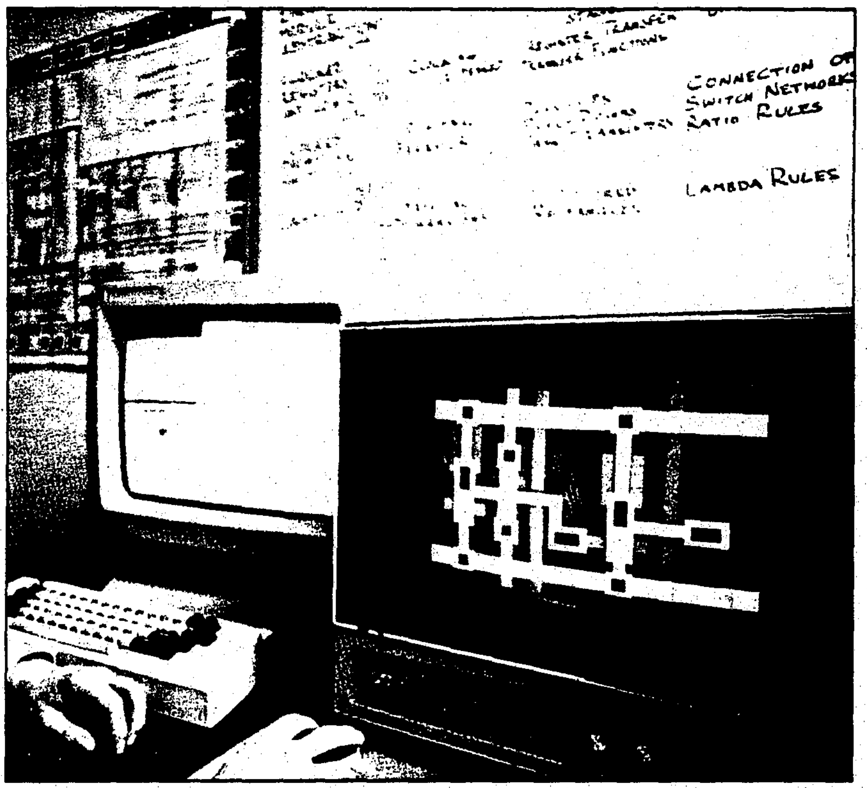
PROSPECTS FOR EXPERT SYSTEMS IN CAD



Although widespread use of expert systems in solving complex CAD problems is several years away, artificial intelligence concepts are being applied in experimental systems for tomorrow's knowledge based design assistance.

by Mark J. Stefik and Johan de Kleer

A new breed of computer systems—*expert systems*—is now emerging from artificial intelligence research and is being used in applications normally thought to require human specialists for their solution. For example, expert systems have been used to solve problems such as equipment diagnosis, medical diagnosis and therapy, experiment planning in genetics, and computer configuration—problems that do not yield to numerical or statistical techniques. Expert systems gain their power by the use of "expert knowledge," recorded in a knowledge base, which enables programs to mimic the reasoning of human experts.

Mark J. Stefik is a member of the research staff at Xerox Corp, Palo Alto Research Center, 3333 Coyote Hill Rd, Palo Alto, CA 94304. Dr Stefik works in the Knowledge Systems Area. He has a PhD in computer science from Stanford University.

Johan de Kleer is a member of the research staff at the Xerox Palo Alto Research Center's Cognitive and Instructional Sciences Group. Dr de Kleer has a PhD from the Massachusetts Institute of Technology's Artificial Intelligence Laboratory.

In applying expert systems to design tasks, the idea is to pit knowledge against complexity, using expert knowledge to whittle complexity down to a manageable scale. Expert systems will eventually be applied in many design areas, but an important example is their use in digital system design, particularly in computer aided design (CAD).

Although there are no expert systems commercially available for electronic system design as yet, work is proceeding at several research centers. For instance, computer scientists at Digital Equipment Corp (DEC) are developing expert systems for several applications, including digital design. One example is an experimental expert system for determining transistor size in integrated circuits, given circuit parameters such as load and capacitance. In conjunction with researchers at Carnegie-Mellon University in 1978, DEC began activity on expert systems to develop a knowledge based program called XCON for configuring VAX-11/780s. This program is now used to configure every VAX that is shipped.

For more than a decade, the Heuristic Programming Project at Stanford University has pioneered and developed expert systems. Eurisko is an artificial intelligence (AI) program used to search for mathematical concepts and configure naval fleets in competition games. Recently, Eurisko was used to search for useful microcircuit structures made possible by multilayer fabrication technology (Fig 1). The program has discovered several novel devices.

Researchers at the Massachusetts Institute of Technology Artificial Intelligence Laboratory have been creating and experimenting with engineering AI programs for many years. Two examples are the EL and SYN programs that help a designer analyze and synthesize analog circuits. Key to these systems is the idea that the possible values for parameters describing circuits are represented and manipulated in terms of constraints. Fig 2 shows an example of the kind of problem that EL can solve.

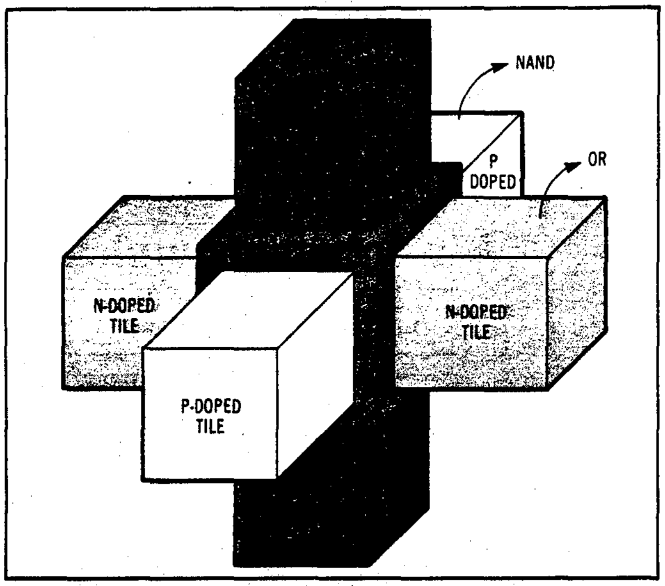


Fig 1 Illustration of a microcircuit device discovered by Eurisko. This device can be used to simultaneously compute NAND and OR. The device handles silicon volume efficiently since it can be packed in the plane and vertically. Eurisko uses a technique called heuristic search to generate plausible devices and find the interesting ones.

At Xerox Palo Alto Research Center and Stanford University, researchers are developing a prototype expert system called Palladio. The key idea is that designers should design not only circuits, but also knowledge. Knowledge in Palladio is expressed in a knowledge representation language called Loops (see Fig 3). Using Palladio, a designer will interact with previously designed circuit fragments and rules taken from knowledge bases. While working on an individual design, a designer can discover gaps and errors in the knowledge base by applying it to his own design. The designer can create personal versions and modifications to the knowledge bases, which can later be incorporated. Palladio is intended to foster experimentation with design methodologies.

Related work is proposed or underway at the Japanese 5th Generation Computer Project, the Institute for Information Sciences, Lincoln Labs, and Symbolics.

The technology

One method for developing an expert system involves a collaboration between a specialist (the *expert*) and a computer scientist (the *knowledge engineer*). The expert provides sample problems and the knowledge engineer provides a programming framework, including a knowledge representation language. The knowledge engineer interviews the expert as he solves the sample problems and helps him to articulate the knowledge he is using. Together they define the scope of the expert system and enter the expert's knowledge into a knowledge base, often in terms of if-then rules. The resulting program is given hypothetical tasks. Differences between the expert's results and the program are identified, and the knowledge base is updated.

Developing an expert system generally takes several man-years. For a project to succeed, the expert and the knowledge engineer must become familiar with each other's field. During this process, a body of knowledge is articulated and formalized. Often the necessary formalization of knowledge leads to a crisper and deeper understanding by the expert of his field. Sometimes the formalization leads to knowledge and methods of reasoning that have not been used previously.

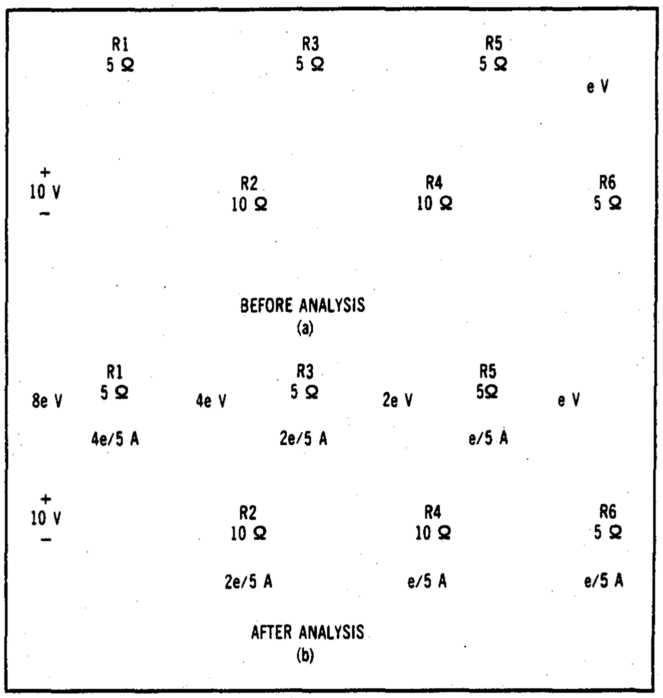


Fig 2 A simplified example that can be solved by EL "Before analysis" (a) shows an electrical circuit whose components are known, but whose voltages and currents need to be analyzed. Analysis begins by assigning the symbol e to the unknown voltage at the upper right corner of the ladder. Other values are derived by stepwise application of Ohm's and Kirchoff's laws to produce the "after analysis" diagram (b).

Although this depiction of an expert system's construction and organization is oversimplified, it illustrates the kinds of issues the expert system builder must face and the technology that must be applied. The knowledge engineer's task is to encode the expert's knowledge to be effectively used by a computer. This task is difficult because the expert is rarely articulate about the knowledge. Such difficulties are addressed by expert systems technology.

Debugging a knowledge base provides a vehicle for formalizing an expert's often tacit knowledge. Throughout expert system development, the knowledge at certain

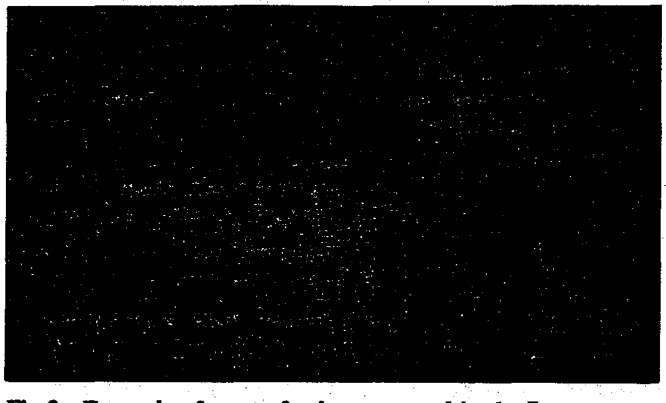


Fig 3 Example of a set of rules expressed in the Loops language. Rules like these can provide heuristics for assigning directions and levels to "wires" in a switch representation of a circuit.

places is incomplete and incorrect. This is apparent when a human expert's conclusions differ from the expert system's. Identifying which piece of knowledge is absent or incorrect is difficult.

Transparency is an important property of expert systems. Transparency means that the knowledge an expert system uses should be made visible to its users. In particular, the user can view and often modify the knowledge base. More important, the system, in arriving at a recommendation, maintains an audit trail of the steps and pieces of knowledge used. Hence, the culprit piece of knowledge is easily identified. (Such audit trails also provide explanations for its recommendations, thereby helping users to develop confidence in an expert system.) These mechanisms for user feedback allow the knowledge base to continue to grow long after the knowledge engineer has left.

In an application, knowledge can include facts, theorems, heuristics, equations, rules of thumb, assumptions, strategies, tactics, probabilities, advice, and causal laws. To manage such diverse forms of knowledge, AI has developed a wide variety of *knowledge representation* and *inference* schemes. A knowledge representation scheme is a way to codify knowledge; an inference scheme is a way to use knowledge to arrive at new knowledge. One of the first decisions a knowledge engineer must make is the choice of knowledge representation and inference schemes.

A good starting place is the vocabulary and description of those things that the expert system will reason about. For example, an expert system about chip design would include a vocabulary of transistors used as switches, restoring logic, clocks, and steering logic. One popular approach is to define such things in terms of objects (Fig 4). Most knowledge representation languages come with special facilities for instantiating, combining, and specializing object descriptions.

Often the knowledge that goes into an expert system is decision-making knowledge that can be expressed in terms of if-then rules (Table 1). These rules indicate that certain actions can be taken if certain kinds of situations arise. Most rule based systems contain hundreds of rules, each representing a "chunk" of knowledge about a particular field. Rules can be used to represent both *inferences* and *heuristics*.

Each rule in a knowledge base represents the knowledge behind a single decision. Organizing the rules to work collectively on problems is an important task for a knowledge engineer. As this is an area of active research, there is more than one approach. One of the simplest strategies is to have an interpreter scan the rules to find one whose antecedents match assertions in the data base. In more sophisticated systems, rules are organized into networks that determine when rules get applied. Such networks can be organized to apply rules when certain data are changed, or to try rules when certain goals are indicated. Inferences are statements about what facts follow from given conditions; heuristics are rules of thumb that guide the search for solutions.

The current technology of building expert systems has weak points, some of which require substantially more research. For example, an expert, like a good designer, is distinguished from a mediocre one not only by the knowledge possessed about the field, but by intuition, taste, and common sense. Unfortunately, these are difficult to recognize, let alone formalize. As a consequence, most current expert systems lack breadth and common sense. That is not to say that they cannot solve difficult problems requiring sophisticated expertise. Rather, when posed slightly different problems than they were designed for, they can fail in surprising ways.

AI is only recently facing these issues squarely, and it will be some time before one sees anything even approximating broad human expertise. Nevertheless, a great

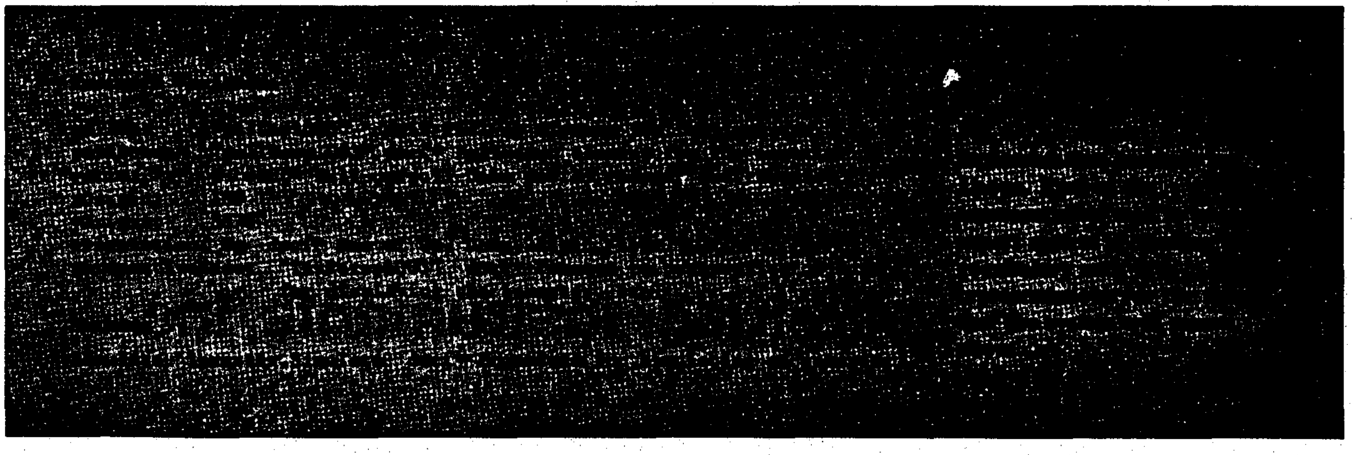


TABLE 1

Sample If-Then Rules

The XCON System (configuring VAX systems)	
IF:	The current active context is selecting a box and a module to put in it
	The next module in the optimal sequence is known
	The number of system units of space that the module requires is known
	At least that much space is available in some box
	That box does not contain more modules than some other box on a different unibus
THEN:	Try to put that module in that box
The MYCIN System (medical diagnosis and therapy)	
IF:	The site of the culture is blood
	The portal of entry of the organism is GI
	The patient is a compromised host
THEN:	It is definite (1.0) that bacteroides is an organism for which therapy should cover
The PROSPECTOR System (mineral exploration)	
IF:	Volcanic rocks in the region are contemporaneous with the intrusive system (coeval volcanic rocks)
THEN:	(800, 1) The level of erosion is favorable for a porphyry copper deposit

deal of technology can be of immediate practical use in expert systems to solve problems that require a significant amount of expertise or creativity.

Design as search

In artificial intelligence, many problem solving systems are based on the formulation of problem solving as search. In this formulation, a description of a desired solution is called a goal, and the set of possible steps leading from initial conditions to possible solutions is the space to be searched. Problem solving is carried out by searching for sequences that lead to solutions that satisfy a goal.

Often, rules in expert systems can be viewed as heuristics for generating and pruning candidate solutions. Search is at the heart of a reasoning system, and failure to organize it properly can result in problem solvers that are inefficient, naive, or unreliable. The simplest approach is to search a solution space exhaustively (ie, to search it in a way that will find all possible solutions). This is appropriate only if the space of possible solutions is quite small, or if powerful pruning heuristics are available for quickly eliminating most of the space from consideration. Another approach is to use heuristics for plausible generation (ie, to guide the search to the most promising avenues).

Stanford University's Eurisko is an example of an AI program that uses heuristic search. It has been applied to the task of inventing new 3-dimensional microelectronic devices that can be fabricated using laser recrystallization techniques. Eurisko's exploration is carried out by generating a device configuration, computing its input/output behavior, parsing this into a functionality that it recognizes, and then evaluating the device against known comparable devices.

Two different solution space characterizations have been tried in Eurisko. In the first experiments, the solution space was characterized in terms of abutted regions of doped and undoped semiconductors. Devices were analyzed at the level of charged carriers moving under electric field effects. Many well-known primitive devices were synthesized quickly, such as the metal oxide semiconductor field effect transistor, the junction diode, and the bipolar transistor.

In the next experiments, a level describing circuits in terms of *tiles* was tried. In the tile model, each region is a three-space cube of approximately the same size. A device is a lattice of tiles in a particular 3-dimensional configuration. Fig 1 is an example of a device described in terms of tiles. Initial tile level experiments were done using exhaustive search. Basic elements supplied included the logical operations, flipflops, stack cells, and a few others. Thousands of hours of runs with this version of Eurisko convinced the experimenters that the "hit rate" for good devices was below one in a billion.

In the next set of experiments, Eurisko used heuristic search rather than exhaustive search. In the previous experiment, a new device was synthesized every 0.9 s. Now, with a hundred heuristics guiding the generation process, it took about 30 s to produce each device design (using a Xerox 1100 personal scientific information processor). However, the frequency of valuable new devices rose to 1 in 10. Much of the power came from a symmetrizing heuristic.

Eurisko's success suggests that computers can play active creative roles in the design process. Although large circuit design is still a distant possibility, a new symbiotic relationship is emerging between designer and machine. The designer can be free from concern about a particular device's structure, and instead concern himself with heuristics for guiding the computer program's search for interesting designs. The computer can thus explore the possibilities based on the heuristics and present alternatives to the designer for evaluation.

Gaining leverage through abstraction

Coping with design details is a challenge. While failure to attend to them leads to disaster, focusing on them in a project's early stages can blind a designer to the overall picture. For example, a digital systems designer can become bogged down in his approach to system architecture if he initially tries to contend with the details of every transistor. Instead, the approach should make big design steps to get the whole picture, then tend to the details systematically.

In software practice, programming languages provide an abstract level of description for programs that allow bigger steps than machine instructions. The same idea of using formal languages has been proposed for hardware.

Fig 5 illustrates three architectural approaches to designing a hardware stack. In a high level architectural language like Linked Module Abstraction (LMA), which emphasizes storage and communication features, descriptions are concise. Furthermore, minor architectural variations often correspond to minor changes in the descriptions. The pointer stack (a) corresponds to the usual software implementation. Information is stored in a register array. An index register (the pointer) contains the top of the stack's address. Push and pop instructions increment and decrement the pointer. The roving marker stack (b) uses a mark bit associated with each storage cell to indicate the top of the stack. In a push or pop instruction all cells receive the command, but only the one with the mark bit set performs the operation. It combines a register array for data storage with a shift register for marker storage. In the buffered stages stack (c), the top of the stack is always the leftmost cell and all data in the stack move simultaneously. Intermediate stages buffer the data as they move between cells.

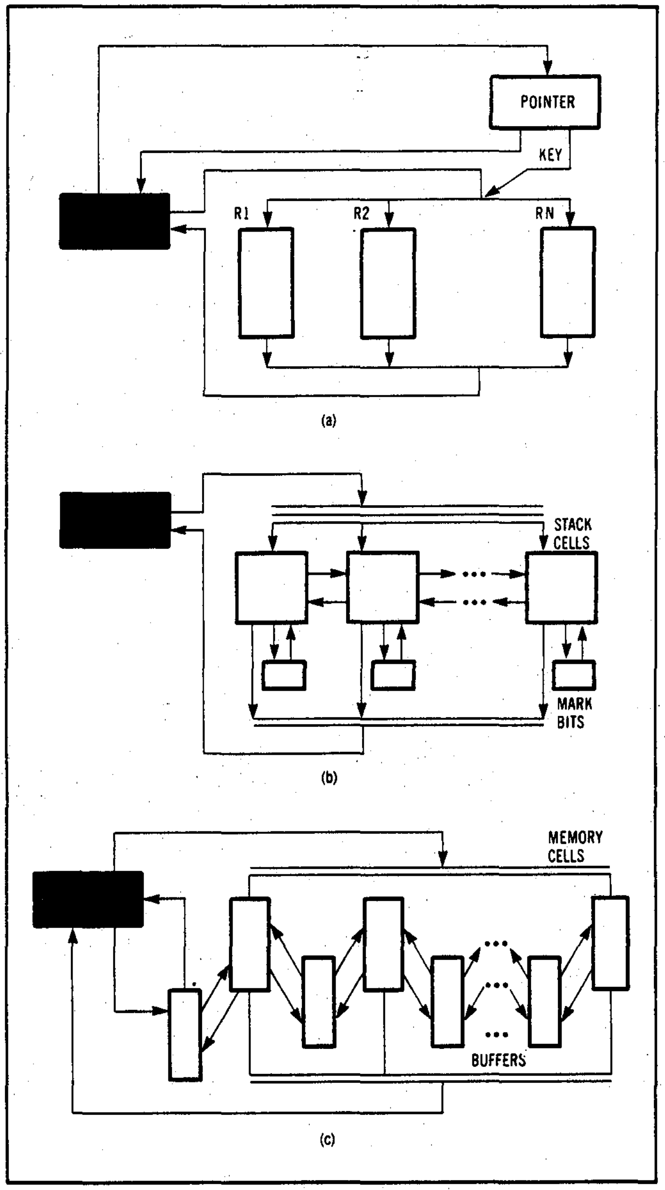


Fig 5 Some alternative designs for a stack: pointer stack (a), roving marker stack (b), and buffered stages stack (c). When specifications for these different stacks are written in a language that emphasizes storage and communications features, the description is much shorter than one written in terms of device layouts. Furthermore, minor architectural variations often correspond to minor changes in the descriptions.

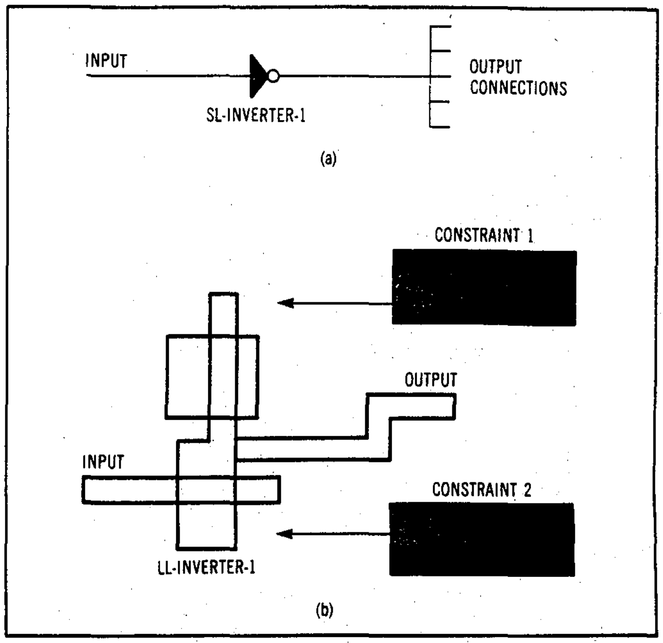


Fig 6 Introducing constraints in a design. In this figure, the layout level description (b) is an implementation of the switches-level description (a). Implementation is only partially specified since the designer chooses only to introduce constraints about connections to power and ground, rather than deciding exactly how to route the wires to make the connections.

Ideas of languages and big steps can be iterated to yield an ordered set of languages providing intermediate abstractions. (See Table 2.) Languages serve to partition design concerns, and each successive description of a system surfaces a new level of details to consider. Using the languages in design can be helpful to a designer if decisions that are made at a more abstract level do not need to be revised in a more detailed language. A designer who systematically carries a design through several implementations in different languages is guided by an "invisible hand" that determines the kinds of decisions made at each step.

Invention and language experimentation for describing abstractions is not unique to expert systems. For example, using multiple languages to describe hardware has been tried many times. The main point is that such languages can enable big steps in roughing out a design. Since the languages divide up a design process, they also provide a framework in an expert system for organizing knowledge according to sets of concerns.

Representing constraints and dependencies

Exploratory design often requires making significant changes late in the design cycle. This is inconvenient with current design tools, which do not capture the dependencies among decisions. To make a change, designers often must alter large portions of a design that are disturbed by a seemingly minor change.

In AI research, it is useful in such problems to represent explicit *dependencies* and *constraints*. A dependency is a situation where one quantity depends on others. This is analogous to the algebraic distinction

TABLE 2

Synthesis Languages for Palladio

Description Level	Concerns	Terms	Composition Rules	Bugs Avoided
Linked module abstraction (LMA)	Event sequencing	Modules Forks Joins Buffers	Token conservation Fork/join rules	Deadlock Data not ready
Clocked registers and logic (CRL)	Clocking 2-Phase	Stages: Register transfer Transfer functions	Connection of stages	Mixed clock bugs Unclocked feedback
Clocked primitive switches (CPS)	Digital behavior	Pull-ups Pull-downs Pass transistors	Connection of switch networks Ratio rules	Charge sharing Switching levels
Layout	Physical dimensions	Colored rectangles	Lambda rules	Spacing errors

between dependent and independent variables. Dependencies can be used in a design to represent situations where the system could automatically make changes in one part of a design to reflect changes made in another part. A constraint is a situation where several quantities are interrelated. Unlike dependencies, a constraint does not distinguish between dependent and independent quantities. In AI programs, constraints are coupled with knowledge for propagating and satisfying sets of constraints.

EL is an example of an AI system that represents and manipulates constraints. As an aid to designers analyzing analog circuits, EL computes electrical parameters of a circuit using circuit behavior laws, such as Ohm's law and Kirchoff's current law. EL's knowledge about the values of voltages and currents at different parts of a circuit is represented as symbolic constraints.

Much of EL's power derives from the ability to reason with constraints by propagating them algebraically through a circuit description. For example, using Ohm's law constraints can be propagated in three ways. First, if the voltage across the resistor is known, and the resistance is known, the current through it can be assigned. Second, if the voltage across the resistor is known, and the current through it is known, the resistance can be assigned. And third, if the current through the resistor is known, and the resistance is known, the voltage across the resistor can be assigned.

Constraints are an important vehicle for representing partial specifications (Fig 6). For example, constraints can describe requirements for parts of a structure whose implementation is yet to be worked out. The SYN program provides an example of this in an AI system. SYN was applied to the task of circuit synthesis (ie, determining the parameters of the parts of a network). Like EL, SYN uses constraints for representing assertions about a circuit. Much of a constraint's power is that it allows a designer to specify only part of a circuit; eg, imposing a constraint that the bias current be 10% of the transistor current does not completely determine what the bias current is. From the point of view of bias stability, the exact value is irrelevant. Other constraints can be imposed to determine its value.

Using constraints to characterize partial specifications can be especially useful for representing interface contracts between parts of a design, especially when different people design the parts. A design system that includes the means for communication between designers could provide support for negotiating changes to interfaces. Such a system would help mediate the tension between defining interfaces early to divide the labor, versus revising interfaces later as a design is fleshed out. It would make it easier for designers and managers to see when interface contracts were violated, and also to weigh the effect of proposed changes.

Reasoning with heuristics

An important but sometimes unrecognized characteristic of designing is the necessity of making guesses along the way. This follows from the absence of a complete synthetic design theory in most applications. To cope with the lack of a theory, designers resort to heuristic search. They begin projects without knowing exactly what is possible or what they want. In the beginning, a designer typically sets approximate goals. As he works top down, he explores the entailments of the design decisions; as he works bottom up, he gains information about what is possible and can adjust the goals. This amounts to a dialectic between what is desired and what is possible (ie, between goals and possibilities).

Heuristic reasoning is needed to compensate for the lack of a complete theory. Designers guess, but only when they have to. Some bad guesses are inevitable. This creates an incentive to find ways to revise decisions efficiently when guesses do not work out. In AI systems, dependency and constraint records are valuable in systems that reason heuristically, then revise decisions. This can be illustrated by examples from the EL system. In analyzing circuits, EL follows a heuristic approach from electrical engineering called the *method of assumed* *states*. This method uses a piecewise linear approximation for complicated devices. It also requires making an assumption specifying a linear region for device operation. For example, EL has two possible states for diodes (on and off) and three states for transistors (active, cutoff, and saturated). Once a state is assumed, EL can use tractable linear expressions for the propagation analysis as before.

After making an assumption, EL must check whether the assumed states are consistent with the voltages and currents predicted for the devices. Incorrect assumptions are detected by means of a contradiction. When this happens, some assumptions need to be changed.

An infrastructure for groups of designers to create an articulated and permanent body of knowledge is a key idea in expert systems.

Intelligent contradiction processing involves determining which assumptions to revise. This is where the dependency records come in. EL uses these records to save justifications for each of its 1-step deductions. These justifications enable EL to identify the assumptions behind a contradiction. Knowing what information in the analysis is dependent on contradictory assumptions enables EL to focus its attention in revising the analysis. First it decides which asumptions to withdraw, then it can gracefully withdraw those additional decisions that are dependent on the original bad assumptions. Contradictions are remembered so that choice combinations found to be inconsistent are not tried again.

Accumulating design experience

Currently, much of the design knowledge is informal and not written down. Expert designers carry in their heads the knowledge gained from experience, such as special rules of thumb and particular elegant solutions to special problems. This knowledge builds up as designers search for solutions to particular design problems. However, when a designer leaves or forgets, this knowledge can be lost.

A key idea for expert systems for design is to provide an infrastructure for groups of designers to create not only designs, but also an articulated and permanent body of knowledge. For example, a designer of bit-serial signal processors could create a collection of parameterized components (at some level of abstraction) that carry out part of a signal processing task, a body of composition rules for combining these components, and a body of implementation rules that capture some of the critical tradeoffs in mapping these devices into silicon. When such knowledge is saved in a knowledge base, it would enable other designers, less expert in bit-serial processors, to use the library of parts and rules. Community use of such knowledge would also provide the expert with feedback about the correctness and completeness of his knowledge base. In effect, the designer makes his knowledge available (through a computer surrogate) to consult with other designers.

For example, much of the success of university VLSI design courses can be attributed to the development of a network infrastructure for implementing chips quickly and economically. Extensions of this work to include packaging and boards may help accelerate the growing participation of computer scientists and others interested in experimenting with the silicon medium for building computer systems.

Future prospects

Assessing the potential impact of expert systems in CAD, it is interesting to look for limiting factors. One critical factor is the number of people with experience building expert systems. There are fewer than 200 people in the United States with experience in expert systems, and only a fraction of these have a substantial interest in design or electronics. As a field, artificial intelligence in general and expert systems in particular seem to be long on ideas and short on manpower. To maximize its impact, it is essential for AI to simplify its methods and to export its ideas.

It is reasonable to ask whether expert systems for design will emerge gradually from the more traditional work outside the expert system community. Certainly, work on silicon compilers is a step in the right direction. However, several fundamental principles for building expert systems are missing in both the silicon compiler work and the traditional CAD systems and methods. These include the use of explicit representations for knowledge (eg, as rules), explicit representations for goals and constraints, and infrastructure for adding knowledge to a knowledge base and debugging a knowledge base.

There are several encouraging signs, however. For instance, several large companies with an interest in electronics and design, including Fairchild, Hewlett-Packard, Schlumberger, and Texas Instruments, have recently organized artificial intelligence laboratories. Also powerful personal workstations supporting appropriate AI languages (eg, LISP), programming environments, and graphics are now available from Lisp Machine Inc (LMI), Symbolics, and Xerox. Finally, knowledge representation languages tailored for expert systems are emerging: Loops (Xerox), MRS (Stanford University), and OPS5 (Carnegie-Mellon University).

Research in these laboratories is likely to lead to new insights. On balance, more examples of prototypical expert systems for assisting designers in the next two or three years are expected, but the widespread use of expert systems in design is several years away.

Please rate the value of this article to you by circling the appropriate number in the "Editorial Score Box" on the Inquiry Card.

High 707 *Average 708* *Low 709*

Photo on p 65 is of the inter LISP-D/Loops programming environment used at Xerox PARC to create expert systems for VLSI system design.