REPORT ON A WORKSHOP IN NEW TECHNIQUES IN COGNITIVE RESEARCH

held at Carnegie-Mellon University
21-29 June 1972

A. Newell, H. A. Simon, R. Hayes, and L. Gregg

Departments of Psychology and Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania

Report on a Workshop in New Techniques in Cognitive Research held at Carnegie-Mellon University, 21-29 June 1972

A. Newell, H. A. Simon, R. Hayes, and L. Gregg

In June 1972 an intensive workshop was held at CMU under the auspices of the Mathematical Social Sciences Board (MSSB),* in which a number of cognitive psychologists explored and used some new computer techniques for doing research on cogntive processes. This report provides a brief statement of the goals, design issues and arrangements for the workshop plus some description of how it all worked out.

Background. Since the mid-fifties there has been continuous development in experimental psychology of the view that man is a processor of information (see Miller, Galanter and Pribram, 1960; Reitman, 1965; Newell and Simon, 1972; also compare Broadbent, 1958 with 1971). This growth has taken a variety of forms, from providing the conceptual framework within which to ask experimental questions, to constructing computer programs to simulate specific aspects of cognitive behavior. The computer has always played a dual role in this: as a major source of the conceptual ideas about information processing systems and as a means for constructing such systems and exploring their behavior by simulation.

The problem of communicating and understanding research results has always been acute in information processing psychology. Starting in the late fifties the Social Science Research Council sponsored a number of

---

* The Mathematical Social Science Board is an autonomous group devoted to the encouragement of the use of mathematical and quantitative techniques in the Social Sciences. Its primary means currently are small scientific conferences and workshops. Its support derives from an NSF Grant (GS-3256) to the Center for the Advanced Study in the Behavioral Sciences. Additional support for the Workshop was provided by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44610-70-C-0107), which is monitored by the Air Force Office of Scientific Research, and by the National Institute of Mental Health Grant (MH-07722).

of intensive summer sessions on computer simulation at the RAND Corporation (1958, 1960, 1962)* under a general program that was the direct progenitor of the current MSSB. The emphasis in these early summer sessions was on simulation programs and the use of list languages.

Two concerns provided the impetus for the present workshop. One is the continued growth, in complexity and sophistication, of the computer programs used for the study of information processing. This includes direct simulations of cognitive behavior, basic studies in artificial intelligence, and programs that aid the analysis of cognitive data. The characteristics of these current programs appear to be that: (1) each is an embodiment of some specific psychological content; (2) each permits substantial variation and modification; (3) each has a language of interaction, which gives it some of the flavor of a programming language, but a language that speaks directly in psychological terms; (4) each is interactive, so that the user modifies and explores an existing system, rather than creating something from scratch; and (5) each is a large program.

The second concern is the continued ineffectiveness of scientific communication of the content of these various programs via the standard means of papers, hour-long lectures and even the half-day long sessions that characterize the small invitational working conference. One difficulty is that the underlying technical systems of computer science are

---

* There have been other intensive summer sessions on computers for the behavioral scientists during the intervening decade, though in general they have been concerned with the entire spectrum of computer use.

not fully assimilated by the psychological community. This was the underlying motive for the intensive summer sessions in the fifties and sixties (both for mathematics and for computer science). But the difficulties lie also in the nature of the systems -- their size, complexity and the detailed knowledge necessary for understanding and evaluation. These latter difficulties can be seen at work even in sessions of experts.

In any event the time seemed appropriate for another attempt at an intensive session. A plan for the workshop, submitted to the October 1971 meeting of the MSSB, was approved and the four authors of the present report formed themselves into a steering committee to see the enterprise through.

Goals, Difficulties and Design Decisions. The goals of the workshop were to examine the existing state of the computer technology for cognitive research, working within an information processing approach. The task was not to introduce notions of man as an information processor to those still largely unfamiliar with it, but to introduce new tools to those already working within the field. Thus, established workers in cognitive psychology were invited to participate. There were 20 participants, whose names and affiliations are listed in the appendix.

The major thesis behind the workshop - working with actual programs rather than talking about them - posed the main problems of organization and preparation. First, substantial access to a good computer was required. This dictated to all intents that the workshop be held at CMU, the site of the organizers.

A more important issue was the selection of programs. Only a few could be included, but this implied an undesired emphasis on the specific ones selected, for the goal of the workshop was to convey in depth the whole range of things that could be done. The selection problem was compounded by the requirement that the programs run at a given site (CMU), which implied that the assemblage of programs would exhibit undue provincialism. Any attempt to be very eclectic posed difficult problems of getting complex programs up and running on foreign computer systems.

The solution was a compromise: to invite one scientist from outside CMU to participate and to bring his program up on our CMU system. We invited Terry Winograd at the Artificial Intelligence Lab at MIT to join us with SHRDLU, his program for exploring the understanding of natural language. Other than Terry's, all the programs were in active use in the Psychology and Computer Science Departments of CMU. We initially started with a set of about a dozen programs, intending to eliminate some along the way. The final number was seven. We list them here briefly, since we need to refer to them below. Somewhat expanded descriptions are provided in the Appendix.

PAS-I (Protocol Analysis System I). A system for analysing the verbal protocol of a subject solving a cryptarithmetic puzzle to infer the knowledge state of the subject at each instant in time.

PAS-II (Protocol Analysis System II). A system patterned after PAS-I, but freed of the constraint to work only on cryptarithmatic.

MAPP (Memory and Perception Program). A model and simulation of storage of patterns (chunks) in LTM and their interaction with perceptual tasks. (Applied mainly to chess perception.)

PSG (Production System, version G). A system used to model and simulate the control aspects of cognitive processes. (Applied mainly to cryptarithmetic and STM tasks such as the Sternberg paradigm.)

SHRDLU. A system for understanding natural language within a small world (the world of blocks on a table).

CLS (Concept Learning Systems). A combined experimental and simulation system for operating standard concept attainment experiments (e.g., Neisser and Weene paradigm) on a small laboratory computer, while running a simulation of a subject in the main computer.

ATS (Semi-automatic Transcription System). A system used to aid transcription of audio tapes (for protocols), in which the computer segments the speech, displays it (auditorially), permitting easy control and editing, and obtaining timing to centiseconds.

The third and most severe problem was how to make possible useful contact between the participants and these several programs. The difficulties were three-fold. (1) The programs themselves are complex and a few days is a short time in which to understand them. (2) The participants are generally knowledgeable about information processing but are not experts in programming, and the gap between the participants programming knowledge and the complexity and style of the programs was uncomfortably large. (3) The programs themselves are all research efforts, unpolished for user use. The programs are all specific systems that do something of cognitive interest. Furthermore, they require substantial interaction, since they can be specified to do a variety of interesting tasks within their general domain.

To compound this third problem the programs were written in a variety of higher level languages. The seven systems were written in six and half languages: (Stanford Lisp, Mac Lisp, Snobol 4, Fortran, SAIL (the Stanford AI Language), L* (a CMU system building language) and APCOL (the

language on the laboratory computer). In addition, all the program have specialized sublanguages for run-time interaction. Furthermore, only a few of the participants had had any experience operating on a PDP10, the computer system on which the workshop would run (actually, one part was on the small laboratory machine in Psychology, the DDP116, coupled to a 360/67).

The solution adopted to this third problem had three components. First, the authors of each of the individual systems agreed to spend some effort during the year making their systems more public (thus generating at least one by-product of social value). Second, it was proposed to put substantial effort into polishing the structure as a whole, making sure that all programs were in top working shape, that interaction with them was convenient, and that aids and documentation were available. To this end, the MSSB provided budget for one man-year of programming and the rental of enough terminals to prevent a bottleneck at the interface. (Such costs, especially the amount for programming, depart from the usual costs associated with MSSB workshops, and thus put the entire program in a frankly experimental category.)

The third component was to have an adequate staff available during the workshop who were sophisticated generally about the total system, while each was expert on some aspect of it. The magic rule seems to be one staff member per participant (thus if all else goes awry, all participants can be personally diverted and entertained). In the end we obtained a staff of twenty-one, made up of the authors of the systems, graduate students of Computer Science and Psychology (including a graduate student from MIT

working with Winograd), and staff from Computer Science and Psychology. They are listed in the Appendix.

Substantial efforts went into all of the programs to put them into shape for the workshop, even to the extent of dictating the direction of some research efforts over the year of workshop preparation. For instance, the decision to create PAS-II, an interactive and task-free version of PAS-I, was made essentially to mesh with the demands of the workshop.

Detailing each of the preparatory activities would not be illuminating, but a sketch of the efforts required to bring SHRDLU up on the CMU PDP10 will indicate the kinds of problems that had to be solved. While SHRDLU was programmed to run on a PDP10, the MIT and CMU systems differ with respect to the monitor, the assembler, and the dialect of LISP that are employed. A program did exist for translating, though laboriously, from the one LISP dialect to the other; and this was used to bring up a first version of SHRDLU on the CMU machine.

However, SHRDLU itself was in the process of being reprogrammed at MIT. We wished to use the newest version, but not to translate, painfully, each revision. Two steps were taken to alleviate the difficulties A member of our staff (Stu Card) worked with Winograd for six weeks at MIT helping to construct the revised SHRDLU. In addition, the MIT

* We would like to express publicly our thanks to this staff for the tremendous outpouring of energy and devotion to duty, which had so much to do with making the workshop a success. In addition to this staff we also had secretarial and office support (Mildred Sisko and Steve Lewis), as well as in-depth support from our computer operations (especially Paul Newbury). We wish to express our thanks to these people as well.

assembler was converted to run under the CMU monitor, thus permitting nearly effortless program translation. This conversion absorbed two man-months of effort, but left behind another valuable byproduct -- the converted assembler.*

Besides the efforts to polish the individual programs, we decided to incorporate all of the programs within a submonitor-like system which would buffer the participants both from variations in language and style inherent in seven separate major programs and from the PDP10 monitor system. This system, which came to be called ZOG, was the major programming effort designed to make communication possible. The next section describes it.

ZOG. As indicated, the problem was seven (i.e., an indefinite number) major programs, written in seven (i.e., an indefinite number) of diverse languages, each with a special sublanguage of interaction, embedded in a monitor with its own conventions, with support facilities (e.g., editing systems) written in yet other languages. The participants were somehow to be able to penetrate to the content of these systems almost instantly. If they were really to do so, as opposed to simply watching the output of some canned runs, they had also to create files of data, edit them, provide some additional specification within each system and even do some programming.

The solution was to create another system that would be the primary media for interaction between the user and the programs, this system to

* Anyone who is running on the DEC 10-50 monitor and wishes to use MIDAS should contact George Robertson, Computer Science Department, Carnegie-Mellon University.

provide a uniform and simple way of interacting with aids to help the participant understand what was going on. The new system was christened ZOG.* We considered two basic choices for its design. It could be built around a simple English-like system of commands (e.g., RUN PAS2 ON FILE XYZ, TELL ME ABOUT PAS2, HOW DO YOU EDIT A FILE?, etc.). This has the virtue that the language seems instantly familiar; it has the disadvantage that the total system and its limits are not easily representable to the user and learnable by him.

The alternative, which we chose, can be described as a "key word" approach. If you want to find out about the program, PAS2, type 'PAS2', if about editing, type 'EDIT'. Not everything is designated by a single word: within the totality of matters relating to PAS2, one can inquire about: PAS2 DEMOSTRATIONS, PAS2 INSTRUCTIONS, PAS2 BACKGROUND, etc. The language thus has a modicum of structure, but of the simplest type: define what you want to attend to by a sequence of key words from the most general to the most specific.

This technique leads to a tree-like structure. All communications start at a top mode and a sequence of key words takes one down to a subnode of the tree. When the user has reached a node, he can ask for its list of subnodes -- all the subtopics reachable from the node. Since the user is initially uninformed about the multitudinous aspects of all

---

* The name is not acronymic for anything. It was chosen to be short, pronounceable, capable of personification (e.g., "ZOG told me that ..."), and non-conflicting with other names. The system was designed by George Robertson and A. Newell; it was programmed by George Robertson; and the major effort of creating the user system (the so-called ZOG tree) was done by Phil Karlton. A report on the system is in preparation for submission for publication.

the various programs and facilities, the problem for ZOG is not just to communicate specific items of information, but to help the novice user find out what he wants (or should want) to know about.

To meet this last requirement, we decided to make all the information about the programs available via ZOG. That is, we decided not to have documentation of the usual sort, consisting of manuals and operating guides.* Thus, ZOG was to contain the following types of material:

> Orienting descriptions -- what a given program or facility was.
>
> Demonstrations - evocable with essentially no knowledge on the user's part.
>
> Operating instructions -- how to execute the programs and facilities to explore them.
>
> Suggestions -- how to proceed in order to get into something of interest.

As to content, ZOG should contain information about the following sorts of things:

> The seven major programs.
>
> ZOG itself.
>
> Basic facilities, such as editing, checking the status of the systems, making suggestions, etc.
>
> Miscellaneous other available programs that might be of interest or just fun -- e.g., chess programs. This branch of the ZOG tree came to be called the Sideshow.

---

* We did send to the participants prior to the workshop some of the scientific papers concerned with each of the major programs. Thus, there was some general substantive knowledge about the domains of each of these programs. We sent nothing in the way of documentation and operating guides, so that the participants arrived totally innocent as far as making contact with the living programs was concerned. The materials sent were: Chase and Simon (1972), Newell (1971), Newell and Simon (1972), Simon and Gilmartin (1972), Waterman and Newell (1971), Winograd (1972).

By way of written documentation, we produced both an index to the ZOG tree and a complete printout of its contents, called the ZOG book. The ZOG index, included as an appendix to this report, contains about 350 nodes.

Since ZOG itself is an interactive system that had to be understood, even to explore it, we attempted to keep its form genuinely simple. We provided five basic actions. If a user had reached a specific node in the ZOG tree, say SYSTEMS PAS2, he could do five things:

? This prints out the names of the subnodes of PAS2

?? This prints out all the textual explanation associated with the node, PAS2.

! This executes the proper action associated with the node, e.g., runs PAS2.

↑ This goes back up the next higher node, here the node SYSTEMS.

↑↑ This goes back up the top of the ZOG tree (called TOP).

The Appendix provides a number of examples of ZOG's operation, illustrating a number of additional details that are critical for the system to be useful.

Besides these general operational aspects, some systems details were extremely important. For instance, the user should not be required to understand the PDP10 monitor system, hence, all errors, etc., should return to ZOG, rather than to the PDP10 monitor. To accommodate many simultaneous users, ZOG had to be sharable (i.e., all users access a single running ZOG program).

A final design requirement concerns the building of the ZOG tree, for the necessary expertise was distributed throughout the scientists who

had written the various programs. These people, though sophisticated programmers, knew nothing of ZOG as a program. Thus, a system of commands had to be added to a builder's version of ZOG (called BZOG) to permit easy construction of the tree. Finally there had to be a process for assembling user's ZOG from all the separate versions of builder's ZOG that had each been worked on in isolation. This last act, of course, could be performed only by a person who was truly sophisticated in ZOG (he became known as "the Forester").* Add to the above that ZOG had to operate with the reliability and freedom from error of a well-used monitor system. Thus, ZOG turned out to be a major act of software design and construction.**

Terminal Arrangements. A few words on physical arrangements for access to the computer is appropriate. Communication with the PDP10 is entirely via on-line terminals. The terminals are mainly Model 33 teletypes, relatively slow 10 char/sec devices. We rented 8 30 char/sec hardcopy (typewriter) terminals (Datanet 300) on the principle that novices really want an exact past record of what happened, to mull over and to use as a guide for the next session on the machine. In retrospect this decision appears entirely correct.

---

* This was the role played by Phil Karlton.

** Again, details will be forthcoming in the report in progress. Design and construction of the basic ZOG system took essentially one man-month, (ZOG was coded in L*). Several man-months went into the building of the ZOG tree. The Forester worked essentially full time for a month prior to the workshop.

The arrangements for the main room in which all meetings of the whole would take place was, if anything, even more critical. The Computer Science Department has a permanently assigned classroom which is wired for communication with the computer. We commandeered this room for the entire workshop. It was arranged as shown in Figure 1, a large rectangular table around which all the participants and staff could sit (35 people). At the head of the table (actually the lower corner) there was an alphanumeric scope hardwired to the computer, and capable of producing a fresh screen of text in a few seconds. Around the inside of the table were placed a number of TV monitors, slaved to the main scope, that allowed everyone direct and easy viewing of what was going on. Another large monitor was placed at the head of the room, so that the speaker could point at parts of the display. In addition to the main scope, we kept another active hardcopy terminal in the room to permit a second line of access to the machine, and also a telephone line to the PDP10 operator. There was also an audio speaker, since some of the programs (ATS) had voice output.

The Dress Rehearsal. In order to be sure that all would be ready on time, and to exhibit the system to other faculty members and graduate students in Computer Science and Psychology, we held a three-day dress rehearsal on the 26 - 28 May. This consisted of two hours demonstration-lecture on each of the seven systems plus ZOG. The rehearsal was open to all interested people (including some from neighboring institutions). Thus, it provided a genuine audience and forced a major effort to achieve completion.

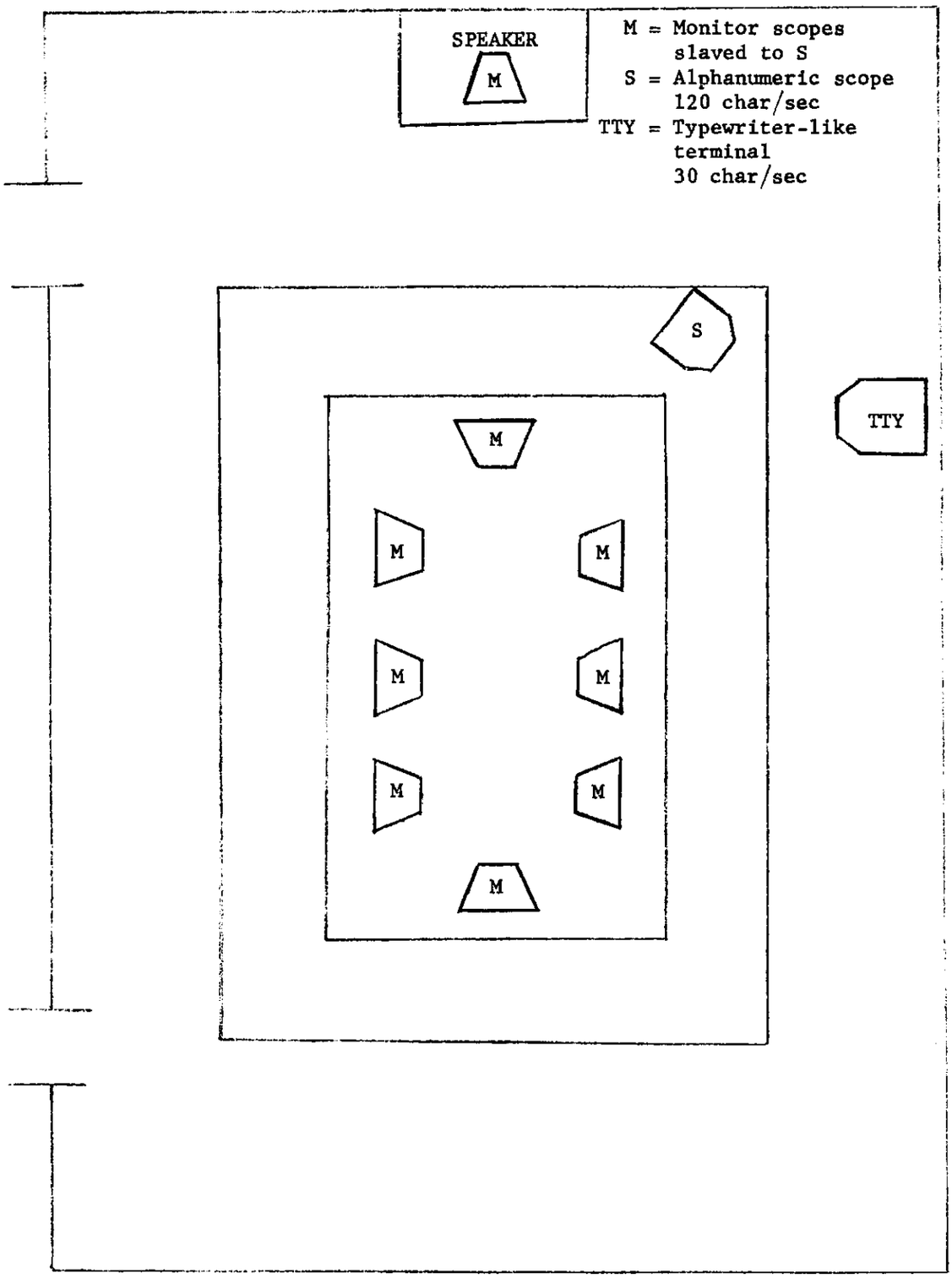
SPEAKER
M
M = Monitor scopes slaved to S
S = Alphanumeric scope 120 char/sec
TTY = Typewriter-like terminal 30 char/sec
S
TTY
M
M
M
M
M
M
M
M
M

Figure 1.

This dress rehearsal played a major role in bringing the system to fruition. All programs had been brought to working order by the Dress Rehearsal, and the four subsequent weeks were devoted to efforts at improvement and completion. (However, the work of building the ZOG tree was done in June, thus affording no opportunity to polish and fill out the tree.)

The Session. The course of the workshop itself is almost anti-climactic, given all the details of preparation. The duration had been set at nine days, with a view to making it highly intensive (it was suggested that families not come and all participants were housed together on campus). The general schedule was designed around morning, afternoon and evening sessions-for all nine days, from Wednesday through the weekend until the following Thursday.

The initial conception of the schedule was simple: (1) demonstration lecture sessions on each of the systems as quickly as possible to make the participants aware of what was available; (2) interlaced sessions on the machine right from the start, so participants could begin exploring the programs themselves; (3) leave open for future determination the exact schedule after the initial period.

Besides the main programs three additional events were put into the schedule. One was a demonstration of an eye movement apparatus being used by John Gould. It was thought that some of the participants might want to generate some behavioral data during the nine days and analyse it using

Visiting at CMU for the academic year from IBM.

the various tools available. The second was a demonstration of the Speech-Understanding System being developed at CMU in Computer Science (Reddy, Erman and Neely, 1972). This program (in its initial task) attempts to play chess by voice with a human user, making use of acoustic, lexical, syntactic and semantic components (the last, a chess program). The third event was a session on non-verbal protocol analysis by Richard Young (a graduate student in psychology) of TV tapes of young children seriating blocks. This was of interest, since most other work on protocol analysis has been done with verbal materials.

After the introduction of the programs it was decided to dissolve entirely into small groups and individual efforts. There were no meetings of the whole again until the last day for the final wind-up session, though smaller meetings of subgroups occurred continually.

How It All Worked Out. In the eyes of both the participants and the organizers the workshop was a success. Such an overall judgment is necessarily subjective, for the ultimate success of any such scientific event must always be measured in terms of long range consequences for future scientific work. No matter what the immediate glow of satisfaction, these consequences are only revealed by the passage of time.

To a first approximation ZOG and all of the programs functioned as advertised throughout the workshop. The PDP10 operated thoughout the entire period (0900-0600) without perceptable failure.* Though there were

* This latter qualification is necessary, since large time sharing systems do occasionally crash and are brought back up within a couple of minutes without essential loss of data. As long as there are few enough of these, they are simply absorbed by the user community as an acceptable part of normal operations. The actual crash rate was 1.1 per day.

a large number of hours of demonstration (about 20), there were no major failures during these demonstrations.*

We had started out with a dozen or more programs as serious candidates for the workshop, and eliminated those that did not make the deadlines. Thus, the seven programs that got into the workshop were those that were in fact operational. (However, the menu was sufficiently rich so we probably should not have had more final entrants in any case.) We committed ourselves to three major programming efforts: ZOG, PAS-II (as a redesign of PAS-I), and the revision of SHRDLU; and the effort in achieving these programs did strain our resources. Explicit deficiencies in the workshop can be traced to each of the three efforts.

The new version of SHRDLU was operational at the start of the workshop, but the full array of demonstrations and guides was not operational until the last couple of days. The effort to make SHRDLU run on the CMU 10 (the conversion of MIDAS and MAC Lisp) was sufficiently expensive that it captured much effort that was to go elsewhere, and delayed ZOG for at least a month. The delay on ZOG resulted in a delay of efforts to redo PSG (the production system). The new PSG did not get its command language until half way through the workshop, and this also translated to an initially minimal ZOG subtree for PSG.

PAS-II, though it became available in time for the workshop, did not become available early enough for others beside the designers to do

---

* There was one exception for the workshop as a whole. During the second demonstration of the CLS system, which ran on the DDP116 and the IBM360/67, the 67 crashed and took 45 minutes for recovery.

much exploration with it. Thus, we did not have the extensive experience with the new version that we had hoped for and this translated directly into what could be said about the program during the workshop. In sum, all the programs functioned but were not in fact as polished as they could be and the effects of this could be seen in the amounts of individual instruction that were required in dealing with the incomplete parts.

The idea of the ZOG interface proved successful. Essentially all the participants were on the machine within the third hour after starting the workshop on Wednesday morning. Several had never been on-line with a large system before. But ZOG provided ways in which small responses by the participants led to meaningful responses by ZOG, which then permitted interaction to explore the ZOG tree and execute a program or two.

Again, we have no fine grained evidence against which to maasure the success of ZOG, only the gross evidence that the participants immediately got into the system and interacted with the 10 at an extremely high rate throughout the entire workshop. As a counter against assigning ZOG too much credit, we did have a large number of staff members available to serve as tutors and to correct bugs. Thus, we have no test of whether ZOG alone could have produced the rapid acclimatization of the participants.

The statistics on the usage of the 10 show that the average number of connect hours per participants (for the 20 participants) was 47 hours for the 9 days, or about 5 hours a day. An average of 78 minutes of central processor time was used per participant over the workshop, or about 8 minutes per day. These numbers include just the participants, not the staff; thus they do not cover the use of the machine during the demonstrations.

They underestimate the time spent at the console in that, often two (and occasionally three) people worked together at a single terminal. On the other side, they overestimate it for a few people who learned the trick of logging in on several terminals in order to run in parallel on several tasks. As is true of all computing, the variance in use was high, a few people logging almost 12 hours a day and using about 30 minutes of processor time per day.

One of the major unknown factors in planning the workshop was how many participants could be accommodated. Twenty proved fine for the demonstrations and the interactions with the staff assembled. It was too many for the computer system. The system was essentially loaded to capacity at all times (though it became tolerable in the wee hours toward the end of the workshop). The workshop commandeered the computer and the terminal room in the morning from 0900 to 1200, but then shared the system with the regular users (Computer Science Department) during the rest of the day.*

One important effect of the loading was to make ZOG less useful, since the rate of interaction became strikingly slower as the system approached capacity. Gradually, less and less purely textual exploration of ZOG occurred and the ZOG indexes and books took over most of this function. The loaded system forced this, but it probably would have

---

* The workshop was a major imposition on the Computer Science Department, almost putting an end to useful work on the machine during half of June. Faculty and students accepted the restriction cheerfully and we wish to express our gratitude to them for their cooperation.

occurred in any event at the data rates possible with our terminals (30 char/sec) and with growing familiarity with the system. On the other hand, there remained substantial local use of ZOG text to determine how to do something at the point when the user wanted to do it, but couldn't quite recall how or had tried and done it wrong.

The participants explored the set of programs differentially. There was, of course, an initial transient -- large numbers of people explored program X right after the demonstration of X. But gradually they began to make decisions about where to concentrate. It was clearly not possible to deal intimately with all the programs. The participants largely moved in their own independent ways. For example, by design no formal attention in the workshop was devoted to the underlying programming languages (e.g., Lisp). However, several participants who had no prior exposure to list languages decided to learn Lisp. Some additional material was added to the ZOG tree on Lisp, and a small tutorial group was generated to discuss Lisp programming. Thus, the workshop was moved in whatever direction seemed appropriate to the participants.*

---

* The workshop had been deliberately designed to be this way. In particular, ZOG had been conceived as a structure that permitted this. However, ZOG proved to be somewhat inflexible in this regard. User's ZOG was non-modifiable (since it was shared by all the users); thus we could not add new branches and leaves to the tree freely and quickly in response to the needs of the participants. Instead we had to cumulate a number of changes and then reassemble the tree. This latter turned out to be a major operation, which we only managed twice during the workshop. This inflexibility in ZOG is mostly a correctable design failure on our part, though it is complicated by basic limitations in the DEC monitor.

The CLS system proved to be a special case. It was located in the Psychology Department in a separate building and made use of a laboratory computer (DDP116) and the regular campus computer (360/67). All of these characteristics put barriers in the way of exploiting it, since the main arena of activity and learning was elsewhere. We were aware of this in the planning stages, but the plans to couple the DDP116 to the PDP10, thus tying them all together for the participants, were one of the items that had to be abandoned.

It is apparent from the foregoing that the workshop was an extraordinarily busy affair and that the participants exploited it fully. However, two issues existed throughout that were never fully resolved, one procedural and one substantive. Procedurally, there was a tension between the demonstrations, in which the participants were passive, and the working sessions, in which the participants could proceed actively. People felt they were not having enough time on the machine, especially at the start. Should the demonstrations be shorter with more time available on the computer? But then some programs would have to wait until late in the workshop. The issues were discussed explicitly several times, seeking a better solution, but no alternative preferable to the original schedule was found.

The substantive issue concerned whether there should have been more discussion of the psychological assumptions, content and implications of the various programs. The main programs in the workshop were all genuine expressions of particular psychological theories and hypotheses.*

* The exceptions were ATS, the transcription system, and the eye movement laboratory. Indeed, these were treated by the workshop as of lesser concern than the programs with substantive content. (Note that the protocol analysis system, though billed as a data analysis scheme, incorporates a large amount of psychological theory of a particular stamp.)

Though each of the demonstration-lectures did talk about the psychology somewhat, this was clearly subordinated to the agenda of getting to know and use the program. There was never any cycling back to the programs in major discussions, though there was, of course, individual talk about them.

Cost. The expenses of the workshop can be divided into two parts: operating costs and development costs. On the operating side there are the travel and living costs of the participants, the costs of the staff, and the costs of the computer. The only interesting cost figure here is the cost of the computer. The high cost of staff is best measured simply by counting heads. The cost of the computer was already quoted earlier in terms of connect time and processor time for the participants. In terms of the costing scheme used on the computer system* this amounts to $16,277, which is $814 per participant. If the capacity had been adequate, so that the response time had not degraded, the participants probably would have used two to three times as much processor time, though little more connect time.

Development costs are again measured in staff time and in computer use. Staff time is ambiguous, since all of the polishing on the major programs also contributes to their stature as scientific efforts and can hardly be allocated entirely to the workshop. It is probably fair to view the month prior to the workshop (from about mid May) as being devoted almost exclusively to preparation for the workshop by a dozen people.

In terms of computing use, the aggregate staff computing on the 10 for the workshop amounted to $39,535 (composed of 87 hours of processing

* Which charges for memory use as well as connect and processor time.

time and 1500 hours of connect time) • This was distributed among 15 people, four of whose usage of the PDP10 programs differed little from those of the participants. This usage was distributed over about six months, but most of it occurred in the two months of May and June. Basic work on some of the programs, e.g., PAS-II and MAPP, was carried on outside of the workshop environment until the last couple of months. To this we must add the effort put in at MIT on SHRDLU, and on the CLS system on the DDP116-360/67.

Exact accounting of the costs is unnecessary, since the event is unique in many ways and even if repeated (whatever that might mean) would have an entirely different cost structure. It can be concluded that such an event is indeed expensive, compared with normal workshops. (Perhaps the best summary view is that it required roughly a third of a large PDP10 system for a two month period) . This expense is overestimated because of the familiar phenomena of very high costs of development, which are not all to be accounted to the single short term event. However, such efforts at communication will always have a substantial developmental cost, even if they should become a regular occurrence. The progressive nature of science guarantees that.

Conclusion. The workshop demonstrated clearly that a group of working psychologists, not themselves computer experts, could assimilate, with an effort that was intensive but limited in duration, a set of highly complex and sophisticated research programs and could make substantial headway in assessing these tools in relation to their own research. The necessary condition (beside the motivation of the participants) was a highly instrumented environment with adequate computer resources operating

in a workshop-like fashion. The workshop indicates some ways in which scientists can communicate effectively about research techniques embedded in highly complex computer programs and about the scientific results that depend on these programs.

The workshop was short enough so that it remained primarily a communication event. However, even within its limited scope, one could see the positive effect of being able to work with adequate computer tools and resources. Implicit in the success of the workshop is a style of operation for experimental cognitive psychology that has such tools and resources available as continuous working tools in the day to day course of research.

References

1. D. Broadbent, Decision and Stress, Academic, 1971.

2. D. Broadbent, Perception and Communication, Pergamon Press, 1958.

3. W. Chase and H. A. Simon, Perception in Chess, CIP Working Paper 182, Psychology Department, Carnegie-Mellon University, 1971.

4. K. Colby and D. Smith, "Dialogues between humans and an artificial belief system," Proceedings of the International Joint Conference on Artificial Intelligence, Washington, 1969.

5. A. de Groot, Thought and Choice, Mouton, 1965.

6. L. Gregg and H. A. Simon, "Process models and stochastic theories of simple concept formation," Journal of Mathematical Psychology, 4, 1967, 246-276.

7. D. Klahr and J. G. Wallace, "Class inclusion processes," in S. Farnham-Diggory (ed.) Information Processing in Children, Academic, 1972.

8. M. Levine, "Hypothesis behavior by humans during discrimination learning," Journal of Experimental Psychology, 71, 1966, 331-338.

9. G. Miller, E. Galanter and K. Pribram, Plans and the Structure of Behavior, Holt, Rinehart and Winston, 1960.

10. M. Minsky (ed.) Semantic Information Processing, The MIT Press, 1968.

11. U. Neisser and P. Weene, "Hierarchies in concept attainment," Journal of Experimental Psychology, 71, 1966, 640-645.

12. A. Newell, "A theoretical exploration of mechanisms for coding the stimulus," in A. W. Melton and E. Martin (eds.) Coding Processes in Human Memory, Winston, 1972, 125-139.

13. A. Newell and H. A. Simon, Human Problem Solving, Prentice-Hall, 1972.

14. R. Reddy, L. Erman and R. Neely, Working Papers in Special Recognition, Computer Science Department, Carnegie-Mellon University, 1972.

15. W. Reitman, Cognition and Thought, Wiley, 1965.

16. R. F. Simmons, "Natural language question-answering systems: 1969," in R. Banerji and M. Mesarovic (eds.) Theoretical Approaches to Non-numerical Problem Solving, Springer-Verlag. 1970.

17. H. A. Simon and M. Barenfeld, "An information processing analysis of perceptual processes in problem solving," Psychological Review, 76, 473-483.

18. H. A. Simon and K. Gilmartin, A Simulation of Memory for Chess Positions, CIP working paper 206, Psychology Department, Carnegie-Mellon University, 1972.

19. H. A. Simon and L. Siklossy, Representation and Meaning: Experiments with Information Processing Systems, Prentice-Hall, 1972.

20. E. Tulving and W. Donaldson (eds.) Organization of Memory, Academic Press, 1972.

21. D. Waterman, "Generalization learning techniques for automating the learning of heuristics," Artificial Intelligence 1, 1970, 121-170.

22. D. Waterman and A. Newell, "Protocol analysis as a task for artificial intelligence," Artificial Intelligence Journal 2, no. 3/4, 1971 285-318.

23. D. Waterman and A. Newell, Preliminary Results with a System for Automatic Protocol Analysis, Department of Computer Science, Carnegie-Mellon University, 1971.

24. T. Winograd, "Understanding natural language," Cognitive Psychology 3, 1972, 1-191.

25. W. A. Woods, "Procedural semantics for a question-answering machine," in Proceedings of the Fall Joint Computer Conference 1968, 33, AFIPS.

Appendix

1. Participants in the Summer Workshop

2. Summer Workshop Staff

3. Announcement: New Technologies for Cognitive Research

4. Summaries of Programs

    Automatic Protocol Analysis
    A Simulation of Perceptual Processes in Chess
    Production Systems
    Semantic Understanding Systems
    Concept Learning System
    Machine-aided Transcription System

5. ZOG

    Index
    Examples

PARTICIPANTS IN THE SUMMER WORKSHOP

Doris Aaronson
Associate Professor of Psychology
New York University

George Baylor
Assistant Professor of Psychology
Universite de Montreal

Robert A. Bjork
Associate Professor of Psychology
University of Michigan

Albert S. Bregman
Associate Professor of Psychology
McGill University

Jerome Elkind
Director, Computer Science Laboratory
Xerox Corporation
Palo Alto, California

James Greeno
Professor of Psychology
University of Michigan

Richard W. Haller
Assistant Professor of Psychology
University of Oregon

L. Rowell Huesmann
Assistant Professor of Psychology
Yale University

Neal F. Johnson
Professor of Psychology
Ohio State University

Paul E. Johnson
Professor of Educational Psychology
University of Minnesota

Kenneth R. Laughery
Professor of Psychology
SUNY at Buffalo

Richard B. Millward
Professor of Psychology
Brown University

John Morton
Member, Scientific Staff
Medical Research Council, Applied Psychology Unit
Cambridge, England

Raymond S. Nickerson
Vice President and Director, Behavioral Science Division
Bolt, Beranek and Newman, Inc.

Gordon Pitz
Professor of Psychology
S. Illinois University

Zenon Pylyshyn
Associate Professor of Psychology and Computer Science
The University of Western Ontario

Jay E. Russo
Assistant Professor of Psychology
University of California at San Diego

Richard M. Shiffrin
Professor of Psychology
Indiana University

Tom Trabasso
Professor of Psychology
Princeton University

Paul C. Vitz
Associate Professor of Psychology
New York University

SUMMER WORKSHOP STAFF

Ruven Brooks - Psychology (graduate student)

Stuart Card - Psychology (graduate student)

William Chase - Psychology (faculty)

Charles Faddis - Psychology (staff)

Kevin Gilmartin - Psychology (graduate student)

John Gould - Psychology (faculty)

Lee Gregg - Psychology (faculty)

John R. Hayes - Psychology (faculty)

Dennis Jazudek - Psychology (staff)

Philip Karlton - Computer Science (graduate student)

David Klahr - Psychology (faculty)

Henry Mashburn - Psychology (staff)

Donald McCracken - Computer Science (staff)

Allen Newell - Computer Science and Psychology (faculty)

George Robertson - Computer Science (staff)

Andee Rubin - MIT Electrical Engineering (graduate student)

Michael Rychener - Computer Science (graduate student)

Herbert A. Simon - Psychology and Computer Science (faculty)

Donald Waterman - Psychology (research associate)

Terry Winograd - MIT Electrical Engineering (faculty)

Richard Young - Psychology (graduate student)

cognitive workshop general information

NEW TECHNOLOGIES FOR COGNITIVE RESEARCH

A SUMMER WORKSHOP

AT CARNEGIE-MELLON UNIVERSITY

PITTSBURGH, PENNSYLVANIA

21 - 29 June 1972

This brief, intensive workshop will examine and discuss recently developed techniques for studying human information processing at the cognitive level:

INFORMATION PROCESSING MODELS OF COGNITION

TECHNIQUES OF DATA ANALYSIS

REPRESENTATION OF COGNITIVE STRUCTURES

ANALYSIS OF TASK ENVIRONMENTS

The models and techniques to be examined will be exemplified in computer programs available to the workshop participants for manipulation and modification in an interactive time-sharing environment. The examples will include simulation programs and data analysis programs (on a PDP-10 and/or an IBM 360/67), as well as programs for running cognitive experiments with laboratory subjects (on a DDP-116).

Systems that will be available for study and experimentation are:

An automatic system for protocol analysis;

A concept formation system, for human experiments in the laboratory, and for simulation;

Production systems for expressing cognitive theories (e.g., cryptarithmetic problem solving);

A system for understanding natural language;

A system for simulation phenomena of visual perception and memory in chess;

A system for machine-aided transcription of audio tapes;

A computer- controlled laboratory system.

PARTICIPANTS. Participation will be invited from psychologists and computer scientists active in rosoarch on human cognitive behavior. General familiarity with information processing psychology will be assumed, as will (substantial) experience with computers, (but not with the specific languages and computers to be used in the workshop).

Workshop systems will be polished to permit interactive use by persons not initially familiar with them. Prior to the workshop, background material will be circulated.

In order to permit adequate access to the computer facilities, the number of participants will be limited to about 20.

PROGRAM. The workshop will hold sessions of the entire group each afternoon. Evenings and most mornings will be left free, with the expectation that most participants will want to work at the computer consoles with programs, or in the laboratory constructing and testing experimental designs. The workshop will be instrumented as fully as possible, so that participants need not merely talk about programs and experiments, but actually explore and test out concrete techniques and ideas.

Participants will bring to the workshop experimental situations, phemomena and paradigms that are of particular interest to them, and will have an

opportunity, with their colleagues, to investigate the relevance of a range of techniques for their theoretical and experimental concerns.

ORIENTATION. The theoretical orientation of the workshop is best described by the typical contents of the journal COGNITIVE PSYCHOLOGY, or of the newly published book by Newell and Simon, HUMAN PROBLEM SOLVING (Prectice-Hall).

The technical orientation will be toward maximal use of computers and programming languages in formulating and testing theories, running experiments and analysing data.

ARRANGEMENTS. The workshop is being organized in the Departments of Psychology and Computer Science of Carnegie-Mellon University, by a committee consisting of Allen Newell, Herbert A. Simon, John R. Hayes and Lee W. Gregg. It is sponsored by the Mathematical Social Sciences Board under a grant from the National Science Foundation. Other faculty members at Carnegie-Mellon University and Terry Winograd from MIT will collaborate in developing and presenting the programs and systems of instrumentation.

Travel and living expenses will be provided for participants, as well as necessary funds for preparation of materials. Housing for participants will be available on the Carnegie-Mellon campus. (Because of the brief, intensive nature of this conference participants are encouraged not to bring their families.)

Workshop sessions will begin on the morning of Wednesday, June 21, and will run through the afternoon of Thursday, June 29, including the weekend as well as weekdays of this period.

cognitive workshop general information

INQUIRIES about the workshop should be addressed to:

Professor John R. Hayes

Department of Psychology

Carnegie-Mellon University

Pittsburgh, Pennsylvania 15213

Automatic Protocol Analysis

Verbal protocols are a primary form of data for the analysis of sequential problem solving behavior on symbolic tasks. Within the general theory in Newell and Simon (1972), the subject is viewed as following a trajectory in a problem space, where each point in the space corresponds to a particular state of knowledge about the task. The protocol is used to infer the subject's trajectory, i.e., to obtain a description of what the subject knows about the task at each moment. This final trajectory is called the Problem Behavior Graph (PBG).

We currently have a program, PAS-I (Protocol Analysis System I, Waterman and Newell, 1971, 1972), which performs the data analysis described above. It takes as input the transcription of the subject's verbalizations (segmented in units corresponding to approximately one task relevant topic) and provides as output a Problem Behavior Graph. In so doing, it constructs a semantic representation of the subject's utterances and infers therefrom what the subject must have known or done to come by the evident bits of knowledge. PAS-I must know in detail the task what the subject is attempting, for many of its inferences are based on knowledge about the task. Thus, PAS-I works exclusively on cryptarithmetic, a task environment in which our theoretical understanding of humn behavior is well developed (see Newell and Simon, 1972). PAS-I currently performs analyses at a level that is substantively interesting, though not yet quite up to manual analysis.

Systems for automatic protocol analysis have implications for achieving volume analysis of protocols, for obtaining objectivity in analysis and for developing measures of the contribution of various components of the data to the models the data is supposed to test and validate. The programs, being sophisticated inference programs in their own right, also have independent interest in revealing what is going on in understanding natural language.

For the workshop PAS-I will be available and can be explored, used and modified. A second system, PAS-II is also available. It consists of an interactive system that contains the task independent parts of PAS-I and permits the user to provide the task dependent parts. Thus, it provides a machine-aided framework within which to carry out protocol analyses in a range of task environments. The scope of PAS-II is still unknown, but should include tasks such as chess, theorem proving, complex concept formation, etc.

A Simulation Of Perceptual Processes In Chess

This program, MAPP, was designed to simulate chess players of varying skill levels extracting information from a chess position after brief visual exposure to it, holding that information in short-term memory, and using it to replace the pieces on the board. The program reproduces a well-known experimental phenomenon -- that a chess master can do a very much better job of reproducing a position from an actual game than can a weaker player, but not a position constructed by random processes. The program is written in SNOBOL.

The program consists of a learning component and a performance component. The learning component, on exposure to a sequence of partial patterns of pieces on a chess board (e.g., pawn chains), grows a discrimination net (EPAM net) to store these patterns as coherent "chunks." The performance component, on exposure to a chess position, recognizes familiar configurations by sorting them through the previously stored EPAM net, and holds the names of familiar chunks (up to a limit of k) in short-term memory. It then uses this information to reconstruct the position. The limits on its accuracy of reproduction depend on the variety and complexity of the chunks that have previously been familiarized in the learning phase. The performance program also incorporates attentional processes that identify salient pieces in the position, and determine the order in which the board will be scanned.

For the workshop, the program can be manipulated in a variety of ways: the size and contents of the recognition net can be varied through training procedures, or by direct intervention, to simulate different levels of chess skill. Its attentional strategies can be altered. It can be presented with chess positions of a variety of kinds -- actual or random. Its performance can be compared with the performance of human players (data are available) on identical positions.

The chess perception program illustrates the use of a few basic components derived from earlier cognitive simulations (discrimination nets, attention-control strategies) to build a program for a new task environment and to simulate experimentally recorded data there. It allows one to formulate and test theories of attention, and of the interaction of short-term and long-term memory.

For background of the task and program (see de Groot, 1965, Chapter 8; Simon and Barenfeld, 1969; and Chase and Simon, 1971).

Production Systems

There are several options available for representing the information processing system of a human subject when solving a problem. Typically one has written programs in a list processing language (IPL or LISP), sometimes in a regular higher level language (ALGOL, FORTRAN, PS/1) or a list processing augmentation of such a language (SLIP). An important new option is the production system (see Newell and Simon, 1972), which casts the program in the form of a set of situation-action rules with a uniform control structure of evoking the rules.

Putative advantages are (1) the appropriate nature of the factorization of the total performance system into individual locally relevant rules, (2) the uniform nature of the encoding, (3) the close-coupling to models of the immediate memory and immediate processor, (4) the potential for learning and development of total systems by the accretion of rules, and (5), on the data analysis side, the ease of induction from the problem behavior graph as inferred from the subject's protocol. Not all of these advantages have yet been fully realized. However, enough uses have occurred to make production systems a major contender as a theoretical substrate for representing human information processing systems (Newell and Simon, 1972; Waterman, 1970; Klahr and Wallace, 1971).

PSG, a system for the interactive use of production systems will be available for the workshop. It consists of a basic language for expressing production systems and for running them under user control. In practice one runs with a partially defined production systems (being at some intermediate point in an analysis), with the user and the machine working cooperatively. The system takes the next step and displays the result. The user accepts this, modifies it, extends the production system, or substitutes his won response for that of the system (this latter being used to pass over the incompletenesses of the partial system). Thus, a production system is created step by step using the computer to check and verify the model as it develops. This system is described in Newell (1972).

Semantic Understanding Systems

In moving beyond cognitive tasks of a highly restricted nature (e.g., limited task environments such as puzzles), the representation of the general knowledge of a subject is critical. Coupled with this, though in many ways independent, is the question of the internal representation of the knowledge obtained by a subject upon interpreting received utterances in natural language.

The current state of the art in the representation of general knowledge is confused and partial, though highly active. The linguists have, from time to time, focussed on something called deep structure, but this has not matured into a well defined system that can be worked with in a reasonable way. Currently, some linguists are entertaining representation for semantics, within the predicate calculus. Recently in artificial intelligence several systems have come into existence that have enough generality and power to be worth serious investigation. The most well known of these are the systems of Winograd (1972), Woods (1968) and Colby and Smith (1969). There are a number of somewhat earlier and more partial efforts (for reviews and collections, see Simon and Siklossy, 1972; Minsky, 1968; and Simmons, 1970). Besides this, two or three psychologists have begun similar systems, these have not yet come to fruition (see a recent conference, Tulving and Donaldson, 1972).

Such models are not yet serious psychological theories in terms of any detailed body of data. However, within the next few years serious theories will probably emerge, based on these systems (or an amalgams thereof). Thus, these systems represent important techniques for developing adequate cognitive models. Winograd's program, called SHRDLU, will be available at the workshop, as an example of a semantic understanding system.

Concept Learning System

One view of the concept attainment problem suggests that the critical aspects of behavior involve the management of memory in hypothesis testing and generation, and the perceptual analysis of the stimuli. The concept learning system attempts to integrate the empirical tools of the Computer Controlled Psychological Laboratory (a DDP116) with simulation models available on the large scale systems. The system provides a way of testing models of learning strategies against data obtained in the laboratory under several experimental paradigms. In the laboratory, concept attainment studies using the same stimuli can be run under (1) random selection of instances with yes/no responses, (2) simultaneous presentation of two instances, one which is always correct with left/right choices, and (3) single instances selected by the subject with yes/no responses to the instance displayes. The simulation program is designed to run interactively so that subroutines corresponding to subject selection and learning strategies can be modified easily.

This system demonstrates the way in which computer technologies can be useful adjuncts in creating process models. It illustrates the interaction of a data acquisition system with simulation programs. Data from live subjects can be compared with the same kind of data generated by the programs. The rapid turn-around time is an important way to facilitate scientific understanding.

Stimuli for the laboratory experiments are patterned after those used by Neisser and Weene (1962). The discrimination learning paradigm where a positive instance is paired with a negative instance in simultaneous presentation was used by Levine (1966). A general orientation to the process model analysis of concept learning is given in the paper by Gregg and Simon (1962).

Machine-aided Transcription System

The transcription of verbal protocols from audio tapes is a necessary step in using verbal protocols for the analysis of human problem solving behavior. The production of transcriptions raises questions of the amount of work involved, of accuracy and objectivity, of obtaining timing information, and of obtaining prosodic and paralinguistic information. Some of these questions, such as objectivity, can only be successfully approached by fully automatic protocol analysis systems (i.e., by the extension of work such as PAS-I toward accepting audio input). But other aspects can be approached with lesser means.

We have running currently a computer-aided transcription system. The system itself is part of a larger effort on developing speech-understanding systems by R. Reddy of the Computer Science Department. It permits the digitization of the audio tape and the human transcription of segments of the tape under computer control. The system will detect and segment the next non-silent interval of speech, play it (auditorily) as often as desired for transcription, associate the typed-in transcription with it and store it on a computer file. Each segment is timed to the nearest 10 ms. Segments can be broken into subsegments by the user and can be timed and stored as distinct intervals. The ultimate file can be edited, etc., just as can any file on the computer.

The system will be available for the workshop and can be demonstrated and used. It will undoubtedly play a minor role, since new data will not be generated in any volume during the workshop. However, it new data is generated, the system may be used to make it available in reasonably short order for subsequent analyses.

TOP - Zog is a Guide ...

1.+ SYSTEMS - The Systems that are Available.

1.+ PAS-I - Protocol Analysis System #1
1.+ DESCRIPTION - Analyzes Cryptarithmetic Protocols ...
1.+ IMPLEMENTATION - Divided into Three Major Parts
1. PHASE-1 - Snobol Front End ...
2. PHASE-2 - Lisp PBG Building Functions ...
3. PHASE-3 - Lisp PBG Display Functions ...
2.+ LIMITATIONS - PAS-I is not Flexible
1. CRYPTARITHMETIC - Works only on Cryptarithmetic ...
2.+ PROBLEM-SPACE - Subjects limited to an Augmented Problem Space ...
1. PRESENT - Process One Column at a Time
2. EXTENSIONS - Simultaneous Equations ...
3. D+G=R--C+R=D - DONALD+GERALD=ROBERT and CROSS+ROADS=DANGER ...
3. PBG - Problem Behavior Graph ...
2. HOW-TO - Type 'TOP SYSTEMS PAS-I RUN!' ...
3.+ EXAMPLES - Examples of Running PAS-I ...
1. ! EX1 - First 15 segments of Subject 3 on DONALD + GERALD = ROBERT ...
2. ! EX2 - First 50 segments of Subject 3 on D + G = R ...
3. ! EX3 - First 50 segments of Subject 4 on D + G = R ...
4. ! EX4 - First 50 segments of Subject 4 on CROSS + ROADS = DANGER ...
4. ! RUN - Run Node for PAS-I

2.+ PAS-II - Protocol Analysis System #2
1.+ DESCRIPTION - An Overview of PAS-II
1.+ APPLICATION - Help in the Analysis of Protocols
1. ANALYZE-PROTOCOLS - PAS-II is used to Analyze Protocols ...
2. RAW-TEXT - PAS-II Starts with Raw Text ...
2.+ ORGANIZATION - Organization of PAS-II
1. MODULAR - Modular Design of PAS-II ...
2. INTERACTIVE - PAS-II Interacts with the User ...
3. PROBLEM-SPACE-INDEPENDENT - PAS-II is Problem Space Independent ...
3.+ REFERENCES - Backround for Theory Underlying PAS-II
1. PAS-I-REF
2. PRODUCTION-SYSTEMS
3. CRYPTARITHMETIC
4. PROTOCOL-ANALYSIS
2. EXAMPLES
3. HOW-TO - Type 'TOP SYSTEMS PAS-II RUN!' ...
4.+! RUN - Run Node for PAS-II
1. ! SAVE - USE THIS NODE TO SAVE YOUR CORE IMAGE
2. ! RESTORE - USE THIS NODE TO RESTORE YOUR CORE IMAGE
5. ! RUNC - Compiled version of PAS-II. Faster but Untested

3.+ ATS - The semi-automated system for transcribing protocols
1.+ DESCRIPTION - System overview ...
1. ORGANIZATION - How the ATS is organized ...
2.+ EXAMPLE - A quick and easy example of the system in action
1. HOW-TO - How to run the example ...
2. ! RUN - Type ! to start the example. (Do EXAMPLE HOW-TO ? first!!)
3.+ HOW-TO - Procedures, documentation, and running
1.+ PROCEDURES - Where things are and how to get at and use them

```
        1.    TAPE-MAKING - Tips on how to record the protocol ...
        2.    JAWING - Processing your audio tape ...
        3.    SET-UP - Setting up the special hardware ...
        4.+   RUNNING - How to actually run the system
            1. !  MAG-TAPE - What you need to know about mag-tapes ...
            2. !  STARTING - How to start the program going ...
    2. !  DOCUMENTATION - How to obtain manuals for the system ...

4.+ MAPP - MEMORY-AIDED PATTERN PERCEPTION
    1.    DESCRIPTION - MAPP is a SNOBOL program for learning ...
    2.+!  LEARN - READ <LEARN>s DESCRIPTION -- OPTION 5 -- ...
        1.+   EXAMPLES -  OF NETS AND PATTERNS. ...
            1. !  CHESS-NET-FILE - a small net of 28 nodes ...
            2. !  CHESS-PATTERN-FILE - a list of patterns ...
        2.+   BUILDING-NEW-NETS - Information, instructions, and input files ...
            1.    DESCRIPTION - THE NET-FILE REQUESTED AS INPUT ...
            2.+   EXAMPLES - TO SEE NEW-NET FILE FOR CHESS OR TIC-TAC-TOE ...
                1. !  CHESS-NEW-NET
                2. !  TIC-TAC-TOE-NEW-NET
            3.    NEW-NET-PARAMETERS - for Chess and Tic-Tac-Toe ...
            4.    NEW-NET-FILE - TO BUILD A NEW-NET FILE ...
            5. !  PATTERN-INPUT-FILE - Take action to type out example
        3.+   SALIENCY
            1.    SALIENT-SYMBOL - A symbol is salient ...
            2.    DEFINITION - Saliency = ...
            3.    SALIENTPOINTS - SALIENTPOINTS ARE DEFINED AS BEING SALIENT ...
            4.    SALIENCYCRITERION - The default value of saliencycriterion = 1.
        4.+   AVAILABLE-INPUT-FILES - NET, PATTERN-INPUT, AND STIMULUS FILES ...
            1. !  NET-FILES-CHESS - CHESS PATTERN RECOGNITION NETS  ...
            2. !  PATTERN-FILES-CHESS - SETS OF CHARACTERISTIC PATTERNS ...
            3. !  STIMULUS-FILES-CHESS - SETS OF CHESS POSITIONS: ...
            4. !  NET-FILES-T-T-T - NETTTT (114 NODES)
            5. !  PATTERN-FILES-T-T-T - INTTT (43 PATTERNS)
            6. !  STIMULUS-FILES-T-T-T - INT1 (1 STIMULUS)
        5.    DESCRIPTION - MAPP USES A PROGRAM, <LEARN>, TO BUILD  ...
    3.+!  RECALL - READ <RECALL>s DESCRIPTION -- OPTION 1 -- ...
        1.    DESCRIPTION - MAPP USES A PROGRAM, <RECALL>, TO RECOGNIZE ...
        2.+   STIMULUS-FILES
            1.    DESCRIPTION - A stimulus file contains one or more  ...
            2.+   EXAMPLES
                1. !  CHESS - Take action to type out example
                2. !  TIC-TAC-TOE - Take action to type out example
        3. !  SAMPLE-RUN - Take action for sample run of <RECALL> program

5.+ PSG - Production system to model human behavior (version G) ...
    1.+   DESCRIPTION - Gives the scheme for the model itself, not how to use it ...
        1.    PS - Production system ...
        2.+   PD - Production ...
            1.    CND - CONDITION ...
            2.+   ACTION - ACTION ...
                1.    ACT - (ACTION (...)(...)...) executes a sequence of actions
                2.    FAIL - FAIL or (FAIL) terminates the execution of action elements ...
                3.    OPR - (OPR (...) ...) executes the rest of the list as a program ...
                4.    PD - (PD X) executes the production X
                5.    PS - (PS X) executes the production system X repeatedly
```

6. PS.1 - (PS.1 X) Executes the production system X for one cycle. ...
7. == - (X1 == X2) sets variable or class name X1 to have value X2 ...
8. ==> - (X1 X2 ... ==> Y1 Y2 ...) does replacement in first STM elm ...
9. ===> - Replacement on second STM elm, otherwise identical to ==>
10. ====> - Replacement on third STM ELM, otherwise identical to ==>
11. NTC - (NTC X) searches STM for X and brings it to the front ...
12. DATA - (...) puts a copy into STM if not a defined action ...

3. STM - Short Term Memory ...
4. ELMS - Elements -- the encoded chunks of knowledge ...

2.+ TASKS - Task environments (= experimental paradigms)

1.+ ST - Sternberg paradigm: binary classification task ...

1. ! RUN
2. ! DEMO1 - A run-through of a trial of a simple Sternberg experiment

2.+ CA - Counting and adding (including subitizing, estimating) ...

1. ! RUN
2. ! DEMO1 - A run-through of a simple subitizing experiment (after Klahr)

3.+ CY - Cryptarithmetic puzzle ...

1.+ CY2 - PS for S2 (the eye-movement subject in N&S Chapter 7)

1. ! RUN
2. ! DEMO1 - A run-through of first couple of seconds of S2

4.+ NJ - Neal Johnson induced chunking task

1. ! RUN - This runs without calling user, so operates as a demo

5. ! DEMO-TASK - A few simple productions to show their operation ...
6. ! NEW-TASKS - Execute to get new tasks since ZOG went to press

3. ! RUN - Basic system without any specific production system defined ...

4.+ HOW-TO - Basic style for running PSG and creating new production systems

1. BASIC-RUN - To run a prepared production system under TASKS ...
2. OWN-RUN - To run your own production system ...
3. CREATE-PS - To create your own production system ...
4. CREATE-FILE - To create a file for a production system ...

5.+ USER-CTRL - Scheme for the user to control and monitor the system ...

1.+ CTRL-STATES - The control states in which user can gain control

1. START/C - At the beginning, waiting for a production system
2. PS/C - Ready to commence next cycle on PS
3. PD/C - Ready to try a PD (PD not necessarily satisfied)
4. CE/C - Ready to commence match with next condition element
5. MCH/C - Ready to match condition and STM elements
6. CND+/C - Finished matching PD and it is satisfied
7. CND-/C - Finished matching PD and it is not satisfied
8. ACT/C - Ready to apply an action
9. CALL/C - Action is operator -- seeking output from TTY
10. VARSET/C - Ready to set a value of a variable
11. TE/C - Ready to take a TE action
12. BHV/C - Ready to take a BHV action
13. ! NEW-STATE - Additional control states defined after ZOG went to press

2.+ COMMANDS - Available PSG commands

1. <CTRL>Z - ↑Z returns control to the system, executing actions ...
2. ←← - X ←← Y sets the number named X to have the value Y ...
3. ACCEPT/PD - Forces the acceptance of a rejected PD (CND-/C only)
4. BEGIN - Goes back to the beginning of the PS (at PS/C with ↑Z)
5. DEFINE - Use X:(Y1 Y2 Y3 ...) to define a new list with name X ...
6. DO - X DO! executes the action X ...
7. PSG.EDIT - X EDIT! begins editing of X, as does (EDIT X) as action ...
8. FILES - FILE.EXT RDF! reads in a properly prepared file ...
9. HOLD - Stops time from advancing until a RELEASE occurs ...

10. IGNORE - Ignores the pending event defined by the control state ...
11. OFF - X OFF! turns off the process X ...
12. OFF.ALL - X OFF.ALL! turns off all the processes in the list X
13. ON - X ON! turns on the process X ...
14. ON.ALL - X ON.ALL! turns on all the processes in the list X
15. PD - X PD! executes the production X, as does (PD X) as an action ...
16. PR - X PR! prints X (both name and structure), as does (PR X)
17. PRVL - X PRVL! prints the value of variable or class X, as does (PRVL X) ...
18. PS - X PS! executes production system X, cycling repeatedly
19. PS.1 - X PS.1! executes productions system X, as does (PS.1 X) but ...
20. PSG.DEMO - PSG.DEMO! sets the pattern of processes in list DEMO.LIST ON ...
21. PSG.WHIZ - PSG.WHIZ! turns all processes off (with the exception of CALL)
22. RELEASE - RELEASE! permits time to move forward again
23. RESET - RESET! initializes the system ...
24. ! RESTART - RESTART is a ZOG action to save a PSG run image
25. ! SAVE - SAVE! will save the entire run, just as is, provided ...
26. START - X START! does a RESET and then a PS
27. STM-1 - STM-1! deletes one cell off the back end of STM
28. STM+1 - STM+1! adds on cell with NIL to the back end of STM ...
29. USE.DSK - FILE.EXT USE.DSK will send the output to your ...
30. USE.TTY - USE.TTY! uses the TTY to output everything
31. WHERE - WHERE! prints out the current control-state
32. ! NEW.COMMANDS - Additional commands defined after ZOG went to press.
33. OLD.COMMANDS - Commands superceded by this new control scheme: ...

3. SYMBOLS - Miscellaneous symbols ...

6.+ SHRDLU - WINOGRAD'S NATURAL LANGUAGE UNDERSTANDING SYSTEM ...
1.+! RANGER-STATION - ORIENTATION.]--> ! ME FIRST OF ALL.
1. ! MAP - 'MAP!' FOR MAP OF SHRDLU FOREST.
2.+! DESCRIPTION - ONE PAGE GENERAL DESCRIPTION OF WHAT SHRDLU IS ...
1.+ IMPLEMENTATION - MACHINE DETAILS AND VERSION INFO
1. ! DESCRIPTION
2.+ LISTINGS - Program listings of SHRDLU and MICROPLANNER. Beware! ...
1. ! BREAK - Debugging and trace utilities listings
2. !. GLOBAL - Global variables listing
3. ! SYSCOM - Common functions + executive listing
4. ! PROGMR - Support functions for PROGRAMMAR listing.
5. ! GINTER - PROGRAMMAR interpreter listing
6. ! GRAMAR - English grammar listing
7. ! SMSPEC - Semantic Specialists listing
8. ! SMUTIL - Semantic Specialists utility functions listing
9. ! SMASS - Semantic functions for answering listing
10. ! NEWANS - Answer routines listing
11. ! DICTIO - Dictionary listing
12. ! BLOCKS - BLOCKS WORLD (written in MICROPLANNER) listing
13. ! DATA - Initial facts for BLOCKS WORLD listing
14. ! SCENE - Internal data for CRT display listing
15. ! PLNR - Interpreter for MICROPLANNER listing
16. ! THTRAC - Trace package for MICROPLANNER listing
17. ! ALL - all the listings (caution > 200 pages)
3.+ EXAMPLES - EXECUTE EXAMPLES BELOW TO SEE HOW SYS WORKS
1.+ PARSER-TRACE - Parser traced for the sentence ...
1. ! DESCRIPTION -]--> HOW THE PARSER WORKS ...
2. ! RUN - TO RUN ON TTY
3. ! PRINT - PRINTS EXAMPLE ON PRINTER

2.+ SEMANTIC-STRUCTURES-TRACE - Oss,RSS,TSS traced for the sentence ...
1. ! DESCRIPTION -]--> HOW THE SEMANTIC SPECIALISTS ARE CALLED ...
2. ! RUN - TO RUN ON TTY
3. ! PRINT - PRINTS EXAMPLE ON PRINTER

3.+ INFERENCE-TRACE - MICROPLANNER traced for the sentence ...
1. ! DESCRIPTION
2. ! RUN - TO RUN ON TTY
3. ! PRINT - PRINTR EXAMPLE ON PRINTER

4.+ WALLPAPER - Above three traces combined
1. ! DESCRIPTION
2. ! RUN - TO RUN ON TTY
3. ! PRINT - TO PRINT ON PRINTER

5. ! DESCRIPTION

6.+ BUILD
1. ! DESCRIPTION
2. ! RUN - TO RUN ON TTY
3. ! PRINT - TO PRINT ON PRINTER

4. ! RUN - ! ME TO RUN CLEAN UNTRACED SHRDLU

5. ! HOW-TO - ! ME FOR HOW TO RUN SHRDLU

7.+ CLS - Concept Learning System

1. GENERAL-DESCRIPTION - This system is run ...

2.+ SYSTEM-STRUCTURE - The system employs two computers, ...
1. DDP-116 - This real-time computer controls ...
2. IBM-360/67 - An interactive message handler, ...
3. INTERFACE - The two computers communicate ...

3.+ EXPERIMENTAL-PARADIGMS - The stimuli are displayed ...
1. ED-JOHNSON - the display is a string ...
2. NEISSER-WEENE - The display is a string ...
3. BOURNE - The display is a sentence ...
4. DO-IT-YOURSELF - The experimenter-user will be prompted ...
5.+ VARIABLES - Under the Control of the User ...
1. DIMENSIONS
2. RANGE - Number of Values in each Dimension==> 2-8
3. SYMBOLS
4. POSITIONS
5. RULES
6. FIXED-VS-PERMUTED - dimensions in the displays.
7. DISTRIBUTION - proportion of positive instances may be varied.
8. INSTRUCTIONS - files can be created and displayed.

4.+ SIMULATION-PROGRAMS - in the form of production systems. ...
1. MEMORY-STRATEGIES - Three hypothesis selection strategies ...
2. OTHER-PARAMETERS - The size of short-term memory, ...

5.+ RUNNING-EXPERIMENTS
1. LOADING - The CCPL staff must load ...
2. AVAILABILITY - The system will be available ...
3. START - On the DDP-116 console, type: ...

8.+ SIDESHOW - MISCELLANEOUS GAMES AND DEMONSTRATIONS.
1. ! ELIZA - PSYCHOLOGICAL COUNSELLING ...
2. ! LESCAL - PERSONALIZED CALENDERS ...
3. ! LIFE - STYLIZED EVOLUTION ...
4. ! 3-D-TTT - THREE-D TIC-TAC-TOE ...
5. ! HANGMAN - RELIVE YOUR CHILDHOOD DAYS ...
6. ! DIGITS - HOW RANDOM CAN YOU BE? ...

7. ! AIQUIZ - Artificial Intelligence Quiz ...

9.+ CHESS - TWO CHESS-PLAYING PROGRAMS ...
1.+! TECH - THE TECHNOLOGY CHESS PROGRAM ...
1. DESCRIPTION - SHORT DESCRIPTION ...
2. BASIC-INSTRUCTIONS - MINIMAL INSTRUCTIONS TO RUN TECH ...
3. ADVANCED-INSTRUCTIONS - FOR THE MORE ADVENTUROUS ...
4. EXAMPLES - TO GET YOU STARTED ...
5. ERRORS - WHAT TO DO ...
2.+! GREENBLATT - THE GREENBLATT CHESS PROGRAM ...
1. DESCRIPTION - SHORT DESCRIPTION ...
2. BASIC-INSTRUCIONS - MINIMAL INSTRUCTIONS TO RUN THE GREENBLATT PROGRAM ...
3. ADVANCED-INSTRUCTIONS - FOR THE MORE ADVENTUROUS ...
4. EXAMPLES - TO GET YOU STARTED ...
5. ERRORS - WHAT TO DO ...

2.+ ZOG - How to Use ZOG ...

1.+ NOTATION - Basic Commands for Moving in the Tree ...
1. QM-COMMAND - ? ==> Give Options ...
2. DQM-COMMAND - ?? ==> Identify this Node ...
3. EXCL-COMMAND - ! ==> Execute Proper Action ...
4. UP-COMMAND - ↑ or up ==> Go Up one Level in the ZOG Tree
5. TOP-COMMAND - ↑↑ or TOP ==> Go to the TOP of the ZOG Tree
6. <NUMBER> - <number> ==> Go to Option <number> of this Node ...
7. HELP-COMMAND - HELP ==> Type Options for NOTATION ...
8. NOVICE-EXPERT - Modes to Control Amounts of Zog Output ...
9. DEMO-NODEMO - Modes for giving Demonstrations ...
10. TQM-COMMAND - ??? ==> ?? and then ?
11. OPTION-LIST - Meaning of '...', '+', and '!' ...
12. EXCL-QM-COMMAND - !? ==> Print Proper Action ...

2.+ TTY-EDIT - Limited editing of messages to or from TTY
1. <CONTROL>O - ↑O ==> Suppress teletype printout until an input is requested.
2. <RUBOUT - RUBOUT or DEL ==> Delete the last character typed.
3. <CONTROL>U - ↑U ==> Delete the entire input line now being typed.

3.+ CONTROL - Function of Zog Control Characters. ...
1. <CONTROL>A - ↑A ==> Escape to Zog // Continue Sub-job ...
2. <CONTROL>B - ↑B ==> Input from TTY // Input from Command File ...
3. <CONTROL>F - ↑F ==> Suppress Output // Emit Output ...
4. <CONTROL>G - ↑G ==> Quote Character (Not a Switch) ...
5. <CONTROL>N - ↑N ==> Input from Proper Action // Input from User ...
6. <CONTROL>R - ↑R (or ↑C↑C REENTER) ==> Panic Button (Not a Switch) ...
7. <CONTROL>T - Force Zog to Send TTY Input to Sub-job ...
8. <CONTROL>V - ↑V ==> Echo Commands // Suppress Echo of Commands ...
9. <CONTROL>W - ↑W ==> Comment until next <CR> or ↑W ...
10. <CONTROL>Y - ↑Y ==> Ready the Sub-job (Not a Switch) ...
11. <AT-SIGN> - @ ==> Execute a Command File ...
12.+ EXAMPLES - Sample Command Files
1. ! EX1 - EX1.CMD[KARLTON] ==> Types EX2.CMD[ROBERTSON]
2. ! EX2 - Demonstrates the use of all Control Characters ...
13. OTHERS - Other Control Characters ...

2. D - D<N1>or D<N1>:<N2> ==> Delete Line(s) ...
3. E - E ==> Save the Current File and Exit the Editor
4. P - P or P<n1> or P<N1>:<N2> ==> Print line(s) on TTY ...
5. L - L or L<n1> or L<N1>:<N2> ==> Print line(s) on Line Printer ...
6. W - W ==> Save the Current File (World) ...
7. F - F<string><altmode> ==> Find <string> ...
8. B - B ==> Go to Beginning of File

4.+ EXAMPLES - Create and Edit a File
1. ! CREATING - Creates a 3 Line File
2. ! EDITING - Edits File Created by CREATING

2. ! TYPE - List a File on the User's TTY ...

3. ! PRINT - List a File on the Line Printer ...

4. ! MAIL - Send a Message ...

5. ! COPY - Copy a File ...

6. ! DIRECTORY - List of Existing File Names ...

7. ! DELETE - Delete Files ...

8. ! RENAME - Rename a File ...

9. ! OPERATOR - Ask the Operator to do Something ...

10. ! STATUS - Gives the Status of the PDP-10

11. FILE-STORAGE - [<name>] References <name>'s File Storage Area ...

12. ! GRIPE - Suggestions, Comments, and Complaints ...

13. MAIL-PENDING - You have been Sent Mail ...

5. PEOPLE - List of Users and Builders ...

6.+ LANGUAGES - Languages Used by the Systems

1.+ L*(G) - L*(G) IS A KERNEL SYSTEM-BUILDING SYSTEM ...
1.+ DOCUMENTATION - NO WRITTEN DOCUMENTATION EXISTS FOR L*(G) ...
1. ! LISTINGS - THE PROPER ACTION HERE GIVES YOU L*(G) LISTINGS ...
2. GENERAL - A GOOD GENERAL INTRODUCTION TO L* SYSTEMS ...
2.+ PRINCIPLES - THERE ARE SEVERAL DESIGN PRINCIPLES ...
1. P1 - SMALL INITIAL SIZE (SIMPLICITY) -- ...
2. P2 - SELF-SUFFICIENCY FOR INTERNAL GROWTH -- ...
3. P3 - TOTAL ACCESSIBILITY (NO DESIGNER'S PREROGATIVE) -- ...
4. P4 - MULTIPLE USE OF STRUCTURE -- ...
5. P5 - INTEGRATED PROGRAMMING ENVIRONMENT -- ...
6. P6 - PERSONALIZATION -- ...
7. P7 - DESIGN ITERATION -- ...
8. P8 - SELECTIVE EFFICIENCY -- ...
3. USAGE - USE OF L*(G) AT CMU IS NOT WIDESPREAD; HOWEVER, ...

```
    4. !  SCRIPT - AN INTERACTIVE SCRIPT FOR TEACHING L*L, ...
    5. !  DEMONSTRATION - AN L*(G) DEMONSTRATION SESSION IN 6 PARTS ...
    6. !  RUN - THIS NODE GETS YOU INTO L*(G)A, ...

2. !  SAIL - Sail is an ALGOL Dialect ...

3.+   LISP
    1.+   STANDARD - LISP INTERPRETER
        1.    DESCRIPTION - TO INITIALIZE, USE ↑N OR ↑T TO ENTER YOUR RESPONSE ...
        2. !  RUN - RUN MODE FOR LISP
    2.+   AUGMENTED - LISP INTERPRETER IN 16K, DECIMAL, WITH 'TEST' FUNCTION
        1.    DESCRIPTION - THIS IS STANDARD LISP PLUS THE FUNCTION 'TEST' ...
        2. !  RUNA - RUN MODE FOR LISP WITH 'TEST' FUNCTION

4.+!  SNOBOL - (STRING-ORIENTED SYMBOLIC LANGUAGE) ...
    1. !  MANUALS - TWO MANUALS ARE AVAILABLE. ...
    2. !  EXAMPLE1 - A VERY SIMPLE SNOBOL PROGRAM ...
    3. !  EXAMPLE2 - A MORE COMPLEX SNOBOL PROGRAM ...
    4.+   SNIP - SNOBOL INTERACTIVE PACKAGE ...
        1. !  MANUAL - A 9 PAGE SNIP MANUAL IS AVAILABLE ...
        2. !  RUN - TO RUN THE SNOBOL INTERACTIVE PACKAGE ...

5.+   MACLISP - LISP AT MIT'S AI LABORATORY ...
    1. !  DOC - !  TO PRINT SOME DOCUMENTATION ON PRINTER ...
    2. !  RUN

6.    APCOL

7.+   MICROPLANNER - PROGRAMMING LANGUAGE.
    1. !  RUN
    2.+!  DESCRIPTION
        1. !  MANUAL - !  TO GET MANUAL ON PRINTER ...
    3.+   EXAMPLES
        1.+!  PCTP - PREDICATE CALCULUS THEOREM PROVER ...
            1.+!  DESCRIPTION
                1. !  MANUAL
            2. !  RUN

8.+   CONNIVER - PLANNING LANGUAGE. ALTERNATIVE FORMULATION OF PLANNER ...
    1.+   EXAMPLES - 'FRINGE' EXAMPLE. VISITS LEAVES OF A TREE. ...
        1. !  LISTINGS - LISTS 'FRINGE' EXAMPLE ON PRINTER
        2.    HOW-TO - HOW TO RUN FRINGE EXAMPLE ...
    2. !  RUN
```

User types "ZOG" to get started.
all input by the user is underlined.

.ZOG ← system asks who I am.
NAME: KARLTON
TYPE TOP?? FOR INTRO, HELP FOR HELP.
(>) ← This is the prompt ZOG gives when it is waiting for input.
Most of the subsystems also have a 'prompt' character.
Taking the advice of the first line ZOG gives us, let's type "TOP??"

>TOP??
- TOP - +
Zog is a Guide
You are at the top of a tree (the ZOG tree). A system (ZOG) is used to enable you to move around in this tree and more importantly to let you use and learn about the systems that ZOG can introduce to you. Type HELP to learn how to use Zog and move around in the Zog tree.
>HELP and then "HELP" to find out how to move in the ZOG tree.
Basic Commands for Moving in the Tree
To move down into the tree, type the name of the node you want to be at. e.g. to get more help type 'TOP ZOG NOTATION'. Other commands are explained by the options under this node.

1. QM-COMMAND - ? ==> Give Options ...
2. DQM-COMMAND - ?? ==> Identify this Node ...
3. EXCL-COMMAND - ! ==> Execute Proper Action ...
4. UP-COMMAND - ↑ or up ==> Go Up one Level in the ZOG Tree
5. TOP-COMMAND - ↑↑ or TOP ==> Go to the TOP of the ZOG Tree
6. <NUMBER> - <number> ==> Go to Option <number> of this Node ...
7. HELP-COMMAND - HELP ==> Type Options for NOTATION ...
8. NOVICE-EXPERT - Modes to Control Amounts of Zog Output ...
9. DEMO-NODEMO - Modes for giving Demonstrations ...
10. TQM-COMMAND - ??? ==> ?? and then ?
11. OPTION-LIST - Meaning of '...', '+', and '!' ...
12. EXCL-QM-COMMAND - !? ==> Print Proper Action ...

we are still at the TOP of the tree. "?" should let us see what options are available here.

>?
- TOP - +
1.+ SYSTEMS - The Systems that are Available.
2.+ ZOG - How to Use ZOG ...
3.+ UTILITIES - Utility Programs: Edit, Copy, Print, etc.
4. PEOPLE - List of Users and Builders ...
5.+ LANGUAGES - Languages Used by the Systems
↑ indicates that options exist under LANGUAGES.
Let's see what SYSTEMS are available.

>SYSTEMS?
- TOP SYSTEMS - +
1.+ PAS-I - Protocol Analysis system #1
2.+ PAS-II - Protocol Analysis System #2
3.+ ATS - The semi-automated system for transcribing protocols
4.+ MAPP - MEMORY-AIDED PATTERN PERCEPTION
5.+ PSG - Production system to model human behavior (version G) ...
6.+ SHRDLU - WINOGRAD'S NATURAL LANGUAGE UNDERSTANDING SYSTEM ...

7.+ CLS - Concept Learning System
8.+ SIDESHOW - MISCELLANEOUS GAMES AND DEMONSTRATIONS.
9.+ CHESS - TWO CHESS-PLAYING PROGRAMS ... → 3 dots indicate more text is available

↑ go back up one level (to TOP, we are at SYSTEMS)
3 go down to option 3 under TOP
?? list the options at that node.

>13??
- TOP UTILITIES - +
1.+! EDIT - Edit or Create a File
2. ! TYPE - List a File on the User's TTY ...
3. ! PRINT - List a File on the Line Printer ...
4. ! MAIL - Send a Message ...
5. ! COPY - Copy a File ...
6. ! DIRECTORY - List of Existing File Names ...
7. ! DELETE - Delete Files ...
8. ! RENAME - Rename a File ...
9. ! OPERATOR - Ask the Operator to do Something ...
10. ! STATUS - Gives the Status of the PDP-10
11. FILE-STORAGE - [<name>] References <name>'s File Storage Area ...
12. ! GRIPE - Suggestions, Comments, and Complaints ...
13. MAIL-PENDING - You have been Sent Mail ...

→ indicates that this node can be 'executed', i.e., ! at this node will initiate the 'proper action' there.

① ② ③ ④
>↑↑ ZOG ?? ? → ① go to TOP of tree ② pick node ZOG ③ type all the text available at that node ④ list its options.
- TOP ZOG - +
How to Use ZOG
The nodes under this node of the tree give information about different aspects of ZOG. To find out what options are available type ?. To get information from one of those options, type its name or number and follow that by ??. Good luck.
- TOP ZOG - +
1.+ NOTATION - Basic Commands for Moving in the Tree ...
2.+ TTY-EDIT - Limited editing of messages to or from TTY
3.+ CONTROL - Function of Zog Control Characters. ... ← this node explains about the control characters that will be used later.
4. RUNNING - What happens when you RUN! ...
5. EXITING-ZOG - In order to leave Zog type ZOGOUT. ...
6.+ SUGGESTIONS - Things to try and do.

Let's try and actually fire off one of the proper actions, e.g., the first example of PAS-I. The name of the node we want is EX1, but I'm going to misspell to see what action the system (ZOG) takes.

>↑PAS-I EX-1!
?INVALID NAME IN "EX-1!" Sure enough, it told me the name does not exist. Note the rest of the line is ignored, but the action of the first part seem taken.

→ Verification: like we are at PAS-I and EXAMPLES are available.
>? Let's see if we can find the node we want?
- TOP SYSTEMS PAS-I - +
1.+ DESCRIPTION - Analyzes Cryptarithmetic Protocols ...
2. HOW-TO - Type 'TOP SYSTEMS PAS-I RUN!' ...
3.+ EXAMPLES - Examples of Running PAS-I The ...
4. ! RUN - Run Node for PAS-I

>EXAMPLES? → go down to EXAMPLES and list its options.
- TOP SYSTEMS PAS-I EXAMPLES - +
1. ! EX1 - First 15 segments of Subject 3 on DONALD + GERALD = ROBERT ...
2. ! EX2 - First 50 segments of Subject 3 on D + G = R ...
3. ! EX3 - First 50 segments of Subject 4 on D + G = R ...
4. ! EX4 - First 50 segments of Subject 4 on CROSS + ROADS = DANGER ...
>1?? - picks the first option and print what information is there.
- TOP SYSTEMS PAS-I EXAMPLES EX1 - !
First 15 segments of Subject 3 on DONALD + GERALD = ROBERT
Elementary Example: Shows no commands.
We are at the correct node. We want to fire off this node. Simple enough all we need to do is type "!"!
>!
Copying Example Segmented Text File
Running PAS-I Snobol Phase

PAS-I is running.

B1. EACH LETTER HAS ONE AND ONLY ONE NUMERICAL VALUE --
(?)
B2. EXP
(EXP)
B3. THERE ARE TEN DIFFERENT LETTERS
(?)
B4. AND EACH OF THEM HAS ONE NUMERICAL VALUE .
(AND)

<B1-4>. ((?))

B5. THEREFORE , I CAN , LOOKING AT THE TWO D 'S --
(THEREFORE) (NUM D 2)

<B5>. ((THEREFORE) (NUM D 2))

B6. EACH D IS 5 .
(EQ D 5)
B7. THEREFORE , T IS ZERO .
(THEREFORE) (EQ T 0)

<B6-7>. ((BECAUSEOF ((EQ D 5)) ((EQ T 0))))

B8. SO I THINK I 'LL START BY WRITING THAT PROBLEM HERE .
(THEREFORE)

<B8>. ((?))
We've seen all this before, so <CONTROL>R can be used to cut off this demo.
↑R

REENTERING ZOG AT TOP ← NOTE: we reenter at the TOP.
>TOP CHESS?? ? - Let's see what the system knows about chess
?AMBIGUOUS:
SYSTEMS MAPP RECALL STIMULUS-FILES EXAMPLES CHESS } trace each occurrence
SYSTEMS CHESS } of CHESS.

If a node with a name you wish to descend to occurs but once in the subtree below your current position, or is a direct son, then you descend to that node. Multiple occurrences of the name in the subtree lead to the "AMBIGUOUS" message. Let's pick one of the paths.

>SYSTEMS CHESS?
- TOP SYSTEMS CHESS - +
1.+! TECH - THE TECHNOLOGY CHESS PROGRAM ...
2.+! GREENBLATT - THE GREENBLATT CHESS PROGRAM ...
>1!
TECH: 1 SEP 1972

1. ↑A → escape to ZOG so we can rummage around in the tree and find out what to do.

NOTE: back in ZOG.

>??
- TOP SYSTEMS CHESS TECH - +!
THE TECHNOLOGY CHESS PROGRAM
TECH WAS DEVELOPED AT C-MU BY JIM GILLOGLY IN 1970-71 AND IS STILL BEING MODIFIED. IT HAS A CLASS D RATING (1286) IN THE U. S. CHESS FEDERATION, AND WON THE SECOND PLACE TROPHY AT THE 2ND ANNUAL COMPUTER CHESS CHAMPIONSHIPS (CHICAGO 1971), PLACING BEHIND CHESS 3.5 OF NORTHWESTERN UNIVERSITY.

this is the text alluded to by the comment about " " ... on the second page

>?
- TOP SYSTEMS CHESS TECH - +!
1. DESCRIPTION - SHORT DESCRIPTION ...
2. BASIC-INSTRUCTIONS - MINIMAL INSTRUCTIONS TO RUN TECH ...
3. ADVANCED-INSTRUCTIONS - FOR THE MORE ADVENTUROUS ...
4. EXAMPLES - TO GET YOU STARTED ...
5. ERRORS - WHAT TO DO ...
>2? BASIC-INSTRUCTIONS - is what we need.
- TOP SYSTEMS CHESS TECH BASIC-INSTRUCTIONS -
NO OPTIONS ah, yes, we need the text here, not the options.
>??
- TOP SYSTEMS CHESS TECH BASIC-INSTRUCTIONS -
MINIMAL INSTRUCTIONS TO RUN TECH
WHEN TECH GIVES THE MOVE NUMBER (WHITE) OR SPACES (BLACK), YOU CAN INPUT YOUR MOVE IN STANDARD ENGLISH NOTATION DISAMBIGUATED BY SLASHES WHENEVER NECESSARY (E.G., P-K4 OR N/Q4XP), OR YOU CAN GIVE ONE OF THE FOLLOWING COMMANDS:

PW<CR>	THE PROGRAM IS TO PLAY WHITE.
PB<CR>	THE PROGRAM IS TO PLAY BLACK.
B<CR>	PRINT THE CURRENT BOARD.
SC<CR>	PRINT THE SCORE. TECH WILL PROMPT 'SCORE' YOU ANSWER 'Y'.
DR<CR>	ASKS IF IT WANTS A DRAW.
↑N<CR>	RETURN TO ZOG.

PROMOTIONS ARE INDICATED WITH '=', E.G. P-Q8=Q. EN PASSANT CAPTURES MUST BE FOLLOWED BY THE SINGLE LETTER 'E', E.G. PXNPE FOR PXNP EN PASSANT.

>↑A back to the program

PW

TECH	PK01
1. P-K4	P-K4
2. N-KB3	N-QB3
3. P-Q4	B

```
*R -*B*Q*K*B*N*R
*P*P*P*P -*P*P*P
 - -*N - - - - -
 - - - -*P - - -
 - - - P P - - -
 - - - - - N - -
 P P P - - P P P
 R N B Q K B - R
```

3. ... PXN

ILLEGAL

3. ... 1A now, what do I do?

>UP ERRORS??

- TOP SYSTEMS CHESS TECH ERRORS -

WHAT TO DO

IF YOUR MOVE IS SYNTACTICALLY IN ERROR, AMBIGUOUS OR ILLEGAL, TECH WILL COMMENT APPROPRIATELY; REPEAT YOUR MOVE, THIS TIME CORRECTLY. IF YOU DETECT AN ERROR BEFORE YOU <CR>, USE <RUBOUT> OR <DELETE> TO BACKSPACE.

>1A — back to the program.

PXP

4. NXP	NXN
5. QXN	N-B3
6. N-QB3	B

```
*R -*B*Q*K*B -*R
*P*P*P*P -*P*P*P
 - - - - -*N - -
 - - - - - - - -
 - - - Q P - - -
 - - N - - - - -
 P P P - - P P P
 R - B - K B - R
```

6. ... 1N

EXIT

>ZOGOUT

LEAVING ZOG

. — this is the PDP-10 monitor's prompt. We are out of ZOG.