sequence diagram for defendant subjects in the "Magistrate's Proceedings" status, showing the allowable sequences for Setting, Holding, and Continuing preliminary examinations. The { } indicate that the enclosed sequence may be repeated zero or more times. The vertical bars denote a fork and join operation on the enclosed independent event sequences. The dotted line indicates that the DISMISS event may be posted at any time between the fork and join to interrupt and curtail the parallel sequences. According to Figure 2, before an INDICT event can be posted, a preliminary examination must first be set (PRL.XM.S) and then held (PRL.XM.H), and optionally continued (PRL.XM.C) and re-held (PRL.XM.H) an arbitrary number of times. However, at any time during this preliminary examination sequence, DISMISS may be posted, thereby escaping from the sequence and causing a status change to the "Post-Trial" status (see Figure 1).

The event dictionary, status diagrams, and sequence diagrams represent the court-oriented knowledge of the CR1MNL system and are declaratively specified in tables rather than being, for example, procedurally embedded. The system acts essentially as an interactive interpreter which accepts proposed events from the user and tries to apply the semantic actions of each event to transform the current state of its subject. The intelligence of the system is exhibited by its responses to proposed events and the reports produced for effective case monitoring. Not only can CRIMNL detect when an event is invalid based on the current state of its subject, but the system can also offer diagnostic advice as to which events are allowable at that time and which events need to be posted first in order that the attempted event become valid. This model-based validity checking and diagnostic advice allows the detection of clerical errors and results in increased data base integrity. This same model-based system also provides the Courts with a valuable training tool for the instruction of docket clerks in criminal procedure. Furthermore, as a result of explicitly defining each subject's state with respect to its progress in the criminal proceedings, it is possible to generate exception reports which monitor case progress and warn of approaching procedural time constraints, an especially complex clerical task imposed on the Courts by the Speedy Trial Act.

The system is programmed in SAIL using DBMS-10 and runs on a DECsystem-10 computer in Washington, D.C., accessible from terminals located in the various user Courts via a value-added network. The initial "pilot" group of Federal Courts are located in Los Angeles, San Francisco, Chicago, Detroit, New York, and Washington, D.C.

REFERENCES

Buchanan, J.R., "Management Information Systems for the Federal Courts," Proceedings of "Interactive Information and Decision Support Systems," Office of Naval Research, Wharton School of Management, November 1975.

Ebersole, J.L., and J.A. Hall, "COURTRAN, An Information System for the Courts," Journal of Computer & Law, Rutgers Univ., 1972.

Federal Rules of Criminal Procedure for the United States District Courts, August 1976.

Lesser, V.R., R.D. Fennell, L.D. Erraan, and D.R. Reddy, "Organization of the Hearsay II Speech Understanding System," IEEE Trans. ASSP, 23, pp. 11-23, February 1975.

Nihan, C.W., "COURTRAN, An Assessment of Applications and Computer Requirements," Federal Judicial Center Report, September 1974.

Speedy Trial Act, Public Law 93-619, 93rd Congress, S.754, January 1975.