Toward a Unified Theory of Cognition

Soar is a computer program that can learn from experience—and that may also explain the basic mechanisms of thought

This is one of an occasional series of articles on cognitive science and artificial intelligence, the study of the mind as an information processor. Next: How Soar works as a unified theory.

DEPENDING UPON WHOM one asks, the idea is outrageous, brilliant, premature, doomed, or even blasphemous. But according to artificial intelligence (AI) pioneer Allen Newell of Carnegie Mellon University, it is an idea whose time has come: "Unified Theories of Cognition," he declares, "are within reach and we should strive to attain them."

Anyone who has spent more than about 30 seconds with Newell lately knows just how enthralled he is with this prospect. He can happily expound upon the subject for hours on end. And yet his purpose is quite serious: nothing less than a reformation in the way research is done in the cognitive sciences. He is trying to wean his allies in cognitive psychology from "microtheories" that explain only one or two experimental results at a time. He is likewise trying to shake his fellow AI researchers out of their habit of writing ad hoc programs that focus only on one little aspect of, say, vision or language understanding. Instead, he wants them to start putting the pieces together. He wants them to start thinking in terms of a single, integrated set of information-processing mechanisms that can explain every aspect of human thought—reasoning, learning, perception, motor control, language, development, emotion, and even awareness.

"Even if the mind has parts, modules, components, or whatever, they all mesh together to produce behavior," he says. "It is one mind that minds them all."

Among the AI researchers and cognitive psychologists contacted by *Science*, the reaction to Newell's proselytizing is typically something like "It's an intriguing idea"—followed by a strong dose of Wait-and-See. People do listen: for more than 30 years, Newell has been one of the most influential theorists in any branch of cognitive science. Yet people are understandably skeptical: given the fragmentary nature of our knowledge about cognition, they wonder if we are really ready for a unified theory. And indeed, Newell is well aware that the burden of proof is on him.

So he is specific. For more than a year now he has been promoting a candidate theory that is embodied in a working computer program known as Soar,* which he has developed in collaboration with his former students John E. Laird of the University of Michigan and Paul S. Rosenbloom of the University of Southern California. In simplest terms, Newell's theory is that all cognition involves some form of problem-solving; Soar is accordingly a general-purpose program for solving problems. It incorporates specific knowledge about the world as a set of rules that guide it in solving problems. And it learns from experience by remembering *how* it solves problems.

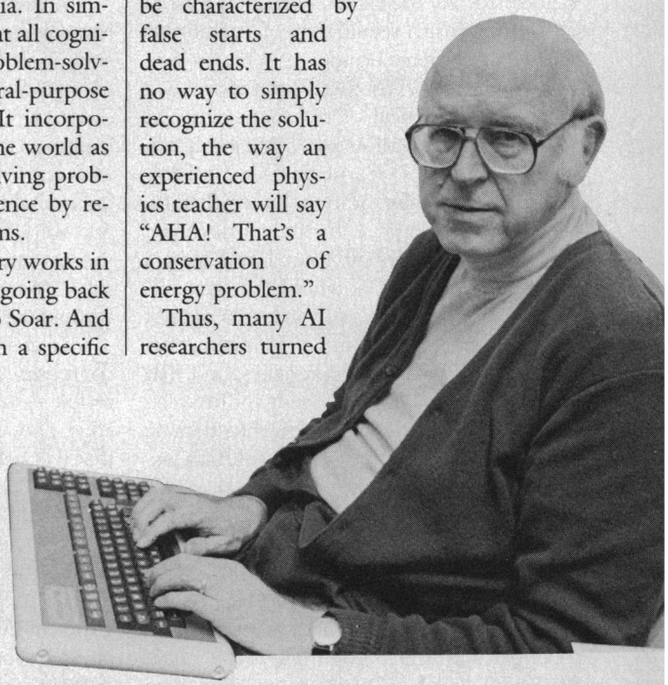
To get a feel for how this theory works in more detail, however, it is worth going back to the issues that first gave rise to Soar. And to do that, it helps to start with a specific question: what would it take to write a program that solves homework problems in elementary physics?

Since there are a wide variety of such problems—static equilibrium, harmonic oscillators, friction, and so forth—one obvious way to proceed might be to try for generality: give the program a handful of generic reasoning methods that it could apply and reapply in all sorts of situations.

As it happens this was precisely the approach taken by Newell, his Carnegie Mellon colleague Herbert Simon, and computer scientist Clifford Shaw in their pioneering work on human problem-solving back in the 1950s and 1960s. By analyzing the behavior of human subjects they were able to identify quite a number of these general reasoning techniques. Examples included such traditional favorites as Trial-and-Error and Hill-Climbing (that is, do whatever seems best at the time), as well as a very common technique known as Means-Ends analysis, which can be paraphrased as "If I'm over *here* and my goal is over *there*, then I should try to reduce the difference." Newell and Simon accordingly made Means-Ends analysis one of the cornerstones of a program known as General Problem Solver, which they developed in 1957 as a model for human problem-solving.

As advertised, General Problem Solver was quite versatile. In its various incarnations over the next decade it was able to solve a variety of puzzles, and even prove theorems in symbolic logic. By the early 1970s, however, it had also clarified the limitations of pure generality. Most important was that a *general* problem solver is forever condemned to be a novice: in much the same way that a beginning physics student will flounder around with his or her first homework assignment, a general program's reasoning will always be characterized by false starts and dead ends. It has no way to simply recognize the solution, the way an experienced physics teacher will say "AHA! That's a conservation of energy problem."

Thus, many AI researchers turned to a second approach: instead of striving for general intelligence, focus on a specific domain—in this case, elementary physics—and try to gain problem-solving power by encapsulating the highly specialized knowledge of human experts. This idea quickly gave rise to a class of programs known as expert systems, which were developed by several AI groups in the early and mid-1970s. Mycin, to take the best known example, was an early expert system written at Stanford University in 1975 to do diagnosis of infectious diseases. At its heart was a mass of about 250 rules



Allen Newell. *"Unified theories are within our reach."*

*For reasons made clear below, Soar stands for State, Operator, And Result.

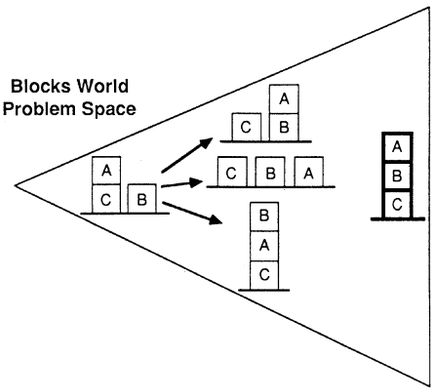
that had been formulated after extensive interviews with human doctors.

The power of these systems was something of a surprise even to their creators; in a few cases the programs outperformed even the human experts. And indeed, expert systems have been largely responsible for the boom in commercial AI applications that began in the early 1980s. They also helped feed the widespread perception that generality in the Newell-Simon sense was a chimera, that the road to intelligence lay through ever more intensive applications of highly specific knowledge—a principle inevitably paraphrased as "Knowledge is Power."

Nonetheless, as even the most ardent knowledge engineers have had to admit, expertise alone was not enough. Suppose, for example, that our expert system for physics were presented with an unusual problem that its rules did not cover. A human physicist would simply go back to first principles and start from there, using whatever general reasoning methods he or she needed. Yet the expert system (at least in its simplest form) would come to an abrupt halt. It would have no idea of what to do.

By the late 1970s these limitations had inspired a great deal of research in the AI community, much of which is still going on. Indeed, Newell's Carnegie Mellon colleagues were doing quite a bit of that research themselves. Ultimately, however, Newell and his student John Laird took a very different tack. Convinced that generality and expertise were really just two ends of a continuum, they wanted a system that would encompass *both* approaches, and that would smoothly bridge the gap between them. Achieving such a system thus became Laird's thesis project. And his solution became the basis of Soar.

As a starting point, Laird went back to the theory of problem-solving that Newell, Simon, and Shaw first set forth back in the 1950s. According to that model, all the mental activity being devoted to a given task takes place within a cognitive arena called the *problem space*. A problem space in turn consists of a set of *states*, which describe the situation at any given moment, and a set of *operators*, which describe how the problem-solver can change the situation from one state to another. In chess, for example, the problem space would be "a chess game," a state would consist of a specific configuration of pieces on the chess board, and an operator would consist of a legal move, such as "Knight to King-4." The task of the problem-solver is to search for a sequence of operators that will take it from a given initial state (say, with the pieces lined up for the start of the chess game) to a given solution state (the opponent's king in checkmate).



Problem-solving in the Blocks World problem space. *Starting from an initial state—the configuration of blocks on the left—the problem-solver has three operators that it can apply to change the blocks to a new state: move block A onto block B, move block A to the floor, and move block B onto block A. The task is to find a sequence of such operators that will produce the stack ABC shown on the right.*

The Newell-Simon-Shaw model of problem-solving is widely accepted within the AI community. Indeed, it seems general enough to account for any kind of goal-directed reasoning, from game playing to language comprehension to the question-answering of expert systems. So Laird incorporated the framework in toto. However, that still left the question of guidance: given that a certain sequence of operators will solve the problem at hand, how is the computer supposed to *find* that sequence?

The standard answer in AI is that the computer (or for that matter, a human problem-solver) has to have access to knowledge about the problem—rules of thumb that will help it avoid the dead ends and that will guide it along the most promising paths. And the standard way of encoding that knowledge is the "production system"† architecture, which was originally introduced by Newell and Simon in the late 1960s in the course of their work on human problem-solving, and which was subsequently embraced by expert systems designers and a wide variety of other AI programmers as well. The idea here is to encode each bit of knowledge as a condition-action rule of the form, "IF *this* is the case, THEN do *that*." A program that models elementary school subtraction, for example, might contain a rule that reads in English, "IF you are working on a given column, and the bottom number is greater than the top number, THEN look at the next column to the left in order to borrow."

For Laird, the production system architecture seemed an obvious starting point. However, it was certainly not the complete answer. And indeed, before he was through, his efforts to bridge the generality/expertise gap had led him to make two key changes.

To understand those changes, think of a production system as a society of little demons—the condition-action rules. These demons spend most of their time in quiet contemplation of something called "working memory," which is a kind of internal blackboard that records data about the current situation. However, when one of them sees something it likes—that is, when the conditions on the IF side of the rule match the current situation in working memory—it jumps up and shrieks out the command listed on its THEN side: "DO *this*." The program obeys, taking whatever actions are demanded and making the appropriate changes to working memory. And then everyone settles down again to wait for another demon to jump up.

Now, so far, so good: if all goes well the computer will simply follow the orders of each shrieking demon in turn, moving from step to step until it reaches a solution to the problem at hand. But what happens if things do not go well? What happens if none of the demons has anything to say—that is, if none of the rules apply to the situation at hand? Worse, what happens if several demons are activated simultaneously and start struggling for power?

This issue of conflict resolution is in fact a long-standing problem in AI, one that has provoked a great deal of research with decidedly mixed results. Laird, however, was able to make it a non-issue. His first change, in effect, was to teach the demons some manners. Instead of having each one shriek out a command when its conditions were met, he had them express opinions such as "operator Q1 (take your opponent's queen) is better than operator Q2 (take your opponent's pawn)," or "operator Q7 (sacrifice your bishop) is best." At the same time, he modified the production system as a whole to operate like an exceedingly polite business conference. Instead of letting the activated demons fight over who gets to give the orders, Laird simply let *all* of them have their say. Only when everyone's opinions were on the table would the program decide what to do.

In itself, of course, there was nothing in this "elaboration-decision cycle" to keep two different demons from nominating the same operator for both the "best" and "worst," and thus producing the same kind of conflicts as before. However, Laird's approach was to treat such impasses not as a crisis, but as an opportunity. His second big change was a mechanism that he and Newell called

† The name comes from the work of the logician Emil Post, who first referred to condition-action rules as "productions" in the 1940s.

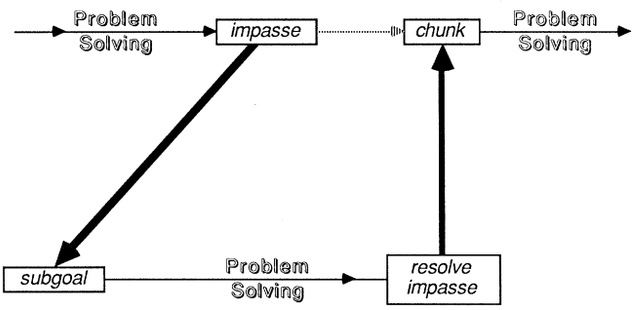
"universal subgoaling." Think of the way a commuter might deal with a traffic jam by turning off on the first available side street and finding a new route home: in much the same way, Laird's program would automatically deal with each impasse by setting up a new problem—"solve this impasse"—and going to work on it.

More than anything else, says Newell, it was universal subgoaling that allowed Soar to overcome the rigidity of the early expert systems. "Conflict resolution is no longer done by a fixed mechanism, but a general one," he says. At every point, "Soar can bring its entire problem-solving apparatus to bear."

At the same time, he says, the elaboration-decision cycle provided exactly the kind of universality that he and Laird set out to achieve. Since Soar only makes its decisions after all the rules have been heard from, it automatically uses the most powerful knowledge it has available. For example, if Soar has no guidance knowledge whatsoever, it will be forced to look at each problem state in turn until it stumbles across the goal. But if the system has at least a little bit of knowledge—that is, if its rules do express a few preferences for one action over another—then it will begin to behave in ways that resemble general methods such as Hill-Climbing or Means-Ends analysis. And if it has a lot of knowledge, so that it always has a clear preference for what to do next, then its behavior will be that of an expert.

Indeed, by the time Laird had completed his thesis project in 1983 there was only one logical gap left in Soar. The program needed a way to learn, so that it could work its way along the novice-expert continuum without having to have its knowledge programmed in by hand. And that, says Newell, was precisely the capability provided by Laird's fellow student, Paul Rosenbloom.

In the beginning, ironically, Rosenbloom's thesis project had little to do with Soar. His goal was to devise a computer program that would model the way humans improve with practice on a variety of tasks, using a learning method known to psychologists as "chunking." Nonetheless, as the work progressed he and Laird became convinced that their approaches were complementary. Along with Newell they even made an informal pact that, once their respective theses were finished, they would fuse their efforts into single research project. And indeed, says Newell, "in one of those great leaps that all researchers dream about, a completely general scheme of chunking was incorporated into Soar in a matter of a day or two." On Monday, 10 January 1984, Laird and Rosenbloom sat down to work. By Wednesday, 12 January, the revised program was up and running.



Learning from experience. *When Soar reaches an impasse in its problem-solving—that is, when it does not know what to do next—it automatically sets up a subgoal to resolve the impasse. When it succeeds, it goes back to where it left off and simultaneously encodes a new "chunk" of knowledge that will keep it from ever having to suffer that particular impasse again.*

The basic idea of chunking is simple: whenever Soar resolves an impasse, it remembers how. More precisely, Soar encodes the results of its problem-solving as a new condition-action rule—a "chunk"—and then stores it away in memory where it operates like any other rule. Its conditions are the relevant contents of working memory at the time the impasse arose; its action is the new solution.

If the basic idea of chunking is simple, however, the consequences are large: SOAR's problem-solving is now cumulative. It does not have to keep reinventing the wheel. The next time it encounters a similar impasse, it can leap directly to the solution without repeating the intervening steps. "The next time," says Newell, "it never even *sees* the impasse." And thus SOAR can spontaneously pass from the slow, painful, trial-and-error problem-solving characteristic of a novice, to the near-instantaneous insight characteristic of an expert. It does not have to be programmed with expertise, because it can learn expertise.

By this point it was clear to Newell, Laird, and Rosenbloom alike that Soar was more than just another thesis project. Granted, it was not the first AI program able to grapple with hard intellectual tasks. Nor was it the first AI program able to learn from experience. But it combined those abilities in a remarkably elegant way. More important, its very generality made it a natural framework for integrating and synthesizing the results of earlier AI research. And in the years since 1984, Newell, Laird, and Rosenbloom have spent a great deal of their time doing just that. Newell has put a small army of younger graduate students to work on Soar-related projects at Carnegie Mellon, while Laird and Rosenbloom are doing the same at Michigan and Southern California. The Soar project as a whole now involves some 35 individuals nationwide.

One of the group's first goals was to test the Soar architecture by reimplementing a variety of classic AI programs. In each case they left the basic architecture untouched; the only thing they changed was the set of rules, which gave Soar the appropriate knowledge about the problem at hand. The results were as successful as they could have hoped. Soar has followed in the steps of General Problem Solver by solving the Tower of Hanoi, the Missionaries and Cannibals problem, and similar puzzles. It has demonstrated its ability to use a whole range of general reasoning methods, including Means-Ends Analysis and Hill-Climbing. It has proved that it can learn from experience on any given problem—and that it can then reapply that experience to improve its performance on new and different problems. And it has successfully tackled a variety of knowledge-intensive expert system tasks in incarnations such as Neomycin-Soar (medical diagnosis), and Designer-Soar (designing computer programs).

Perhaps the most notable test, however, was R1-Soar, which reimplemented one of the largest and most famous of the practical expert systems. R1 itself, which currently contains thousands of rules, is used by the Digital Equipment Corporation to configure its VAX line of minicomputers. Since R1-Soar was intended as a laboratory test it contained only about 25% of the original's functionality. Nonetheless, within that limited domain its performance was comparable with the orignal. Moreover, because of its chunking ability, R1-Soar was able to start over on a configuration problem that had originally taken it 1731 steps to solve, and then solve it again with only 7 steps. The R1 programmers at Digital were sufficiently impressed that they are now incorporating some Soar-ish features into a major revision of the program.

The lessons learned in such efforts have given rise to several rounds of refinements to the basic Soar architecture—version 5 is almost complete—and have set the stage for more challenging research projects. At the same time, the group's rapidly accumulating experience helped convince Newell that the program had the potential to be a unified theory of human cognition.

■ M. MITCHELL WALDROP