Allen Newell
1927–1992

Photo by Ken Andreyo

In Pursuit of Mind …

The Research of Allen Newell[1]

John E. Laird and Paul S. Rosenbloom

Allen Newell was one of the founders and truly great scientists of AI. His contributions included foundational concepts and groundbreaking systems. His career was defined by the pursuit of a single, fundamental issue: the nature of the human mind. This article traces his pursuit from his early work on search and list processing in systems such as the LOGIC THEORIST *and the* GENERAL PROBLEM SOLVER*; through his work on problem spaces, human problem solving, and production systems; through his final work on unified theories of cognition and* SOAR*.*

Allen Newell's research career was defined by his pursuit of a single question. It is the ultimate scientific question underlying psychology and AI as well as a substantial part of philosophy: What is the nature of the mind? Newell's autobiography (American Psychological Association 1986) puts it thusly:

> The central line of Newell's research has remained always the quest for understanding the nature of the mind. The detailed analysis of protocols, the development of production systems, pulling together the theory of human problem solving (in the book of the same name, with Herb Simon), the development of the notion of cognitive architecture, the problem-space hypothesis, a theory of how humans acquire cognitive skills, work on artificial intelligence systems for doing demanding intellectual tasks (such as discovering algorithms), the development of a complete architecture for intelligence—these are some of the main stepping stones. They comprise various mixtures of artificial intelligence and cognitive psychology, as chance and opportunity would have it. This central question will occupy Newell for the rest of his research life, no doubt. (He seems quite incapable of imagining another.) (p. 348)

What Newell does not mention in his autobiography is the magnitude of his stepping stones. Each one of Newell's contributions is not just a small increment in our understanding of the nature of the mind but a major building block of AI and psychology. Newell's research style was to never hold back but to go for the tough problems that would have a high payoff. As figure 1 summarizes, Newell and his colleagues have been responsible for establishing many of the concepts that are central to AI and cognitive psychology, including heuristic search, list processing, problem spaces, weak methods, production systems, chunking, physical symbol systems, and the knowledge level. During his pursuit of the nature of the mind, Newell accumulated a wealth of honors that acknowledge his contributions to AI and psychology as well as his contributions to some of his diversions (see figure 2). Although one of the striking things about Newell's research is his focused pursuit of the mind, along the way he had his share of diversions, as is evident in figure 1. These diversions were not on the main path but nevertheless helped to shape his journey.

This article reviews Newell's research career, starting with symbolic computation in 1954 through the present and his involvement with SOAR and its ramifications. This summary barely scratches the surface of his career as a scientist and cannot convey the spirit and passion that he brought to his work. It excludes his contributions as a senior statesman in establishing and nurturing the fields of cognitive science and AI. It does not convey the significant effort he contributed to building and maintaining one of the premiere computer science departments in the world. It gives only a glimpse of Allen Newell as teacher and mentor.

Preparation

Allen Newell did not grow up planning to devote his adult life to pursuing the nature of the mind. At 17, he wanted to become a forest ranger. In 1945, when 19, he was drafted into the United States Navy and witnessed the atomic bomb tests on the Bikini Atoll from a ship carrying scientists who were there to observe the blasts. After the tests, his job was to make maps of the distribution of radiation over the atolls. When he returned from the Navy, he had decided on a career in science, and he attended Stanford University, majoring in physics. He engaged in undergraduate research in x-ray optics, with his first publication coming from this work (Newell and Baez 1949).

Although physics was his major, Newell took a class from George Polya entitled Mathematical Methods in Physical Science. Polya

Opposite: Allen Newell playing chess with Herbert Simon

National Medal of Science, 1992

National Academy of Sciences, elected 1972
National Academy of Engineering, elected 1980

Harry Goode Memorial Award, American Federation of Information Processing Societies, 1971
A. M. Turing Award (with H. A. Simon), Association for Computing Machinery, 1975
Alexander C. Williams Jr. Award (with William C. Biel, Robert Chapman and John L. Kennedy), Human Factors Society, 1979
Distinguished Scientific Contribution Award, American Psychological Association, 1985
Award for Research Excellence, International Joint Conference on Artificial Intelligence, 1989
Emanuel R. Piore Award, Institute for Electrical and Electronic Engineers, 1990
Franklin Institute's Louis E. Levy Medal, 1992

Doctor of Science (Honorary), University of Pennsylvania, 1986
Doctor in the Behavioral and Social Sciences (Honorary), University of Groningen, The Netherlands, 1989

First President, American Association for Artificial Intelligence, 1980

John Danz Lecturer, University of Washington, 1971
American Academy of Arts and Sciences, elected 1972
John Simon Guggenheim Fellow, 1976-77
Computer Pioneer Award, Charter Recipient, IEEE Computer Society, 1982
William James Lectures, Harvard University, Spring 1987
William James Fellow Award (charter recipient), American Psychological Society, 1989

Figure 1. Allen Newell's Honors

was a distinguished mathematician, and the class covered the material in the now classic *How to Solve It* (Polya 1945). Polya was interested in problem solving, and his book attempted to teach a method based on heuristic reasoning. Newell became fascinated by Polya, and he took every class that Polya taught (as a freshman in physics!), even classes in interpolation and numeric integration and differential geometry. He never developed a personal relationship with Polya, and Polya's work did not immediately impact Newell's career (as evidenced by the lack of any reference to Polya's work during the early work on the LOGIC THEORIST [LT] and the GENERAL PROBLEM SOLVER [GPS]). Newell (1983) reflected on this lack of explicit influence on his own life in an analysis of Polya's influence on problem solving in AI. One suspects, as did Newell, that Polya affected his research implicitly for years to come as Newell attempted to discover the precise details of human problem solving.

From Stanford, Newell went to Princeton in 1949 for graduate studies in mathematics. Once there, a graduate career in pure mathematics did not excite him, and for a time, he contemplated (we don't know how seriously) dropping out of graduate school and raising sheep in New Zealand. While at Princeton, Newell worked with Oskar Morgenstern on game theory and logistic models. By the end of his first year, he abandoned graduate school for RAND Corporation, the Santa Monica think tank that was one of the few places doing work in game theory.

Once at RAND, Newell concentrated on logistics systems and organizational science. He worked with Bob Chapman, Bill Biel, and John Kennedy (with whom he later shared the Alexander C. Williams, Jr., Award, from the Human Factors Society) within the Systems Research Laboratory (SRL). The task of SRL was to study interactions between humans and machines to improve training in the military. Their experimental test bed was a full-scale mock-up of an Air Defense Early Warning Station (no computers!), where decisions had to be made about sending up planes to investigate radar sightings. For various technical reasons, real radar was not used, but simulated displays of radar blips were produced on an IBM, predigital computer programmable calculator. These simulated radar maps were then printed on paper by an old-fashioned 80-column IBM printer. Producing this simulation was one of Newell's responsibilities in the lab, and he did it in collaboration with J. C. (Cliff) Shaw, an actuary turned system programmer who worked

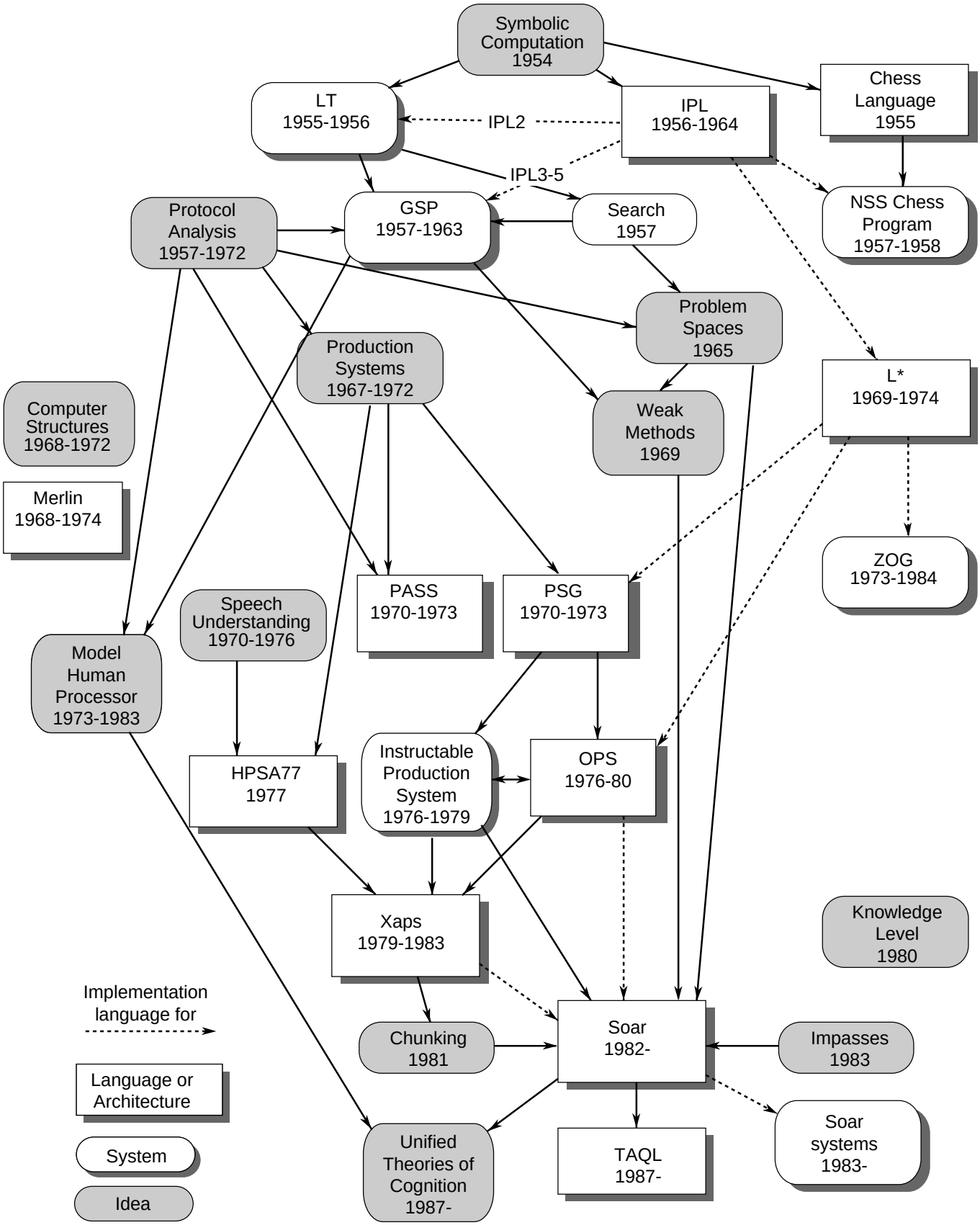


Figure 2. A Summary of Allen Newell's Research.

The way to learn about systems is to try to build one

in RAND's computation department and whose interests were in programming languages and operating systems. The lab was designed so that all phone interactions between crew members were tape recorded and subsequently analyzed to investigate the patterns of interaction among people and their decision-making processes; a precursor to Newell's later work on the analysis of human thinking-aloud protocols.

At the beginning of 1952, shortly after the lab came into operation, Newell met Herb Simon, who was recruited by Kennedy and Chapman as a consultant to the activity. Newell had now made connections with those people with whom he would make major contributions in AI. Moreover, the map-producing program he and Shaw had constructed taught them that computers could be used for symbolic, as well as numeric, processing; Simon learned this lesson from exposure to the program. Through other activities going on at RAND (for example, Merrill Flood was building mechanical *turtles*, little robots that rolled about and looked for power outlets to hook up to when their batteries ran low) and their general curiosity about the computers that were just beginning to appear (Newell and his partners were aware of early efforts on computer chess by Claude Shannon and others, and Simon had written an appendix to a technical report that he generated at RAND in 1952–1953 on a program to play chess by heuristic search), the triad was becoming aware of the possibility of using computers as general symbol processors. In the summer of 1954 while they drove to observe some air exercises, Newell and Simon had a long discussion about using computers to simulate human problem solving or the decision making of the airmen in the lab. However, Newell continued to work on organizational science and saw that as his area of research for the foreseeable future.

The Origins of AI

On a Friday afternoon in mid-November 1954, Newell had what he described as a *conversion experience* while he attended a talk by Oliver Selfridge. This talk converted a general interest in the possibilities of computer simulations of symbolic processes into an urgent desire to devote all his energies to this task. What he had been exposed to as an interesting possibility was now revealed to him as a present reality. If he was to participate in this exciting adventure, he must begin at once. Selfridge's talk was on the pattern-recognition system that he was developing with G. P. Dinneen on the Whirlwind MTC computer at the Massachusetts Institute of Technology's (MIT) Lincoln Laboratories. During this talk,

> it was instantly clear to Newell that intelligent adaptive systems could be built that were far more complex than anything yet done and that they would be programmed on digital computers. Imprinting had occurred, and Newell has been working ever since on discovering and constructing the mechanisms that constitute mind. (American Psychological Association 1986, p. 348)

His immediate response was to attempt to program a computer to learn to play chess because of the perceived difficulty of chess as a form of thought. Immediately attempting to program a computer is central to his methodology: The way to learn about systems is to try to build one. He did not take an optimization approach but instead borrowed heavily from ideas in psychology. To blend research in AI and psychology was another hallmark of Newell. Because his goal from the beginning was to understand human thinking (the airmen in SRL or the chess master) and because his key research tool was computer simulation, AI and psychology had to go hand in hand. His work on chess was based squarely on notions of *heuristic search*, that is, search using rules of thumb, which not only were the basis for the way he saw humans attempt to play chess but also were necessary to chop down the enormous problem space to a manageable size. This led to Newell's first publication related to AI: He reported on a design for his chess machine at the 1955 Western Joint Computer Conference (Newell 1955). A chess language was designed in the summer of 1955, but the restricted memory sizes at the time proved to be the stumbling block to building a working system, which clearly frustrated Newell. In his first effort at AI, he believed in the centrality of a working implementation, even if he had not yet achieved it.

> As every design engineer knows, the only difference between a good design and the actual machine is time and effort.... The scheme presented here is not far from a good design.... But this is not sufficient. These mechanisms are so complicated that it is impossible to predict whether they will work. The justification of the present article is the intent to see if in fact an organized collection of rules of thumb can pull itself up by its bootstraps and learn to play good chess. (Newell 1955, p. 108)

Newell had been invited to go to the Center for the Advanced Study of the Behavioral Sciences at Stanford University, but Simon convinced him to move to the Carnegie Institute of Technology in the spring of 1955 to complete a doctoral degree and to work with Simon on simulations. Newell and Simon maintained their connection with Shaw and the computer by telephone, teletype, and travel. All during this time, Newell had been discussing his plans with Simon and Shaw, and in early 1955, Shaw and Simon joined with Newell to build a computer program that would demonstrate complex information processing. Although Newell had made some progress with chess, the three of them looked for something a bit simpler to start with. They initially started trying to build a system for working in geometry but abandoned it to work on propositional logic, in part because Simon owned a copy of *Principia Mathematica*.

LOGIC THEORIST

During 1955, Newell, Shaw, and Simon attempted to create a computer program that could replicate the proofs of Chapter 2 in Whitehead and Russell's (1935) *Principia Mathematica*. On 15 December 1955, Simon performed a hand simulation of the system that demonstrated the feasibility of their enterprise. This research led directly to LT, which on 9 August 1956 created the first mechanical proof of a theorem (theorem 2.01) (Newell and Simon 1956b; Newell, Shaw, and Simon 1957).

LT was one of the first working AI programs, and it incorporated many of the ideas that, over the years, have become the foundation of AI. First and foremost was heuristic search, where a problem is solved by a sequence of transformations of symbolic structures. The symbolic structures, called *states*, are theorems, while the transformations, called *operators*, create new theorems by combining, modifying, or decomposing existing theorems. At each state, many different operators could be applied, and general rules of thumb, called *heuristics,* are used in selecting appropriate operators.

LT was also the first program that tried to solve problems the way people solved problems. Thus, it did not try to overcome the difficulty of discovering proofs through either brute-force search or what today is called theorem proving. Newell, Shaw, and Simon were extremely sensitive to the fact that working with logic did not mean that the problem solver had to be deductive, even if the final result was a proof.

> The reader should not be misled by words "proofs" and "logic." Discovering proofs is not a deductive process. Moreover, the fact that the task is one in symbolic logic does not make the problem solving process any more "logical" than if some other task—e.g., writing this book—were involved. (Newell and Simon 1972, p. 105)

INFORMATION-PROCESSING LANGUAGE

While Newell, Shaw, and Simon were attempting to build LT and a chess-playing program, they were faced with the fact that no computer languages supported the symbolic processing that was ubiquitous in their programs. At the time, computers were used to process numbers, not symbols, and most, if not all, programming was done in machine code or assemblers. In what was to be an often repeated task, Newell, Shaw, and Simon were faced with building their own tools. Thus, in early 1956, as part of their work on LT, they developed the first implemented list-processing language: INFORMATION-PROCESSING LANGUAGE (IPL) (Newell and Shaw 1957). IPL was a follow-up to their LOGIC LANGUAGE, which was never programmed for a computer but was used for hand simulations (Newell and Simon 1956a). LT was written in IPL-II, running on the JOHNNIAC computer developed at RAND.

Until 1964, IPLs went through continued development, with a total of six distinct versions being developed.

> The best one of the bunch was IPL-III..., which had no syntactic structure at all. It was beautiful. But [it] was so space intensive that you simply couldn't run it on the machines [which had only 4000 words]. We had to abandon it. What it did was execute every symbol. L* was like it. Each symbol would go back and extract from the program itself the operands in successive places so that you could go do anything you wanted in the syntax because there wasn't any syntax. (A. Newell, September 1991, conversation with John Laird, Univ. of Michigan AI Lab)

Along the way, IPLs introduced many of the ideas that initially became fundamental to list processing and later to computer science in general, including lists, associations, schemas (frames), dynamic memory allocation, data types, recursion, associative retrieval, functions as arguments, and generators (streams). The IPL-V manual (Newell 1961; Newell et. al.

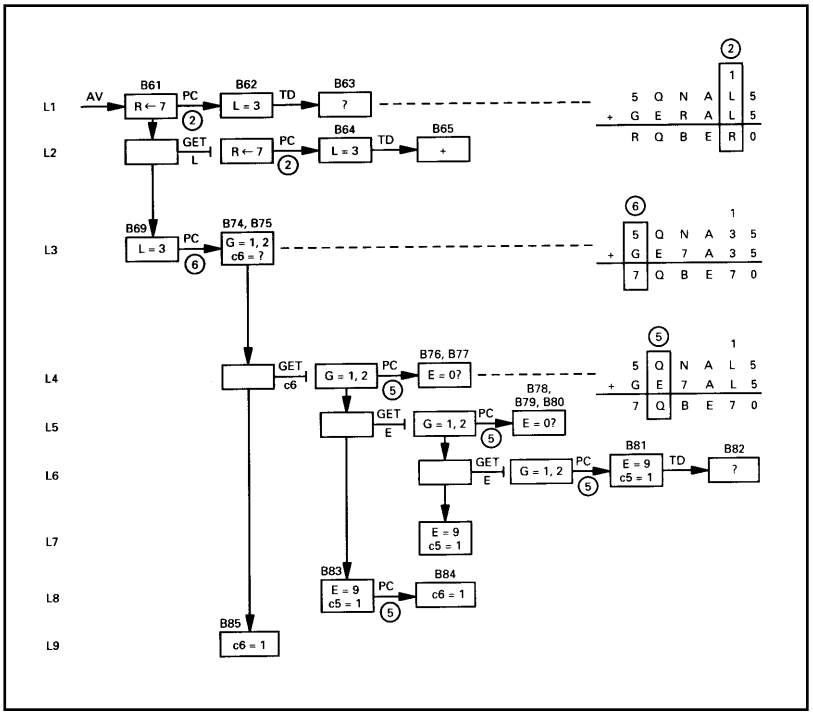


Figure 3. Problem-Behavior Graph during a Complex Situation in Cryptarithmetic (Newell 1990)

1964) advocated a design philosophy for programming that years later would be reinvented independently as structured programming, with such concepts as incorporating top-down design and short subroutines, and avoiding GOTOs (Simon and Newell 1986). IPL suffered from superficial weaknesses in its syntax (derived from the JOHNNIAC assembler) and its requirement that the user explicitly manage the available space. Lisp, developed by John McCarthy at MIT in 1958, included a background garbage collector that reduced the programmer's need to track unneeded data structures and has since become the standard language for list processing in AI (although IPL-V was used extensively on many early AI systems and is still available—even on a personal computer).

Ph.D.

During the original development of LT and IPL, Newell was still working for RAND, although he spent all his time in Pittsburgh. In 1957, Newell generated his thesis by stapling together some of the original papers on chess and LT and received his Ph.D. from the Graduate School of Industrial Administration at the Carnegie Institute of Technology (Newell 1957). His thesis identified ideas that would appear 25 years later in SOAR: dynamic subgoal creation, multiple evaluation methods for controlling search, resource-limited reasoning, learning based on applying existing knowledge, and the close integration of learning and problem solving. In 1961, although tempted to return to RAND and California, Newell decided to stay in Pittsburgh and accepted a position as institute professor at Carnegie Tech. Newell claimed that one of the smartest things he ever did was to never be an assistant professor.

Chess

Toward the end of the work on LT, Newell, Shaw, and Simon returned to chess. Between 1957 and 1958, a chess-playing program, called NSS (named for its authors), was written in IPL-IV (Newell, Shaw, and Simon 1958). Many other chess programs were being developed at the time, and although NSS never did remarkably well, it had many of the innovations that were being developed concurrently in other systems, such as alpha-beta pruning and intermediate goals. The NSS program was distinguished from the others by aiming at the simulation of human players (de Groot 1946). It was a descendant of earlier programs by Simon (from 1953) and Newell (from 1955), and it was independent of the others being developed, which were pure AI. The NSS team was interested in studying the mind, not in building AI systems to show computer smarts.

As was customary with all the tasks that Newell attacked, he did a detailed analysis of the underlying problem space of chess. One of his observations was that there are between 4000 and 5000 different legal moves that can arise in a game of chess. In 1958, this number was large enough so that it was not feasible to have all the moves precomputed. However, 20 years later, a chance discussion of this fact led to an exploration into creating a chess-move generator in which all the 4000+ moves were precomputed and stored in hardware. At each step in the search, the analysis required to generate the legal moves could be done in parallel with tests of whether each of the 4000+ moves independently tested was appropriate to the current situation. This investigation led to the very large-scale integrated move generator of the HI-TECH chess machine.

GENERAL PROBLEM SOLVER

Newell, Shaw, and Simon's work on LT, chess, and IPL gave them experience with the fundamentals of symbolic processing and the

requirements for building simple single-task AI systems. In LT, they had a system that demonstrated intelligent behavior but in a limited domain. All its methods and knowledge were specific to the task of propositional logic. Their inclination was to not build more and more task-specific systems (although many of their students during this time did landmark work in applying heuristic search to a variety of domains [Feigenbaum and Feldman 1963]).

Instead, they were interested in the next major step, which for them was the generalization of these basic techniques so that a single system could model human behavior across many different domains.

> [GPS] was an attempt to create a general intelligent system and also create a theory of human problem solving. It was the first program to have pretensions to generality, because it separated out the program structure for problem solving from the program structure to describe a particular task. (Newell 1992, p. 35)

After comparing LT's behavior to thinking-aloud protocols of human subjects, it was clear that LT's structure was insufficient. The protocols of humans working on the logic task suggested a more goal-directed strategy, where operators were selected on the basis of how well they could help achieve the goal given the current state: means-ends analysis. Was it possible to build a system that could use a few general methods on many different tasks in ways similar to humans?

To answer this question, Newell, Shaw, and Simon extended the methodology that they had been using for LT and NSS. They did not try to create an optimal intelligent system from scratch that could solve only a single problem but instead tried to use what they knew about human behavior to aid in their development of a more general intelligent system. Pursuant to this approach, they resurrected protocol analysis as a technique for exploring human behavior. Protocol analysis had fallen into disfavor during the rise of behaviorism because of its supposed reliance on introspection.

A raw protocol is of limited value until it is analyzed and coded. Although this work might seem more appropriate for a graduate student (and it is an immense amount of work), Newell and Simon analyzed the protocols themselves. Newell claimed that only by getting down into the data is it possible to get the important insights into behavior. This hands-on approach was characteristic of his research style.

In studying these protocols, Newell and Simon developed problem-behavior graphs as a notation for tracking the performance of the subjects. This notation allowed them to code the decisions that were being made by their subjects and, ultimately, to compare the performance of computer systems with humans. Figure 3 shows a small segment of problem solving for the cryptarithmetic problem DONALD + GERALD = ROBERT. The task is to replace each of the letters with numbers, so that the same letter is always assigned the same number; no two letters have the same number; and when the letters are replaced by numbers, the numbers for DONALD and GERALD add up to the number for ROBERT. During this segment, the subject was already given that $D = 5$ and concluded that $T = 0$. The subject then considers $R = 7$ and how that implies that $L = 3$ and $G = 1$ or 2 depending on the carry into the final column, and so on.

By going over and over these protocols, it was obvious that the subjects were often searching the space of legal operators that they could use and that *means-ends analysis*, where actions are selected based on their ability to achieve a goal, was a strong component of the control.

The research on GPS started in 1957. The first flowchart for the organization of GPS was produced in October 1957, including not only means-ends analysis but also hierarchical planning. The first two publications on GPS appeared in 1960 (Newell, Shaw, and Simon 1960a, 1960b), with three more following in 1961 (Newell and Simon 1961a, 1961b, and 1961c). GPS was first presented publicly in talks during 1958, which were, in turn published in 1962 as the first detailed computer model of human problem-solving behavior (Newell, Shaw, and Simon 1962).

As mentioned previously, the general method that GPS used was means-ends analysis, a commonsense approach where operators (the *means*) are selected to reduce the differences between the current situation (the *state*) and the goal (the *ends*) of the problem. As a simple example, consider that someone is at the School of Computer Science at Carnegie Mellon University (CMU-SCS) in Pittsburgh, Pennsylvania, and wants to get to the offices of the American Association for Artificial Intelligence (AAAI) in Menlo Park, California. Here, the current state is being at CMU-SCS, and the goal is to be at AAAI.

The system's ability to solve a problem is determined in large part by the problem spaces and operators that the system has available. As mentioned earlier, the operators are the ways the system can move through

GPS was an attempt to create a general intelligent system and … a theory of human problem solving

the problem, and the problem space consists of the operators and the set of possible states that can be visited. If an operator takes us immediately from our current state to the goal, the problem is trivial. If many operators can apply to each situation, and the problem requires a long sequence to achieve the goal, the problem is much more difficult. In our example, the problem space contains all the possible locations of a person in the world, and the operators are actions, such as walking; driving a car; taking a bus, cab, train, ship, or airplane. The complete, expanded problem space does not exist explicitly in the world (the person is only at one location and, thus, one state at a time), nor does it have to exist explicitly within the person as a data structure. Instead, the states of the problem space are generated dynamically as operators are applied, so that during problem solving, only a small subset of the states are ever generated.

Many other activities might be involved in solving our example problem, such as buying tickets and getting money, that expand the set of operators. The problem space is similarly expanded to include not only the location of the person but also whether he/she has a ticket, how much money he/she has, and so on. Other problems have different problem spaces and operators. In cryptarithmetic, the states include partial assignments of numbers to letters in a puzzle, such as *DONALD* + *GERALD* = *ROBERT,* and the operators either assign numbers to letters or test that the assigned numbers are consistent with the constraints in the puzzle (such as $D + D = T$). Even a single problem can be attacked using different problem spaces. Cryptarithmetic becomes an easier problem if the problem space is expanded to include the carries for the columns, sets of potential numbers for given letters, and the parity of the number for a given letter (even or odd). The operator set is similarly expanded to include operators for summing columns, computing carries, proposing and pruning sets of numbers for a letter, and so on.

By using the common framework of problem spaces to represent any problem, the activity of solving a problem can be reduced to the problem of selecting and applying appropriate operators. In GPS, the operators are selected in a two-step process. The first step is to compute the differences between the current state and the goal, and the second step is to select the operator that best reduces the most important difference. In our example, the initial difference would be that the person is more than 2000 miles from his/her destination. With this difference, an operator is selected that can best reduce this difference. In our example, the operator would most likely be to take a commercial airline flight.

Once an operator is selected, GPS attempts to apply it to modify the current state. If a person is in a room, and an operator is selected to walk to the door, the operator applies immediately, and the current state then has the person at the door. However, in many cases, the operator selected cannot apply immediately. Remember that the operator is selected to reduce the difference to the goal, and at this point, the ability to apply the operator to the current state is ignored. Sometime, the selected operator cannot apply, as in our example, where a person at CMU-SCS cannot immediately take an airplane because he/she is not at the airport, and he/she does not have a ticket.

In such situations, GPS considers the inability to apply the current operator as a subproblem and recursively attempts to solve it. In our example, the system would now attempt to get to the airport and buy a ticket using the same method as the original problem. This approach was used by GPS to solve a variety of different problems (Ernst and Newell 1969) and has become a staple for AI planning systems, with STRIPS being the classic example (Fikes and Nilsson 1971).

However, Newell and Simon knew that simple means-ends analysis, as modeled by the initial GPS, wasn't the whole story. In logic problems, the subjects would slip into short bursts of abstract problem solving, where they ignored many of the details of an axiom as they tested its usefulness. Once a short partial plan was created, the subjects would immediately return to fill in the details. In the travel example, a person might create a complete plan for driving and flying to Menlo Park without ever worrying about the details of walking and buying tickets. The original GPS could not create this complete initial plan and then later refine it.

Observations from the human protocols led to an extension to GPS in which a problem would first be solved completely at an abstract level, ignoring details that would be easy to fill in later. The solution at the abstract level would then serve as a guide for solving the problem with all the details. This system, called PLANNING GPS (Newell and Simon 1972), was the first planning system to use multiple levels of abstraction, predating a similar approach used in ABSTRIPS (Sacerdoti 1974). However, none of these systems ever matched the flexibility seen in the protocols where the humans switched back and forth between abstraction and ground-level opera-

tors during the problem solving.

Another shortcoming of pure means-ends analysis was noticed in cryptarithmetic, where many of the control decisions were not based solely on the difference between the current situation and the goal but could best be characterized as independent pieces of knowledge that were conditional on the current situation. Thus, instead of encoding knowledge as a fixed sequence of behavior, as in standard programming languages, or even as a fixed method, such as means-ends analysis in GPS, knowledge should be encoded as small individual units that are not explicitly called but that determine for themselves when they should apply.

Newell and Simon also observed that the subroutines of GPS, ignorant of higher-level context, would grab hold and dig into a deeper and deeper hole (like depth-first search) (Newell 1962). From the combination of observations of human behavior and frustrations with the control structure of GPS came the identification of if-then rules as a notation for describing the behavior. Rules in this form were originally called *productions* (with a computational system based on them being called a *production system*) by E. L. Post (1943) when they were used for work in symbolic logic.

> To wit, you build problem behavior graphs. [Then] you say, "Let's build simulation programs." So you sort of say, "What are we going to have? We are going to have an operator at each node of this problem behavior graph." That is what the problem behavior graph shows. [Then] you say, "Let's write at each node the collection of conditions on the state." Now you are sitting just about where you say, "Well gee, if I just wrote what are the conditions and here's the operations, and then I'll just gather them altogether, and then I have a production system." That's actually how it happened. From my point of view, it was derived entirely from the data analysis via problem behavior graphs as the right programming schema to make it clear whether in fact the conditions were the same or different at different nodes and so forth and then you can build a programming language. As soon as that happens, sitting in this [computer science] environment,...you say, of course these are in fact like Floyd productions, like Post productions. (A. Newell, September 1991, conversation with John Laird, Univ. of Michigan AI Lab)

Newell and Simon had been familiar with productions for some time, given Selfridge's (1959) related work on PANDEMONIUM for perceptual processing and Floyd's (1961) use of production systems in algorithm design. Surprisingly, even the metaphor of blackboards, usually considered a derivative of production systems (leading to the creation of HEARSAY II [Erman et al. 1980] and the use of blackboards as a control structure within AI [Nii 1986]), was being used by Newell.

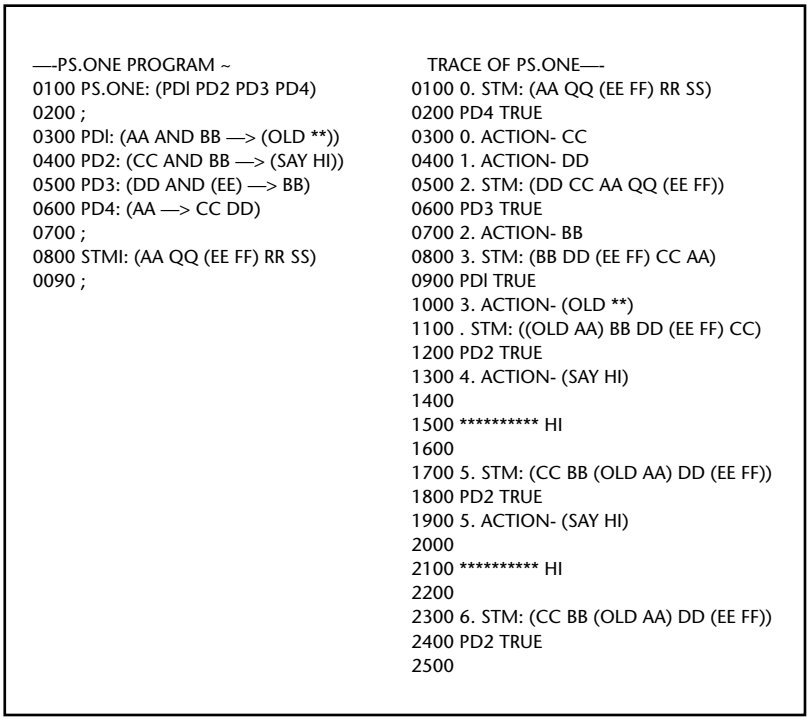
> Metaphorically we can think of a set of workers, all looking at the same blackboard: each is able to read everything that is on it, and to judge when he has something worthwhile to add to it. This is just that of Selfridge's PANDEMONIUM: a set of demons, each independently looking at the total situation and shrieking in proportion to what they see that fits their natures. It is a much easier mode of organization to use when the processes are only perceptual—that is, look at, but do not change, the environment—than with active problem-solving where the workers are busy writing and erasing as they are reading. (Newell 1962, p. 13)

GPS had other weaknesses, including that the initial formulation of the problem had to be provided by a programmer. A key insight in solving any problem is knowing which operators (such as movement, ticket buying) and which aspects of the situation (such as location, possession of tickets, amount of money) to include and which to ignore. The issue of determining the appropriate problem space was never addressed directly by GPS, although Simon later addressed it in the UNDERSTAND project (Hayes and Simon 1974, 1976), as would Newell within SOAR (Newell 1990; Newell et al. 1991). Many other aspects of GPS would later be generalized in SOAR (Newell 1992), a problem-solving architecture based on problem spaces and production systems (Laird, Rosenbloom, and Newell 1986c).

PLANNING GPS was the first planning system to use multiple levels of abstraction

Human Problem Solving

The book *Human Problem Solving* (Newell and Simon 1972) was the culmination of Newell and Simon's investigations into complex problem solving. It brought together their studies of protocol analysis, GPS, and production systems, presenting a computational theory of human problem solving based on heuristic search in problem spaces. It reiterated many earlier themes, such as the importance of analyzing the exact nature of a task and suggesting that human behavior is determined in large part by the constraints of the task (analogous to Simon's [1969] famous ant



```
—-PS.ONE PROGRAM ~
0100 PS.ONE: (PDl PD2 PD3 PD4)
0200 ;
0300 PDl: (AA AND BB —> (OLD **))
0400 PD2: (CC AND BB —> (SAY HI))
0500 PD3: (DD AND (EE) —> BB)
0600 PD4: (AA —> CC DD)
0700 ;
0800 STMI: (AA QQ (EE FF) RR SS)
0090 ;
```

```
TRACE OF PS.ONE—-
0100 0. STM: (AA QQ (EE FF) RR SS)
0200 PD4 TRUE
0300 0. ACTION- CC
0400 1. ACTION- DD
0500 2. STM: (DD CC AA QQ (EE FF))
0600 PD3 TRUE
0700 2. ACTION- BB
0800 3. STM: (BB DD (EE FF) CC AA)
0900 PDl TRUE
1000 3. ACTION- (OLD **)
1100 . STM: ((OLD AA) BB DD (EE FF) CC)
1200 PD2 TRUE
1300 4. ACTION- (SAY HI)
1400
1500 ********** HI
1600
1700 5. STM: (CC BB (OLD AA) DD (EE FF))
1800 PD2 TRUE
1900 5. ACTION- (SAY HI)
2000
2100 ********** HI
2200
2300 6. STM: (CC BB (OLD AA) DD (EE FF))
2400 PD2 TRUE
2500
```

Figure 4. Example Production System (PS.ONE) Run Using PSG (Newell 1973a).

on the beach) together with the human's goals and knowledge of the structure of the task. Most of *Human Problem Solving* was based on work that was done during the late fifties and early sixties. The book itself was started in the late fifties. Needless to say, the writing of the book took much longer than either Newell or Simon expected, and when the work on production systems started to gain momentum in the late sixties, they had to resist the temptation to rewrite *Human Problem Solving* using production systems as the underlying representational formalism.

Simon and Newell remained fast friends until Newell's death, but they never undertook another joint research project as significant as *Human Problem Solving*. There was no rift between them, but the lack of close collaboration was probably a natural drift as each built a research strategy to reflect differing beliefs about the source of the biggest payoff for studying intelligence. Simon (1991) successfully pursues

> programs such as GPS, EPAM, the sequence extrapolator, and BACON, which simulate human behavior over a significant range of tasks but do not pretend to model the whole mind and its control structure. (p. 328)

Newell's (1992) emphasis was on the computational architecture and attempts to model the control structure underlying intelligence.

> An architecture is a fixed set of mechanisms that enable the acquisition and use of content in a memory to guide behavior in pursuit of goals. In effect, this is the hardware-software distinction: the architecture is the hardware that is supporting the software and the software is the collection of data structures that encode the content. This is the essence of the computational theory of mind. (Newell 1992, p. 27)

One of Newell's (1973b) classic papers, "You Can't Play 20 Questions with Nature and Win," written for the 1973 CMU symposium on visual information processing, goes to the heart of his philosophy on research in pursuit of the mind. Newell put forth the hypothesis that cognitive psychology should reconsider its attempt to carry out experiment after experiment to peel back the structure of human cognition. Newell proposed that sufficient data already exist to pin down much of the structure of the mind. Thus, theories should be built that cover a wide range of phenomena. This line of argument led to his call for unified theories of cognition, and he was so bold as to claim that there exist enough psychological data, when taken together, to serve as sufficient constraint on unified theories. Such a statement was not to deny his love for data and, most likely, was a bit tongue in cheek.

Production Systems

Newell and Simon's work on production systems, which was the first use of production systems for AI and cognitive modeling, began in the mid- to late sixties. Although Newell was working intensely on production systems during the sixties, he did not publish an article on them until 1972 (Newell 1972; Newell and Simon 1972). However, he gave a series of lectures at Stanford in 1967 on production systems as control architectures that proved to be influential. Bruce Buchanan and Ed Feigenbaum (a former student of Simon's and a collaborator with Newell on IPL-V) attended his talk and were left with the insight to use productions as a representation for the knowledge of their DENDRAL program (Lindsay et al. 1980), the progenitor of all expert systems.

Although the idea of production systems is relatively straightforward—long-term memory consists of a set of productions that are continually matched against the system's current situation represented in its working

memory—the space of possible production-system architectures is huge, with many different alternative schemes for representing productions and working memory, matching productions to working memory, and selecting between competing productions. In the early seventies, a series of experimental languages were developed to study different possible design decisions for production-system languages (Newell 1973a). The final version, called PSG (production-system version G) is the direct precursor to the official production-system (OPS) languages (Forgy and McDermott 1977).

Figure 4 shows a simple production system and its behavioral trace, as implemented in PSG. There are four productions (PD1–4), and working memory initially contains (AA QQ (EE FF) RR SS). PSG used production order to determine which production would fire if more than one matched working memory.

The early development of production systems during the late sixties and early seventies was in parallel with continued work in protocol analysis. Protocol analysis was extremely time consuming, and as with any computer scientist, once faced with a difficult manual process, there is an urge to automate it. Thus, Newell attempted to build an automatic protocol analysis tool with Don Waterman (Waterman and Newell 1971). The result was the creation of the PASS systems. The goal of PASS-I was to go from natural language to a problem-behavior graph, and it achieved some level of competence for cryptarithmetic. The follow-up program, PASS-II, was to be an adaptive production system that was aided by a human in doing the analysis. However, a useful working program never emerged.

The work in PSG opened the question of whether a production system could become a complete programming system. Mike Rychener (1976) took up this challenge as his thesis with Newell and developed the PSNLST production system in which he reimplemented many classic AI systems. Although this work was successful, it did not address one of the fundamental challenges of production systems: Can very large production systems, with tens of thousands of productions, be built and operated successfully? This challenge was in terms of both underlying technology and the knowledge encoded in the production systems. For the underlying technology, there were questions about whether a production-system architecture could support matching large numbers of productions efficiently. For the knowledge encoded in the system, there were questions about how this amount of knowledge could be acquired and then, once acquired, how a human could possibly understand thousands of productions. These interests led to the creation of the Large Production-System Project in 1975. By 1976, the concern for knowledge acquisition and maintenance led to the hypothesis that the only way to build large production systems was to eschew programming and writing of individual productions and base all knowledge acquisition on instruction. Thus, the project quickly became the Instructable Production System (IPS) Project (Rychener and Newell 1977; Rychener 1980, 1983), which included Charles Forgy, John McDermott, Mike Rychener, Kamesh Ramakrishna, Pat Langley, John Laird, and Paul Rosenbloom.

Although this project produced many different systems, none of them could take general instructions, and none grew to a reasonable size. However, even though the project was described by Newell as "a first-class failure," it spawned a wealth of research:

First, the OPS family of languages (1–7) and the RETE match algorithm were built to support the IPS project (Forgy 1981). OPS5 (Forgy 1981; Brownston et al. 1984) became one of the most widely used production systems and is the forerunner of OPS83 (Forgy 1984), KNOWLEDGE-CRAFT, CLIPS, and SOAR.

Second, R1/XCON was developed by John McDermott (1982) after working on IPS. Sam Fuller of Digital Equipment Corporation, a former faculty member at CMU, suggested using an AI system to help with the configuration of computer systems. A secondary motivation was some friendly taunting by Ed Feigenbaum and his colleagues in the Heuristic Programming Project at Stanford University because the work on production systems at CMU had never been tested on real tasks. R1, implemented originally in OPS4 and then rewritten in OPS5, was one of the first AI systems used commercially on a day-to-day basis. Years later, a reimplementation of R1 in SOAR, called R1-SOAR (Rosenbloom et al. 1985a), led to the creation of a new language for building expert systems at Digital, called RIME (Soloway, Bachant, and Jensen 1987).

Third, work on using problem spaces and weak methods as an organizing principle for production systems grew out of frustration with the lack of a higher-order organization for productions, which, in turn, led to the creation of SOAR in 1982.

MERLIN

Production systems were not the only representation of knowledge that Newell investi-

Herbert Simon Remembers Allen Newell

I must say something about my closest partner in the venture, who remains a close associate and friend to the present day. Although this volume is speckled with vignettes of various of my associates, I can provide only a scanty one of Al. Our paths have meshed so intricately over such a long time that telling about our collaboration and friendship would require writing another book. I will, however, say a bit about the young Al Newell, as I first encountered him.

When I first met Al at RAND in 1952, he was twenty-five years old, and fully qualified for tenure at any university—full of imagination and technique, which is what it takes to make a scientist. I suspect he was ready for tenure soon after birth, Athena springing fully armed from the brow of Zeus. His energy was prodigious, he was completely dedicated to his science, and he had an unerring instinct for important (and difficult) problems. If these remarks suggest that he was not only bright but brash, they are not misleading.

His earliest and probably most important education as a cognitive psychologist came when, an undergraduate physics major at Stanford, he took several courses from the distinguished mathematician George Polya, who recorded many of his ideas about problem solving in a widely used book called *How to Solve It* (1945). Polya introduced Al to the word *heuristic* and to the idea behind that word. A year as a graduate student in the rarefied atmosphere of the Mathematics Department at Princeton convinced Al that his interests lay in applied, rather than pure, mathematics, and that for the immediate future he wanted to be involved in hands-on research rather than graduate studies. He then accepted a position at RAND, where I found him.

If imagination and technique make a scientist, we must also add dollars. I learned many things in the postdoctoral training I took with Al, few more important than how to position the decimal point in a research proposal. My first lesson came from the Systems Research Lab, a grandiose project if there ever was one outside physics and space science. Al and his three colleagues simply took it for granted that it was reasonable for the air force to build an entire simulated air defense station and to staff it for years with an air force unit, enlisted men and officers. It was, indeed, reasonable, but I am not sure that would have occurred to me before I saw it happen.

Thinking big has characterized Al's whole research career, not thinking big for bigness' sake, but thinking as big as the task invites. Al learned about research funding through his early association with physicists, and it is a lesson that we behavioral scientists still need to study with him. (He has been teaching us this lesson at Carnegie Mellon University, with the funding of research in cognitive science and artificial intelligence and, more recently, with the computer networking of our campus.)

From our earliest collaboration, Al has kept atrocious working hours. By this I don't mean that he is more of a workaholic than I am—perhaps a dead heat—but that he works at the wrong time of day. From the start, he preferred sessions that began at eight in the evening and stretched almost to dawn. I would have done most of my day's work by ten that morning, and by ten in the evening was ready to sleep, and not always able not to.

Perhaps his greatest pleasure (at least as judged by his behavior) is an "emergency" that requires him to stay up all night or two consecutive nights to meet a deadline. I recall his euphoria on our visit to March Air Force Base in 1954, when the air exercise extended over a whole weekend, twentyfour hours per day.

Some of these memories are frivolous, but high spirits, good humor, and hard work have characterized my relations with Al from the beginning. We have not been closely associated in joint research projects since the mid1960s, certainly since our book *Human Problem Solving* appeared in 1972. But, however much our working paths have diverged, we still find, whenever we are together, that remarkable community of beliefs, attitudes, and values that has marked our association from the first ten minutes of meeting in February 1952.

In describing our style of work during the years, especially from 1955 to the early 1960s, when we met almost daily, I will paraphrase an interview I gave Pamela McCorduck about 1974, when she was writing *Machines Who Think*. It worked mostly by conversations together. Al probably talked more than I; that is certainly the case now, and I think it has always been so. But we ran those conversations with the explicit rule that one could talk nonsensically and vaguely, but without criticism unless you intended to talk accurately and sensibly. We could try out ideas that were half-baked or quarter-baked or not baked at all, and just talk and listen and try them again.

Aside from talking shop, Al and I have frequently shared our personal concerns and problems. And after Lee Bach left Pittsburgh, Dorothea and I designated the Newells in our wills as guardians of our children, an indication of the closeness and trust we felt toward them. But mainly, we talk about our research, except sometimes when dining or socializing with Noel, Al's wife, and Dorothea.

Whatever hobbies and recreations we have outside our work, we have pursued separately. My own guess is that, when together, we would not resist taking up again those issues that are central to the lives of both of us—our science. The content of our talk would vary little, whether climbing a mountain together or talking in Al's study or my living room.

From *Models of My Life* , pages 199–201. (New York: Basic Books., ©1991. Reprinted with permission)

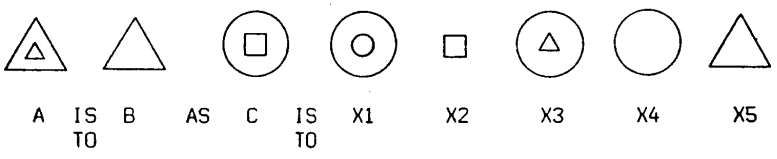
gated in the late sixties and early seventies. Together with Jim Moore and Richard Young, Newell worked on a system called MERLIN that had analogy as its core method (Moore and Newell 1974). MERLIN originally started as an AI tutoring system that was supposed to teach CMU graduate students about the various weak methods in AI. Newell had been pursuing research in general methods for problem solving based on the experience from GPS and protocol analysis, which, in turn, led to a study of the weak methods and the paper on ill-structured methods (Newell 1969). Thus, MERLIN was to be a system that knew all about the methods and other topics in AI.

> MERLIN was originally conceived (with the name CSA, standing for almost nothing) in 1967 out of an interest in building an assistance-program for a graduate course in artificial intelligence. The task was to make it easy to construct and play with simple, laboratory-sized instances of artificial intelligence programs. Because of our direct interest in artificial intelligence, the effort transmuted into one of building a program that would understand artificial intelligence—that would be able to explain and run programs, ask and answer questions about them, and so on, at some reasonable level. The intent was to tackle a real domain of knowledge as the area for constructing a system that understood. (Moore and Newell 1974, p. 201)

Like all his previous architectures, it was process oriented but this time with assimilation through analogy at its core. The basic processing was "the construction of maps from the structure that represents what MERLIN knows to be the structure that MERLIN seeks to understand" (Moore and Newell 1974). From today's perspective, it looked much like a frame (schema) system, and from this perspective MERLIN was one of the first. The basic representational unit, the beta structure, represented a concept *X* in terms of some other concept *Y* with further specification.

Figure 5 shows a simple analogy problem, where the subject is to pick out which of *X*1–*X*5 fits in with *C* analogously to the way *B* is related to *A*. In this case, *X*4 is correct. These objects are represented in beta structures by a list of features, where features can be other objects as well. The problem is then represented by *X*: [*C B*/*A* #], meaning that *X* can be seen analogous to *C* in the same way that *B* is analogous to *A*.

MERLIN contained many new ideas before they became popular in mainstream AI, such as attached procedures, general mapping,



β-structure representation:

A: [TRIANGLE LARGE [TRIANGLE SMALL INSIDE]]
B: [TRIANGLE LARGE]
C: [CIRCLE LARGE [SQUARE SMALL INSIDE]]
X1: [CiRCLE LARGE [CIRCLE SMALL INSIDE]]
X2: [SQUARE SMALL]
X3: [CIRCLE LARGE [TRIANGLE SMALL INSIDE]]
X4: [CIRCLE LARGE]
X5: [TRIANGLE LARGE]

TRIANGLE: [PLANE-FIGURE TRIANGLE-NESS]
CIRCLE: [PLANE-FIGURE CIRCLE-NESS]
SQUARE [PLANE-FIGURE SQUARE-NESS]

X: [C B/A#]
→ [C [A [[TRIANGLE SMALL INSIDE] NULL]]#]
→ [C [[TRIANGLE SMALL INSIDE] NULL]]
(From the posit: TRIANGLE-NESS/SQUARE-NESS, we conclude that [TRIANGLE SMALL INSIDE]/[SQUARE SMALL INSIDE].
→C [[SQUARE SMALL INSIDE]/NULL]]
→ [CIRCLE LARGE]

X/Xi → (fail) for i = 1, 2, 3 & 5.
But:
X/X4 —> [CIRCLE LARGE] (success!)

Figure 5. Geometric Analogy Problem Represented in MERLIN (Moore and Newell 1974).

indefinite context dependence, and automatic compilation. MERLIN would automatically compile its solutions to multistage mapping problems into a single step, a precursor to chunking in SOAR. MERLIN was a reaction against the brittleness of existing AI systems that had sharp boundaries between the problems with which they were successful and those with which they failed. Instead of blaming the discrete character of digital computation for the brittleness, Newell and Moore attempted to avoid brittleness by allowing elastic mappings between representations, working with the character of the processing and not the declarative representation.

Even with all its innovations, by the end of the project, Newell regarded MERLIN as a failure. It was a practical failure because it never worked well enough to be useful (possibly because of its ambitious goals), and it was a scientific failure because it had no impact on the rest of the field. Part of the scientific failure can be attributed to Newell's belief that it was not appropriate to publish articles on incomplete systems. Many of the ideas in MERLIN could have been published in the late sixties, but Newell held on, waiting until these ideas could be embedded within a complete running system that did it all; the only publication was many years after the initial work

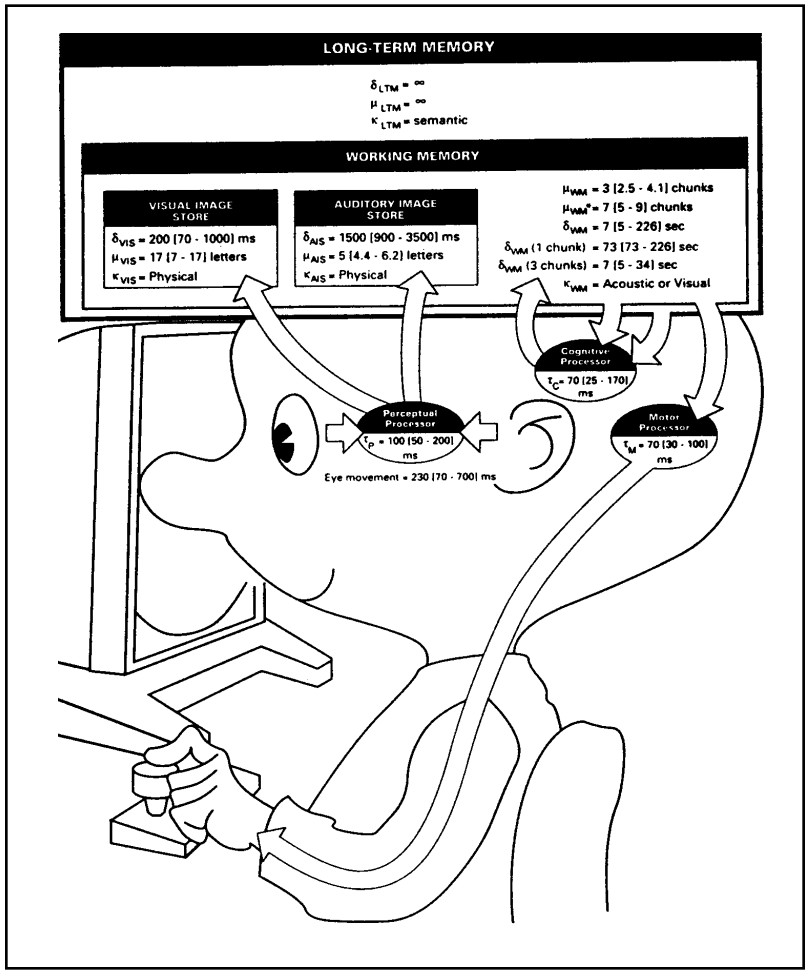


Figure 6. The MODEL HUMAN PROCESSOR *Block Diagram (Card, Moran, and Newell 1983)*

was completed (Moore and Newell 1974). Unfortunately, the team could never make it all work the way it had been envisioned, and the final paper that describes MERLIN requires three readings (at least for us) to understand "How Can MERLIN Understand?" Needless to say, MERLIN slipped into obscurity.

The Psychology of Human-Computer Interaction

As the work on human problem solving matured, Newell looked for applications of applied cognitive science (and an excuse to return periodically to the West Coast). Although never mentioned explicitly, his continual use of computers must have suggested human-computer interaction as a possible area. In early 1971, he discussed the possibility of doing research in this area with George Pake and Robert Taylor of the Xerox Palo Alto Research Center (PARC). Xerox PARC was formed in 1970 as a major research center for exploring digital electronic technologies in the context of office information systems. Newell thought it was the perfect place for the type of research project he had in mind. In 1973, Newell became a consultant to Xerox PARC, and in 1974, two of his students from CMU, Stu Card and Tom Moran, joined Xerox PARC to form the Applied Information-Processing Psychology Project to apply psychological theory to human-computer interaction. At the time, the design of computer interfaces was an art, without any scientific or engineering principles. Card, Moran, and Newell attempted to pull together existing psychological data and theory into a form that could be applied by interface designers, with the product being *The Psychology of Human-Computer Interaction* (Card, Moran, and Newell 1983).

> The domain of concern to us, and the subject of this book, is how humans interact with computers. A scientific psychology should help us in arranging this interface so it is easy, efficient, error-free—even enjoyable. (Card, Moran, and Newell 1983, p. 1)

For example, Fitts' Law predicts the time it takes for people to move their hands to touch an object. The key parameters are the distance to the object and the object's cross-section. Fitts' Law, combined with models of human typing, can be used to determine when it is most cost effective to use a mouse, a light pen, or keyboard commands for certain computer tasks.

Card, Moran, and Newell went one step beyond collecting and categorizing existing theory and data. They developed an engineering-level model of routine cognitive skill called the MODEL HUMAN PROCESSOR. Figure 6 shows a block diagram of the MODEL HUMAN PROCESSOR, with the perceptual, cognitive, and motor processors as well as the memories and stores that the processors access and store information in. Within each processor is its basic cycle time, and within the memories are their sizes, durations, and medium of representations.

Although the MODEL HUMAN PROCESSOR brings together many existing theories and data, it does not provide a methodology for analyzing the details of new tasks. This methodology was provided by GOMS, a language for analyzing and describing such behaviors. GOMS stood for goals, operators, methods, and selection, the basic components of the language used to describe a task. GOMS allowed a designer to make estimates of

the time it would take a human to carry out a routine task. The resulting model was not expected to match human behavior exactly but to capture the major components so that the time estimate would be within 20 percent of the actual time. The MODEL HUMAN PROCESSOR served as a preliminary venture for Newell's later attempt at casting SOAR as a unified theory of cognition.

The Symbol and Knowledge Levels

The eighties began with Newell helping to launch AAAI; he was its first president. His presidential address in 1982 provided an occasion to "propose a theory of the nature of knowledge, namely, that there is another computer system level immediately above the symbol (or program) level" (Newell 1982, p. 87). By proposing the *knowledge level,* he was claiming that an abstract level of analysis of a computational system (human or artificial) existed in which predictions of behavior can be made by knowing just the system's knowledge and goals (where prediction includes explaining behavior, controlling behavior, or constructing something that behaves to specification).

> Knowledge systems are just another level within this same hierarchy [of computer systems], another way to describe a system.... The knowledge level abstracts completely from the internal processing and the internal representation. Thus, all that is left is the content of the representations and the goals toward which that content will be used. As a level, it has a medium, namely, knowledge. It has a law of behavior, namely, if the system wants to attain goal *G* and knows that to do act *A* will lead to attaining *G*, then it will do *A*. This law is a simple form of rationality—that an agent will operate in its own best interests according to what it knows. (Newell 1990, pp. 48–49)

The force of Newell's proposal comes when systems are described at the knowledge level in terms of only knowledge and goals, and their behavior can successfully be predicted. Not all systems can be described this way because they fail to fully exploit the knowledge they have available to them (including most current computer systems), but the hypothesis is that intelligent systems can successfully and usefully be described at the knowledge level.

Just the year before his presidential address to AAAI, at the founding of the Cognitive Science Society, Newell declared that the

> most fundamental contribution so far of artificial intelligence and computer science to the joint enterprise of cognitive science has been the notion of a physical symbol system, i.e., the concept of a broad class of systems capable of having and manipulating symbols, yet realizable in the physical universe. (Newell 1980b, p. 136)

This idea was a continuation of the Turing Award lecture given by Newell and Simon (1976) in which they discussed the importance of symbols and search in AI and computer science in general. The importance of symbol systems comes from the hypothesis that they are sufficient for realizing the knowledge level.

> The entire field of artificial intelligence is, in a sense, devoted to discovering the symbol-level mechanisms that permit a close approximation to the knowledge level. (Newell 1990, p. 80)

Practice

In 1979, Newell agreed to contribute a piece on human practice to the Sixteenth Annual Carnegie Symposium on Cognition that was being organized by John Anderson for the spring of 1980. Because Paul Rosenbloom had just returned from a year as a visiting graduate student in the Psychology Department at the University of California at San Diego (UCSD) and was at somewhat loose ends from both the year of study and the termination of the IPS Project, he was recruited to help out.

What struck Newell most about human practice was the apparent uniformity of the shape exhibited by the experimental practice curves over a wide range of human performance. Sighting a potential regularity that could prove informative about the nature of the mind—one powerful source of regularity is the architecture—Newell and Rosenbloom (1981) first set out to establish whether this regularity was real or illusory and then, given its reality—the curves turned out uniformly to be power laws—to look for explanations of both the shape and its ubiquity. They eventually settled on an architectural explanation based on extending the already ubiquitous notion of chunking (Miller 1956) to procedural learning (Newell and Rosenbloom 1981). The initial abstract model of chunking was made operational in a task-specific fashion in the XAPS2 production-system language (Rosenbloom and Newell 1987)[2] and then in a task-independent, goal-based fashion in the XAPS3

the knowledge of the task would lead to behavior that constituted the appropriate method

language (Rosenbloom and Newell 1986). The activation side of the XAPS languages turned out to be a dead end because it was found necessary to eliminate it in going to the domain-independent XAPS3 model. In place of the activation side were added notions of goals and impasses (of a restricted form). The theory of chunking was taken considerably beyond the XAPS realization in its incorporation within SOAR. One interesting side-effect of this work was a theory of stimulus-response compatibility that has since been extended by Bonnie John (John, Rosenbloom, and Newell 1985).

SOAR

With the breakup of the IPS Project in the late seventies, Newell made another attempt to construct large production systems, but this time, he focused on using problem spaces as the organizing principle. This work began in the late seventies with a theoretical exploration (Newell 1980c). The work on problem spaces then took a practical turn in the fall of 1981 when Newell decided to create a system for building problem spaces as a tool for the introductory AI course at CMU. This project was to be an excuse to learn Lisp, but as the summer passed and it became clear he would not have time to do it alone, he recruited John Laird to help build the system. This system was never put to serious use, and in retrospect, Lisp was not an appropriate language for representing the operator and search-control knowledge required for problem spaces.

In parallel with this activity, Newell and Laird were attempting to find a way to characterize the weak methods in some unifying framework. This effort was a continuation of both MERLIN, which attempted to use frame-like beta structures for representing methods, and the IPS Project, which lacked a unifying framework for representing methods. The goal was to create a universal weak method such that there was no need to select explicitly the appropriate method for a task, but instead, the knowledge of the task would lead to behavior that constituted the appropriate method.

Newell and Laird did some initial work with data flow diagrams and then turned to the tool developed for the course. They had resisted using production systems as the underlying representation of knowledge, worrying that production systems would only complicate the problem of determining the relationship between methods and problem spaces. However, when attempts at using data flow diagrams and Lisp failed, the obvious place to turn was production systems.

Instead of choosing one of the OPS languages, they used XAPS2, with minor modifications to build the system to support the universal weak method. It was originally called SOR (state, operator, and result) after the main problem-solving step in a problem space. It then quickly became SOAR. Although SOAR was based on XAPS2, it was unable to make direct use of the XAPS2 task-specific chunking mechanism. The synthesis of chunking and problem spaces came later, after SOAR was reimplemented as an extension of OPS5, and automatic impasse-driven subgoal creation was implemented within SOAR in 1983 (Laird, Rosenbloom, and Newell 1986b; Laird, Newell, and Rosenbloom 1987).

In the summer of 1983, Newell, Laird, and Rosenbloom decided to place their intellectual bets with SOAR as an architecture for general intelligence and make it the center of their research for the immediate future. From 1983 until his death, Newell, in partnership with Laird and Rosenbloom, centered his research on SOAR. The first step of this research was to establish SOAR as an architecture for complex, real-world tasks. Following some friendly taunting from John McDermott (who had since been converted to expert systems), Rosenbloom, Laird, and Newell accepted the challenge of reimplementing part of R1 in SOAR and created R1-SOAR (Rosenbloom et al. 1985a). R1-SOAR demonstrated the viability of SOAR for building expert systems; it also demonstrated that it was possible to build an expert system with deep task knowledge, which was general but expensive, that was automatically compiled, through chunking, into efficient, shallow knowledge.

The mid-eighties saw a flurry of activity on integrated problem solving and learning within SOAR (Laird, Rosenbloom, and Newell 1984, 1986a, 1986b; Rosenbloom, Laird, and Newell 1987; Rosenbloom et al. 1985b; Steier et al. 1987). However, even as this activity was going on, Newell was studying algorithm discovery with David Steier and Elaine Kant in a system called DESIGNER (Kant and Newell 1983, 1984), which then migrated to DESIGNER-SOAR (Steier and Newell 1988).

Although SOAR was developed originally with purely functional constraints in mind, it became clear that it had many of the properties required to model human cognition. This fact is not completely surprising, given its roots in GPS, problem spaces, production systems, and chunking. Also, its basic design—which includes firing productions until exhaustion, automatically creating goals whenever an impasse arises during problem

solving, and integrating chunking with the impasse mechanism—began to raise the possibility that it would be an effective architecture for cognitive modeling. These observations, combined with the pressure of an invitation from Harvard University to present the William James Lectures, spurred Newell to formulate SOAR as the basis for a unified theory of cognition.

SOAR was in many ways a weak vessel for this attempt. At the time, it had little ability to interact with the environment, and most of the cognitive modeling efforts were still developing. However, Newell's purpose was not to claim that SOAR was the only unified theory but instead to use it as a vehicle to demonstrate what a unified theory would look like, encourage the field to embrace unified theories as an appropriate goal for psychological research, and encourage others to join in the search for unified theories.

> In my view, candidate unified theories should all play the game of "Anything you can do, I can do." We should strive to get our own theories to do what other theories can do well. (Newell 1992, p. 55)

> A high-jump metaphor is useful. As the world record high-jump moves higher, the field establishes an increasing threshold that each new high jumper must get over, just to qualify as a competitor. Similarly once we get the process established, each new unified theory must get over a threshold of coverage and adequacy before it can even be considered a candidate. The issue is not who's best—not who momentarily holds the championship—but who can qualify for the event. ACT* established an initial threshold. I think that SOAR, even though it has been largely exemplary so far, also qualifies as a candidate. (Newell 1992, p. 56)

The deadline for the lectures also provided an effective motivating force for his graduate students, leading to new results that modeled cognitive development, immediate behavior (Polk, Newell, and Lewis 1989), complex problem solving, and even cryptarithmetic. Newell, once again garnering the forces of a graduate student (Olin Shivers), went back to the original protocols of cryptarithmetic and attempted to have SOAR model the subjects more accurately than was possible with GPS. This work was successful and is reported in detail in Newell's (1990) book *Unified Theories of Cognition*.

Following the lectures, research on SOAR as a theory of cognition expanded around the world (Lewis et al. 1990), with Newell taking an active role in research on cognitive development (Simon, Newell, and Klahr 1991), natural language (Lehman, Lewis, and Newell 1991; Lehman et al. 1992), instruction taking (Lewis, Newell, and Polk 1989), visual attention, human-computer interaction, and syllogistic reasoning (Polk and Newell 1988).

During his last 10 years, Newell focused on SOAR as a general architecture for intelligent behavior, but he also found time to work on projects that either supported the use and development of SOAR or that were significant extensions of the concepts in SOAR.

Production-System Efficiency

A concern that often arises with production systems, be it PSG, OPS, or SOAR, is the efficient matching of large numbers of productions. The RETE algorithm was a first step but what about other algorithms? Would a specially designed production-system machine provide orders-of-magnitude improvement over software implementations? What role could parallelism play in production systems? What influence did learning have on the underlying architectures? These issues were investigated by Newell and his students and colleagues during his last 10 years (Gupta 1986; Gupta et al. 1988; Tambe et al. 1988; Tambe, Newell, and Rosenbloom 1990). Most recently, Bob Doorenbos created a system, DISPATCHER-SOAR, that learns over 10,000 productions (Doorenbos, Tambe, and Newell 1992).[3] This system has served as an important research tool and has demonstrated that for some applications, the speed of match does not deteriorate as even tens of thousands of productions are learned.

Rapid Task Acquisition

As identified early in the IPS Project, the acquisition of new tasks is difficult. To enhance the ability of users to create new tasks in SOAR, Newell worked with Gregg Yost to develop a system in which users program at the level of problem-space constructs and not at the level of productions (Yost and Newell 1989). The resulting system, TAQL, allowed Yost (1992) to create specific expert systems in SOAR as fast or faster than other developers using domain-specific knowledge-acquisition tools.

ASPM

As part of the work on cognitive modeling that Newell and his students were doing in SOAR, they had to determine which parameter settings of their model would fit the given data as closely as possible. Efficient solutions

1. To each scientific life, its own style.
   And each style defines a life.
2. Science is in the details.
3. The scientific problem chooses you, you don't choose it.
4. Diversions occur, make them count.
   Salvage what is possible for the main goal.
5. Embrace failure as part of success.
   But use it for the main goal.
6. Solve whatever problems must be solved.
   But do not be seduced by them.
7. Preserve the insight and deepen it.
   Deep ideas transform themselves beyond imagining.
8. Deep scientific ideas are exceedingly simple.
   Others usually see them as trivial.
9. Choose a final project to outlast you.
10. Everything must wait until its time.
    Science is the art of the possible.
11. To each scientific style its own maxims.
12. To work with results of field X, must be a professional in X.
13. We are all flawed—
    We all carry our burden of incapacities.
14. There is no substitute for working hard—very hard.
15. New things get started by evolution or chance, not design.
16. A scientist is a transducer from nature to theory.
    Seek out nature and listen to her.
17. The science is in the technique. All else is commentary.

Figure 7. Allen Newell's Maxims for a Dedicated Scientific Life.

have been found for certain types of model-fitting problems (for example, linear and integer programming) that involve specific types of parameters (usually continuous) and specific types of models (usually linear). However, these techniques often do not apply to computational cognitive models whose parameters are often discrete and symbolic and whose internal workings must be treated as a black box for the purposes of fitting—exactly the case that comes up with many of the SOAR models. Newell, together with Thad Polk, developed ASPM (analysis of symbolic parameter models) (Polk, Newell, and VanLehn 1992), a suite of computational tools for fitting and analyzing such symbolic parameter models. It exploits a few constraints that are common among computational models (for example, that only a few parameters are relevant to each task) to avoid doing an exhaustive search of all parameter settings. ASPM was successfully used to fit and analyze computational models with well over 10-billion parameter settings and is currently available to interested researchers.

Social Systems

Five years ago, Newell returned to the area that led him to AI: theories of social agents and organizations. Working with Kathleen Carley, he developed a theory of the nature of social agents, called the MODEL SOCIAL AGENT, a parallel to the MODEL HUMAN PROCESSOR (Carley and Newell 1992).

Diversions

Newell dedicated himself to his research in AI and psychology in pursuit of the nature of the mind. However, he had his share of diversions from this quest, which arose in large part for personal reasons.

Computer Structures

In the late 1960s, Newell was diverted from his research on mind for the first time (1968–1972). This diversion arose when he agreed to help Gordon Bell write a book on computer systems. This textbook was to be the first to give detailed descriptions of computer systems. In the end, it was much more because in the process of trying to describe the different architectures, Bell and Newell created languages for two different levels of computer design: the system level (PMS) and the instruction level (ISP). At the system level, the PMS language allowed a description of computers and networks in terms of their component memories, processors, switches, controls, transducers, data operators, and

links. At the instruction level, the ISP language provided the means for describing the detailed operation of the instruction set. Later, when an interpreter was built for the ISP language, the dream of universal emulation was realized, and it was possible to simulate any computer on another computer as long as there was an ISP description of the first computer and an ISP interpreter for the second.

Although this diversion from the study of intelligence was significant, it crystallized the ideas of architecture and a hierarchy of levels for analyzing computational systems. The parallels between the levels of computer architecture and the human mind were evident to Newell and Simon as early as 1962 (Newell 1962) and were a continual theme in Newell's work, including his views on the knowledge level and unified theories of cognition.

L*

A second diversion that developed in the late sixties and carried over into the seventies was a language called L* (Newell et al. 1971). This language was developed with Peter Freeman, George Robertson, and Don McCracken. It was an attempt to create a language for system development so that it would be easy for a system programmer to create a customized operating system and user interface. The underlying belief was that each user would want to have an individually customized system and that the way to create such a system was to provide an underlying kernel that could then quickly be grown into a complete system. Thus, L* provided the ability to rapidly prototype a new system.

> Our demonstration of it to the local community was to start with an empty machine and during the course of the seminar to build all of the structure of L* from scratch, before your very eyes. (A. Newell, September 1991, conversation with John Laird, Univ. of Michigan AI Lab)

It is hard to come up with a simple characterization of L*. It's not $X + Y$, where X is some other existing language, such as Lisp, and Y is some additional capability such as coroutines. Possibly, its most novel feature was that every symbol had its own interpreter, close to some of the concepts behind object-oriented programming. It also had a universal-type mechanism, so that every symbol had a type. L* was also a chance to build a system programming language with many of the list-processing ideas that came up in IPLS.

L* was never a widely used language, although it was the implementation language for PSG, OPS1, and ZOG, another diversion for Newell that came a bit later. In contrast to Newell's previous diversion with Gordon Bell, which had a great impact on computer science, L* suffered the same fate as MERLIN and slipped into obscurity.

Speech Understanding

Another diversion for Newell in the early seventies was the Advanced Research Project Administration (ARPA) speech effort. In 1970, ARPA was considering a major program in speech understanding. Newell was asked to take the lead in writing the report for the study group, specifically because he was not an active speech researcher. The report laid out the first organizational plan for a multi-institutional research effort in AI. This report led directly to the ARPA Speech-Understanding Research Effort; Newell became the chair of the Steering Committee and wrote the final report (Newell et al. 1973).

Although Newell did not lead any of the speech-understanding teams, he was interested in the underlying architecture of HARPY (Lowerre 1976), the most successful system (which is the predecessor of today's hidden-Markov model speech-understanding systems). Newell decided to take the HARPY model seriously as a model of human speech perception and attempt to view it in terms of a production system. This effort led to a new production system, HPSA77 (Newell 1980a), in which Newell evaluated the sufficiency of HARPY. HPSA77 had many novel features, including activation of working memory, but it was never implemented, although it would later influence Paul Rosenbloom in the development of XAPS.

The parallels between the levels of computer architecture and the human mind were evident to Newell and Simon as early as 1962

List Processing	1956
Search	1957
Protocol Analysis	1957
Symbols	1960
Problem Spaces	1965
Production Systems	1967
Weak Methods	1969
Model Human Processor	1973
Knowledge Level	1980
Chunking	1981
Impasses (universal subgoaling)	1983

Figure 8. Scientific Ideas of Allen Newell.

System name	*Co-developers*	*Task*	*Years*	*Innovative Contributions*
LT	Shaw, Simon	Propositional Logic	l955-56	Heuristic search
IPL	Shaw, Simon (Feigenbaum in IPL V)	List Processing	1956-64	Lists Schemas (frames) Dynamic memory allocation Recursion Functions as arguments Generators (streams)
NSS	Shaw, Simon	Chess	1957-58	Human-like behavior
GPS	Shaw, Simon (Earnst)	Puzzles	1957-63	Means-ends-analysis Recursive goal structures Computer model of complex human behavior
MERLIN	Moore, Young	Intelligent Tutoring	1968-74	Attached procedures General mapping Indefinite context dependencies Automatic compilation
L*	Robertson, McCracken	System software	1969-74	Universal interpretation
PSG		Cognitive modeling	1970-73	Production system model of human behavior
PASS	Waterman	Protocol Analysis	1970-73	
ZOG	Robertson, McCracken	Interface	1973-84	
IPS	Rychener, et al.	Instruction	1976-79	
OPS	Forgy, et al.	Production system	1976-80	Efficient production match
HPSA77		Speech understanding	1977	
XAPS	Rosenbloom	Learning	1979-83	Chunking
DESIGNER	Kant, Steier	Algorithm design	1982-85	
SOAR	Laird, Rosenbloom		1982-	Integration of production systems and problem spaces Universal weak method Impasse-driven goal creation Universal subgoaling Integrated learning and problem solving
CParaOPS5	Gupta, et al.	Parallel OPSS	1985-89	Parallel rete production match
TAQL	Yost	Knowledge acquisition	1987-92	Problem-space level programming
RL-SOAR	Rosenbloom, Laird	Computer configuration	1983-86	Integrated deep and shallow reasoning
DESIGNER-SOAR	Steier	Algorithm design	1985-88	Learning in design
SYL-SOAR	Polk	Syllogisms	1987-88	Mental models as states in problem space
SC-SOAR	T. Simon, Klahr	Series completion	1988-89	
IR-SOAR	Polk	Immediate reasoning	1988-90	Model multiple immediate reasoning tasks
NL-SOAR	Lewis, Lehman	Natural language	1988-92	Recognition-based comprehension
Q-SOAR	T. Simon, Klahr	Number conservation	1989-91	Model transition during development
VR-SOAR	Polk	Verbal reasoning	1990-91	
PLURAL-SOAR	Carley, et al.	Warehouse management	1990-92	
BROWSER-SOAR	John	Database browsing	1990-92	
MATHEMATICA-SOAR	Pathak, Steier	Software system control	1990-92	
DISPATCHER-SOAR	Doorenbos	Message management	1991-92	Large learning system
RAIL-SOAR	Altmann	Railroad yard switching	1991-92	Multiple learning strategies

Figure 9. Systems of Allen Newell.

ZOG

In the summer of 1972, Newell helped organize a summer workshop on simulation for cognitive psychologists. To provide hands-on demonstrations of several large simulation programs for novice computer users, Newell, George Robertson, and Don McCracken developed a program that provided a uniform interface for using the simulation programs. This system was called ZOG[4] and had as its chief deficiency the use of 300-baud output to hard copy. This deficiency proved to be such a bottleneck that the system was temporarily abandoned. In 1975, they became aware of a remarkably similar system called PROMIS (problem-oriented medical information system), an effort led by Dr. Lawrence Weed of the University of Vermont Medical School (Schultz and Davis 1979; Walton, Holland, and Wolf 1979). PROMIS avoided the failure of the initial ZOG by using state-of-the-art touch-screen technology to provide one-half–second response.

PROMIS remained dedicated to its medical application, but Newell, Robertson, and McCracken decided to create a new general version of ZOG that used the new technology. ZOG became a fast-access hypermedia system and interface for preexisting programs using a touch-sensitive screen (Robertson, McCracken, and Newell 1980). The goal of the research was to study how people can use such an interface, where new information is displayed very rapidly (within 1/10 second) upon selection of an item. Thus, it was a forerunner of hypermedia systems such as HyperCard; however, it was also a tool for studying how people use these systems.

When ZOG was developed, it pushed the limits of the existing workstation, disk, and touch-screen technology. In addition to being a research tool, an experimental ZOG system was fielded on the USS *Carl Vinson*, a nuclear-powered aircraft carrier (Newell et al. 1982). ZOG acted as an interactive management information and planning system. The system was a mixed success, being fielded on one of the first workstations (PERQs) in an unstable electric environment (power surges are the norm on an aircraft carrier) and within a military system that had to work. Subsequent to his work at RAND, this instance was one of the few examples of applied research that Newell was involved in.

Reflections

Many threads run through Allen Newell's research career, but the strongest was his pursuit of the architecture of the mind.

> Cognitive science has one result of extreme generality and general acceptance. In the present context it can be expressed as: *unified theories take the form of architectures*. (Newell 1992, p. 27)

His first work in AI was in chess and LT. What came out of LT were not new insights into logic but, instead, insights into the fundamentals of thought: symbol systems and heuristic search. In GPS, he and his colleagues created the first AI architecture, a system that "separated out the program structure for problem solving from the program structure to describe a particular task" (Newell 1992, p. 35). From there, he tried to create architectures that avoid the rigidities and brittleness that at times seem inherent to AI systems. This pursuit was nonstop; Newell never showed an inkling of slowing down. SOAR was his latest attempt, which was "another try at putting together a complete system that integrated all these elements and learning too, as it turned out" (Newell 1990, p. ix). SOAR has had remarkable staying power for an AI architecture, but after 10 years, it still feels too early to predict its eventual fate.

Another striking thing about Allen Newell that was difficult to convey in this article was the purity in his approach to science. In his Distinguished Lecture titled "Desires and Diversions" presented on 4 December 1991 at CMU, he summarized his methodology (and his philosophy of science) with a series of maxims. His list is recreated in Figure 7. Some of these maxims relate to a specific style of research (one dedicated to pursuing a single question: 4, 5, 6), but most apply to a dedicated scientific life in general, for which Newell set the perfect example.

We expand here on a few of our favorite maxims, those that are quintessential Newell.

2. Science is in the details. The breadth of Newell's research might suggest that Newell "skimmed the cream," contributing mostly frameworks or theories and leaving the details for others. As evidenced by the depth of his contribution in each area, nothing could be further from the truth. For Newell, generalities, such as his work on physical symbol systems or the knowledge level, were insufficient, even when one addressed a subject as broad as the nature of the human mind. Theories had to have an associated computational model that addressed the details of human behavior. These details included how long it takes people to determine whether a number is in a list they've just seen (the Sternberg task) (Newell 1973a, 1990), how long they take to learn to press buttons to match a light pattern, and how

What came out of it were not new insights into logic but, instead, insights into the fundamentals of thought: symbol systems and heuristic search

they solve puzzles like cryptarithmetic (Newell and Simon 1972; Newell 1990). These details were not the details of the research but its core because from details, Newell abstracted and generated his theories.

7. Preserve the insight and deepen it. Deep ideas transform themselves beyond imagining. Newell believed that to understand an idea, you had to study it long and study it hard. Consider the idea of chunking that arose originally for declarative structures in work by George Miller (1956). Newell and Rosenbloom took chunking and extended it to procedural learning in a specific task to study the effects of practice. From there, chunking was generalized to be a domain-independent form of goal caching. Goal caching might seem to be a relatively straightforward type of rote learning, although as implemented in SOAR, it was essentially explanation-based learning (Rosenbloom and Laird 1986). In SOAR, with subgoals generated automatically for all impasses in knowledge, chunking became a general learning mechanism that could be used to acquire selection and generation knowledge for problem spaces, states, and operators—all the types of knowledge encoded in SOAR. However, it appeared that chunking would be restricted to symbol-level learning and be unable to acquire new facts about the world. However, under further examination and with more experimentation, it was discovered that SOAR's chunking mechanism could learn declarative structures, so-called *data chunking*, if the problem solving on which the learning was based was itself inductive (Rosenbloom, Laird, and Newell 1987; Rosenbloom and Aasman 1990). Chunking has since been used to model human concept formation (Miller and Laird 1991) and development (Simon, Newell, and Klahr 1991).

8. Deep scientific ideas are exceedingly simple. Others usually see them as trivial. One of Newell's guides in his pursuit of the mind was to look for simple, uniform ideas. Figure 8 lists what he considered his major ideas, all of which are simple but also powerful when examined in detail. For example, in SOAR, all long-term knowledge is represented as productions, all short-term knowledge is represented as attribute values, all problem solving is formulated as processing within problem spaces, all goals are generated from architectural impasses, and all learning is gathered by chunking. Uniformity alone is not sufficient; it must also be general, with the ability to be specialized to the demands of a specific domain. For example, chunking is SOAR's only learning mechanism, and it is applied uniformly to every problem. Furthermore, what it learns is specific to the problem and to the knowledge used to solve the problem. In MERLIN, mapping was everywhere, but in L*, every symbol had its own interpreter, and all structures arose from the combinations of these interpreters (unfortunately, these latter systems were ultimately not successful for reasons unrelated to their underlying scientific ideas).

14. There is no substitute for working hard—very hard. Newell's work was his life. If he ever tired of what he was working on, he simply switched to a new topic and focused his remarkable energies on it. As evidenced by his contributions throughout the years, he worked hard, very hard during his complete research career.

Many a student tells the story of being up late at night (into the morning) and, in the midst of puzzlement, dashing off a short note to Newell, expecting a response sometime the next day. More often than not, the response would be immediate. If not immediate, it would be waiting the next morning, with an extensive exploration of the implications of the student's question, far beyond what the student thought possible.

16. A scientist is a transducer from nature to theory. Seek out nature and listen to her. Newell, as is true of almost every researcher, had significant ego invested in his theories. However, he was not afraid of the truth, and if the truth dictated it, he was ready to abandon a theory. He did not continually jump from one theory to another as he found flaws because he also believed that a theory should be nurtured. As evidence to the contrary accumulates, a theory should evolve to embrace this evidence if possible rather than immediately discard it.

One of his favorite sayings to graduate students as they labored over their computers, running huge numbers of simulations or analyzing reams of human protocol data, was, "I love data!"[5]

> Back in the sixties, whenever I used to go home kind of depressed because I hadn't learned anything in a week, I would take a new protocol. I would take an evening, and I'd clear the evening away and I'd spend the evening with that protocol and I would always find out something about human behavior that I didn't know. Just listen to nature. (Newell 1991)

When presented with data that didn't fit his theories, he would say, "Now that's interesting." New data might have given him insight into how to change his theories because his theories were not floating crys-

talline structures that a single imperfection would bring crashing down. His theories were based on established fundamentals that covered a wide variety of phenomena but always had edges and corners where they didn't fit as well as incomplete structures where there was just not enough time to fill in all the details. Thus, he was always looking for new data that might help twist or stretch or, best of all, simplify a theory for more coverage. His search was active because he wanted to be the first to find data that confirmed or inspired extensions of a theory or that pointed out the theory's faults. For example, in the mid-sixties, after many years of modeling verbal protocols of cryptarithmetic in GPS, the story was still incomplete.

> In sum, linguistic data is still far from sufficient either to discover the processing system being used or to verify conclusively that it accurately models the human information processing system.
>
> We need to obtain additional data. (Newell 1967, p. 34)

The additional data in this case were eye movements of subjects in addition to their verbal protocols, and this research led to one of the first models of human behavior at this level of detail for a complex task (Newell 1967).

17. The science is in the technique. All else is commentary. In his talk, he mentioned two subpoints. The first subpoint is, "What lives is what your scientific descendants must use to solve their problems" (Newell 1991). For Newell, the real contributions were the theory, the data, and the tools—sometimes programs and sometimes methodology—that could be used by others. He spent enormous energy developing tools and architectures (IPL, GPS, L*, PSG, OPS5, MERLIN, SOAR) and techniques (problem-behavior graphs, protocol analysis, GOMS). To him, a theory was only useful if it could be captured in a technique or a system that others could use.

The second subpoint is, "Do your philosophy on the banquet circuit" (Newell 1991). Newell had little use for philosophy and metatheory as tools for advancing science. Science is in the details of data and computational theories. Ironically, some of his most influential work—symbol systems, 20 questions, and the knowledge level—is metatheory. However, to him, these accomplishments were only the by-products of science.

> One element in AI's methodology is that progress is made by building systems that perform: synthesis before analysis.
>
> What is science in AI? It is knowledge—theories, data, evaluations—that describes the means to reach a class of desired ends given certain structures and situations. Science reaches beyond the situation of its generation and becomes a source of knowledge for future scientists and technologists—investment rather than consumption. Knowledge of means-ends relations is what characterizes the artificial sciences, of which AI is a part. (Newell 1977, p. 970)

Newell lived all these maxims, but the maxim that captures Allen Newell's scientific life more than any other came from his father (who said it incessantly): "Keep your eye on the main chance."

Conclusion

Over the eons, many have taken the nature of the human mind as their pursuit. Newell was fortunate enough to start his pursuit immediately following the birth of computation, and he was brilliant enough to understand its potential. Newell blazed a path, developing a new understanding of the mind. The results are staggering. Newell led the field by developing programs, languages, and theories, as listed in figure 9. Many of his individual contributions would be considered the culmination of a successful scientific career, and together, they define many of the fundamentals of cognitive science and AI. It is hard to imagine AI and cognitive science without him. He is possibly the greatest scientist that cognitive science and AI will ever see.

Those of us who worked closely with him will miss him dearly: his brilliant and deep analysis of a new result; his penetrating questions; his wit; his enthusiasm; but, most of all, his vision into all the areas of the mind we have yet to even consider. Although we are greatly saddened, we see his loss to the field as even greater. He led the field in his vision, his results, and his style. He gave the field an example of what a great scientist is and what great science requires. Without his example, we must redouble our efforts so that the field will not lose sight of the ultimate question and settle for something less.

Acknowledgments

Special thanks go to Herbert Simon for his substantial contributions to this article. Additional thanks go to Ed Feigenbaum, Mitch Waldrop, Doug Pearson, Craig Miller, and John R. Laird for commenting on earlier drafts of this article. Thanks also go to David Steier and Thad Polk for help in collecting material.

Notes

1. This article is based on material drawn from many sources. These sources include Allen Newell's brief autobiography published in *American Psychologist* (American Psychological Association 1986), his Distinguished Lecture entitled "Desires and Diversions" presented on 4 December 1991 at Carnegie Mellon University, the Historical Addendum to *Human Problem Solving* (Newell and Simon 1972), Pamela McCorduck's (1979) *Machines Who Think*, Herbert Simon's (1991) autobiography *Models of My Life*, and Allen Newell's many publications over the years. The interested reader should consider McCorduck's book and Simon's autobiography for a deeper and more personal look at Allen Newell (and other early AI researchers), especially during the late fifties and early sixties.

2. The original XAPS (experimental activation-based production system) was developed by Rosenbloom while he was at the University of California at San Diego (UCSD) in an attempt to merge insights from OPS4, HPSA77, and the work on parallel distributed processing that was just beginning at UCSD.

3. At last count, it was at 113,000.

4. Although the source of SOAR as an acronym is obscure, it does exist. In contrast, ZOG is not an acronym, just a three-letter nonsense syllable.

5. Given his love of data, there was significant irony in his claim that psychology has sufficient data to build unified theories of cognition.

References

American Psychological Association. 1986. Awards for Distinguished Scientific Contributions: 1985. *American Psychologist* 41:337–353.

Brownston, L.; Farrell, R.; Kant, E.; and Martin, N. 1984. Programming Expert Systems in OPS5: *An Introduction to Rule-Based Programming*. Reading, Mass.: Addison-Wesley.

Card, S.; Moran, T.; and Newell, A. 1983. *The Psychology of Human-Computer Interaction*. Hillsdale, N.J.: Lawrence Erlbaum.

Carley, K., and Newell, A. 1992. On the Nature of the Social Agent.Computer Science Dept., Carnegie Mellon Univ.

de Groot, A. 1946. *Het Denken van Den Schaker*. Amsterdam: N. H. Utig. Mij.

Doorenbos, R.; Tambe, M.; and Newell, A. 1992. Learning 10,000 Chunks: What's It Like Out There. In Proceedings of the Tenth National Conference on Artificial Intelligence, 830–836. Menlo Park, Calif.: American Association for Artificial Intelligence.

Erman, L.; Hayes-Roth, F.; Lesser, V.; and Reddy, D. 1980. The HEARSAY-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys* 12(2): 213–253.

Ernst, G., and Newell, A. 1969. *GPS: A Case Study in Generality and Problem Solving*. San Diego, Calif.: Academic.

Feigenbaum, E., and Feldman, J. 1963. *Computers and Thought*. New York: McGraw-Hill.

Fikes, R., and Nilsson, N. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2(3–4): 189–208.

Floyd, R. 1961. An Algorithm for Coding Efficient Arithmetic Operations. *Communications of the ACM* 4:42–51.

Forgy, C. 1984. The OPS83 Report, Technical Report, Computer Science Dept., Carnegie Mellon Univ.

Forgy, C. 1981. OPS5 User's Manual, Technical Report, Computer Science Dept., Carnegie Mellon Univ.

Forgy, C., and McDermott, J. 1977. OPS, A Domain-Independent Production-System Language. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 933–939. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Gupta, A. 1986. Parallelism in Production Systems. Ph.D. diss., Computer Science Dept., Carnegie Mellon Univ.

Gupta, A.; Tambe, M.; Kalp, D.; Forgy, C.; and Newell, A. 1988. Parallel Implementation of OPS5 on the ENCORE Multiprocessor: Results and Analysis. *International Journal of Parallel Programming* 17(2): 95–124.

Hayes, J., and Simon, H. 1976. Understanding Complex Task Instructions. In *Cognition and Instruction*, ed. D. Klahr, 269–286. Hillsdale N.J.: Lawrence Erlbaum.

Hayes, J., and Simon, H. 1974. Understanding Written Problem Instructions. In *Knowledge and Cognition*, ed. L. Gregg, 167–200. Hillsdale, N.J.: Lawrence Erlbaum.

John, B.; Rosenbloom, P.; and Newell, A. 1985. A Theory of Stimulus-Response Compatibility Applied to Human-Computer Interaction. In Proceedings of CHI-85, the Annual Conference on Computer-Human Interaction, 213–219. New York: Association of Computing Machinery.

Kant, E., and Newell, A. 1984. Problem-Solving Techniques for the Design of Algorithms. *Information Processing and Management* 20(1–2): 97–118.

Kant, E., and Newell, A. 1983. An Automatic Algorithm Designer: An Initial Implementation. In Proceedings of the Third National Conference on Artificial Intelligence, 177–181. Menlo Park, Calif.: American Association for Artificial Intelligence.

Laird, J.; Newell, A.; and Rosenbloom, P. 1987. SOAR: An Architecture for General Intelligence. *Artificial Intelligence* 33(3): 1–64.

Laird, J.; Rosenbloom, P.; and Newell, A. 1986a. Chunking in SOAR: The Anatomy of a General Learning Mechanism. *Machine Learning* 1:11–46.

Laird, J.; Rosenbloom, P.; and Newell, A. 1986b. Overgeneralization during Knowledge Compilation in SOAR. In Proceedings of the Workshop on Knowledge Compilation, ed. T. Dietterich, 46–57. Corvallis, Ore.: Oregon State University.

Laird, J.; Rosenbloom, P.; and Newell, A. 1986c. *Universal Subgoaling and Chunking: The Automatic Generation and Learning of Goal Hierarchies*. Norwell, Mass.: Kluwer Academic.

Laird, J.; Rosenbloom, P.; and Newell, A. 1984. Toward Chunking as a General Learning Mechanism. In Proceedings of the Fourth National Con-

ference on Artificial Intelligence, 188–192. Menlo Park, Calif.: American Association for Artificial Intelligence.

Lehman, J.; Lewis, R.: and Newell, A. 1991. Natural Language Comprehension in SOAR: Spring 1991, Technical Report, CMU-CS-91-117, Computer Science Dept., Carnegie Mellon Univ.

Lehman, J.; Newell, A.; Polk, T.; and Lewis, R. 1992. The Role of Language in Cognition: A Computational Inquiry. In *Conceptions of the Human Mind*, ed. G. Harman. Hillsdale, N.J.: Lawrence Erlbaum. Forthcoming.

Lewis, S.; Huffman, S.; John, B.; Laird, J.; Lehman, J.; Newell, A.; Rosenbloom, P.; Simon, T.; and Tessler, S. 1990. SOAR as a Unified Theory of Cognition: Spring 1990. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, 1035–1042. Hillsdale, N.J.: Lawrence Erlbaum.

Lewis, R.; Newell, A.; and Polk, T. 1989. Toward a SOAR Theory of Taking Instructions for Immediate Reasoning Tasks. In Proceedings of the Ninth Annual Conference of the Cognitive Science Society, 514–521. Hillsdale, N.J.: Lawrence Erlbaum.

Lindsay, R.; Buchanan, B.; Feigenbaum, E.; and Lederberg, J. 1980. *Applications of Artificial Intelligence to Organic Chemistry: The DENDRAL Project*. New York: McGraw-Hill.

Lowerre, B. 1976. The HARPY Speech-Recognition System. Ph.D. diss., Computer Science Dept., Carnegie Mellon Univ.

McCorduck, P. 1979. *Machines Who Think*. San Francisco: Freeman.

McDermott, J. 1982. R1: A Rule-Based Configurer of Computer Systems. *Artificial Intelligence* 19(1): 39–88.

Miller, G. 1956. The Magic Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review* 63:81–97.

Miller, C., and Laird, J. 1991. A Constraint-Motivated Lexical Acquisition Model. In Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society, 827–831. Hillsdale, N.J.: Lawrence Erlbaum.

Moore, J., and Newell, A. 1974. How Can Merlin Understand? In *Knowledge and Cognition*, ed. L. Gregg, 201–252. Hillsdale, N.J.: Lawrence Erlbaum.

Newell, A. 1992. Unified Theories of Cognition and the Role of SOAR. In *SOAR: A Cognitive Architecture in Perspective*, eds. J. Michon and A. Akyurek, 25–75. Norwell, Mass.: Kluwer Academic.

Newell, A. 1991. Desires and Diversions. Distinguished Lecture, School of Computer Science, Carnegie Mellon University, 4 December.

Newell, A. 1990. *Unified Theories of Cognition*. Cambridge, Mass.: Harvard University Press.

Newell, A. 1983. The Heuristic of George Polya and Its Relation to Artificial Intelligence. In *Methods of Heuristics*, eds. R. Groner, M. Groner, and W. Bischof, 195–238. Hillsdale, N.J.: Lawrence Erlbaum.

Newell, A. 1982. The Knowledge Level. *Artificial Intelligence* 18(1): 87–127.

Newell, A. 1980a. HARPY, Production Systems, and Human Cognition. In *Perception and Production of Fluent Speech*, ed. R. Cole, 289–395. Hillsdale, N.J.: Lawrence Erlbaum.

Newell, A. 1980b. Physical Symbol Systems. *Cognitive Science* 4(2): 135–183.

Newell, A. 1980c. Reasoning, Problem Solving, and Decision Processes: The Problem Space as a Fundamental Category. In *Attention and Performance VIII*, ed. R. Nickerson, 693–718. Hillsdale, N.J.: Lawrence Erlbaum.

Newell, A. 1977. Reflections on Obtaining Science through Building Systems. In Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 970–971. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Newell, A. 1973a. Production Systems: Models of Control Structures. In *Visual Information Processing*, ed. W. Chase, 463–526. San Diego, Calif.: Academic.

Newell, A. 1973b. You Can't Play 20 Questions with Nature and Win. In *Visual Information Processing*, ed. W. Chase, 283–308. San Diego, Calif.: Academic.

Newell, A. 1972. A Theoretical Exploration of Mechanisms for Coding the Stimulus. In *Coding Processes in Human Memory*, eds. A. Melton and E. Martin, 373–434. Washington, D.C.: Winston and Sons.

Newell, A. 1969. Heuristic Programming: Ill-Structured Problems. In *Progress in Operations Research III*, ed. J. Aronofsky, 360–414. New York: Wiley.

Newell, A. 1967. Eye Movements and Problem Solving. In Computer Science Research Review, 29–40. Pittsburgh, Penn.: Carnegie-Mellon University Computer Science Department.

Newell, A. 1962. Some Problems of Basic Organization in Problem-Solving Programs. In *Self-Organizing Systems*, eds. M. Yovits, G. Jacobi, and G. Goldstein, 393–423. Washington, D.C.: Spartan.

Newell, A. 1961. *Information Processing Language V Manual*. Englewood Cliffs, N.J.: Prentice Hall.

Newell, A. 1957. Information Processing: A New Technique for the Behavioral Sciences. Ph.D. diss., Graduate School of Industrial Administration, Carnegie Institute of Technology.

Newell, A. 1955. The Chess Machine: An Example of Dealing with a Complex Task by Adaption. In Proceedings of the 1955 Western Joint Computer Conference, 101–108. New York: Institute of Radio Engineers.

Newell, A., and Baez, A. 1949. Caustic Curves by Geometric Construction. *American Journal of Physics* 29:145–147.

Newell, A., and Rosenbloom, P. 1981. Mechanisms of Skill Acquisition and the Law of Practice. In *Learning and Cognition*, ed. R. Anderson, 1–55. Hillsdale, N.J.: Lawrence Erlbaum.

Newell, A., and Shaw, J. 1957. Programming the Logic Theory Machine. In Proceedings of the 1957 Western Joint Computer Conference, 230–240. New York: Institute of Radio Engineers.

Newell, A., and Simon, H. 1976. Computer Science as Empirical Inquiry: Symbols and Search. *Communications of the ACM* 19(3): 113–126.

Newell, A., and Simon, H. 1972. *Human Problem*

Solving. Englewood Cliffs, N.J.: Prentice-Hall.

Newell, A., and Simon, H. 1961a. Computer Simulation of Human Thinking. *Science* 234:2011–2017.

Newell, A., and Simon, H. 1961b. GPS, A Program That Simulates Human Thought. In *Lernende Automaten*, ed. H. Billing, 109–124. Munich: Oldenbourg. (Reprinted in 1963. *Computers and Thought*, eds. E. Feigenbaum and J. Feldman, 39–70. New York: McGraw-Hill.)

Newell, A., and Simon, H. 1961c. The Simulation of Human Thought. In *Current Trends in Psychological Theory*, 152–179. Pittsburgh, Penn.: University of Pittsburgh.

Newell, A., and Simon, H. 1956a. Current Developments in Complex Information Processing, Technical Report, P-850, RAND Corporation, Santa Monica, California.

Newell, A., and Simon, H. 1956b. The Logic Theory Machine: A Complex Information-Processing System. *IRE Transactions on Information Theory* IT-2:61–79.

Newell, A.; Shaw, J.; and Simon, H. 1962. The Processes of Creative Thinking. In *Contemporary Approaches to Creative Thinking*, eds. H. Gruber, G. Terrell, and J. Wertheimer, 63–119. New York: Atherton.

Newell, A.; Shaw, J.; and Simon, H. 1960a. A Variety of Intelligent Learning in a General Problem Solver. In *Self-Organizing Systems*, eds. M. Yovits and S. Cameron, 153–189. Elmsford, N.Y.: Pergamon.

Newell, A.; Shaw, J.; and Simon, H. 1960b. Report on a General Problem-Solving Program for a Computer. In Information Processing: Proceedings of the International Conference on Information Processing, 256–264. Paris: United Nations Educational, Scientific, and Cultural Organization.

Newell, A.; Shaw, J.; and Simon, H. 1958. Chess-Playing Programs and the Problem of Complexity. *IBM Journal of Research and Development* 2: 320–325. (Reprinted in 1963. *Computers and Thought*, eds. E. Feigenbaum and J. Feldman. New York: McGraw-Hill.)

Newell, A.; Shaw, J.; and Simon, H. 1957. Empirical Explorations of the Logic Theory Machine: A Case Study in Heuristics. In Proceedings of the 1957 Western Joint Computer Conference, 218–230. New York: Institute of Radio Engineers (Reprinted in 1963. *Computers and Thought*, eds. E. Feigenbaum and J. Feldman, 109–133. New York: McGraw-Hill.)

Newell, A.; Freeman, P.; McCracken, D.; and Robertson, G. 1971. The Kernel Approach to Building Software Systems. In Computer Science Research Review 1970–71, 39–51. Pittsburgh, Penn.: Carnegie Mellon University Computer Science Department.

Newell, A.; McCracken, D.; Robertson, G.; and Akscyn, R. 1982. ZOG and the USS *Carl Vinson*. In Computer Science Research Review, 95–118. Pittsburgh, Penn.: Carnegie Mellon University Computer Science Department.

Newell, A.; Tonge, F.; Feigenbaum, E.; Green, B.; and Mealy, G. 1964. *Information Processing Language V*, 2d ed. Englewood Cliffs, N.J.: Prentice-Hall.

Newell, A.; Yost, G.; Laird, J.; Rosenbloom, P.; and Altmann, E. 1991. Formulating the Problem-Space Computational Model. In *Carnegie Mellon Computer Science: A 25-Year Commemorative*, ed. R. Rashid, 255–293. Reading, Mass.: Addison-Wesley.

Newell, A.; Barnett, J.; Forgie, J.; Green, C.; Klatt, D.; Licklider, J.; Munson, J.; Reddy, D.; and Woods, W. 1973. *Speech-Understanding Systems: Final Report of a Study Group*. Amsterdam: North-Holland.

Nii, H. 1986. Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *AI Magazine* 7(2): 38–53.

Polk, T., and Newell, A. 1988. Modeling Human Syllogistic Reasoning in SOAR. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, 181–187. Hillsdale, N.J.: Lawrence Erlbaum.

Polk, T.; Newell, A.; and Lewis, R. 1989. Toward a Unified Theory of Immediate Reasoning in SOAR. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, 506–513. Hillsdale, N.J.: Lawrence Erlbaum.

Polk, T.; Newell, A.; and VanLehn, K. 1992. Analysis of Symbolic Parameter Models (ASPM): A New Model-Fitting Technique for the Cognitive Sciences, Department of Computer Science, Carnegie Mellon Unversity.

Polya, G. 1945. *How to Solve It*. Princeton, N.J.: Princeton University Press.

Post, E. 1943. Formal Reductions of the General Combinatorial Decision Problem. *American Journal of Mathematics* 65:197–268.

Robertson, G.; McCracken, D.; and Newell, A. 1980. The ZOG Approach to Man-Machine Communication. *International Journal of Man-Machine Studies* 14(4): 461–488.

Rosenbloom, P., and Aasman, J. 1990. Knowledge Level and Inductive Uses of Chunking (EBL). In Proceedings of the Eighth National Conference on Artificial Intelligence, 821–827. Menlo Park, Calif.: American Association for Artificial Intelligence.

Rosenbloom, P., and Laird, J. 1986. Mapping Explanation-Based Generalization onto SOAR. In Proceedings of the Fifth National Conference on Artificial Intelligence, 561–567. Menlo Park, Calif.: American Association for Artificial Intelligence.

Rosenbloom, P., and Newell, A. 1987. Learning by Chunking: A Production-System Model of Practice. In *Production System Models of Learning and Development,* 221–286. Cambridge, Mass.: Bradford Books.

Rosenbloom, P., and Newell, A. 1986. The Chunking of Goal Hierarchies: A Generalized Model of Practice. In *Machine Learning: An Artificial Intelligence Approach,* volume 2, eds. R. Michalski, J. G. Carbonell, and T. M. Mitchell, 247–288. San Mateo, Calif.: Morgan Kaufmann.

Rosenbloom, P.; Laird, J.; and Newell, A. 1987. Knowledge-Level Learning in SOAR. In Proceedings of the Sixth National Conference on Artificial Intelligence, 499–504. Menlo Park, Calif.: American Association for Artificial Intelligence.

Rosenbloom, P.; Laird, J.; McDermott, J.; Newell, A.; and Orciuch, E. 1985a. R1-SOAR: An Experiment in Knowledge-Intensive Programming in a Problem-Solving Architecture. *IEEE Transactions on Pattern*

Analysis and Machine Intelligence 7(5): 561–569.

Rosenbloom, P.; Laird, J.; Newell, A.; Golding, A.; and Unruh, A. 1985b. Current Research on Learning in SOAR. In Proceedings of the 1985 Machine Learning Workshop, 163–172. New Brunswick, N.J.: Rutgers University.

Rychener, M. 1983. The Instructable Production System: A Retrospective Analysis. In *Machine Learning: An Artificial Intelligence Approach*, eds. R. Michalski, J. Carbonell, and T. Mitchell, 429–459. San Mateo, Calif.: Morgan Kaufmann.

Rychener, M. 1980. Approaches to Knowledge Acquisition: The Instructable Production System Project. In Proceedings of the First National Conference on Artificial Intelligence, 228–230. Menlo Park, Calif.: American Association for Artificial Intelligence.

Rychener, M. 1976. Production Systems as a Programming Language for Artificial Intelligence Applications, Technical Report, Computer Science Dept., Carnegie Mellon Univ.

Rychener, M., and Newell, A. 1977. An Instructable Production System: Basic Design Issues. In *Pattern-Directed Inference Systems*, eds. D. Waterman and F. Hayes-Roth, 135–153. San Diego, Calif.: Academic.

Sacerdoti, E. 1974. Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence* 5(2): 115–135.

Schultz, J., and Davis. L. 1979. The Technology of PROMIS. *Proceedings of the IEEE* 67:1237–1244.

Selfridge, O. 1959. PANDEMONIUM: A Paradigm for Learning. In Proceedings of the Symposium on the Mechanization of Thought Processes, 511–529. London: H. M. Stationery Office. (Reprinted in 1966. *Pattern Recognition: Theory, Experiment, Computer Simulations, and Dynamic Models of Form Perception and Discove*ry, ed L. Uhr. New York: Wiley.)

Simon, H. 1991. *Models of My Life*. New York: Basic Books.

Simon, H. 1969. *The Sciences of the Artificial*. Cambridge, Mass.: MIT Press.

Simon, H., and Newell, A. 1986. Information Processing Language V on the IBM 650. *Annals of Computing* 8:47–49.

Simon, T.; Newell, A.; and Klahr, D. 1991. A Computational Account of Children's Learning about Number Conservation. In *Concept Formation: Knowledge and Experience in Unsupervised Learning*, eds. D. Fisher, M. Pazzani, and P. Langley, 423–462. San Mateo, Calif.: Morgan Kaufmann.

Soloway, E.; Bachant, J.; and Jensen, K. 1987. Assessing the Maintainability of XCON-IN-RIME: Coping with the Problems of a Very Large Rule Base. In Proceedings of the Sixth National Conference on Artificial Intelligence, 824–829. Menlo Park, Calif.: American Association for Artificial Intelligence.

Steier, D., and Newell, A. 1988. Integrating Multiple Sources of Knowledge into DESIGNER-SOAR, an Automatic Algorithm Designer. In Proceedings of the Seventh National Conference on Artificial Intelligence, 8–13. Menlo Park, Calif.: American Association for Artificial Intelligence.

Steier, D.; Laird, J.; Newell, A.; Rosenbloom, P.; Flynn, R.; Golding, A.; Polk, T.; Shivers, O.; Unruh, A.; and Yost, G. 1987. Varieties of Learning in SOAR: 1987. In *Proceedings of the Fourth International Workshop on Machine Learning*, ed. P. Langley, 300–311. Amsterdam: Kluwer.

Tambe, M.; Newell, A.; and Rosenbloom, P. 1990. The Problem of Expensive Chunks and Its Solution by Restricting Expressiveness. *Machine Learning* 5(4): 299–348.

Tambe, M.; Kalp, D.; Gupta, A.; Forgy, C.; Milnes, B.; and Newell, A. 1988. SOAR/PSM-E: Investigating Match Parallelism in a Learning Production System. In Proceedings of the ACM/SIGPLAN Symposium on Parallel Programming: Experience with Applications, Languages, and Systems, 146–160. New York: Association of Computing Machinery.

Walton, P.; Holland, R.; and Wolf, L. 1979. Medical Guidance and PROMIS. *Computer* 12: 19–27.

Waterman, D., and Newell, A. 1971. Protocol Analysis as a Task for Artificial Intelligence. *Artificial Intelligence* 2: 285–318.

Whitehead, A., and Russell. B. 1935. *Principia Mathematica*. Cambridge: The University Press.

Yost, G. 1992. TAQL: A Problem Space Tool for Expert System Development. Ph.D. Diss., School of Computer Science, Carnegie Mellon Univ.

Yost, G., and Newell, A. 1989. A Problem-Space Approach to Expert System Specification. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, 621–627. Menlo Park, Calif.: International Joint Conferences on Artificial Intelligence.

Paul Rosenbloom and **John Laird** received their Ph.D. degrees in computer science from Carnegie Mellon University in 1983; Allen Newell was their thesis adviser. Since 1983, Rosenbloom, Laird, and Newell have formed the core of the team that has been developing SOAR as both an integrated intelligent system and a unified theory of human cognition. Rosenbloom is currently an associate professor of computer science and a project leader in the Information Sciences Institute at the University of Southern California. Laird is currently an associate professor of electrical engineering and computer science at the University of Michigan. Their primary research focus is the development and application of architectures capable of supporting general intelligence, in particular, the SOAR architecture. Related to this overall focus are interests in machine learning, problem solving and planning, models of memory (and their implementation), robotics, autonomous agents in simulation environments, expert systems, neural networks, and cognitive modeling.