

AD-769 973

COMPUTER SCIENCE RESEARCH REVIEW  
1972-1973

Barbara Anderson

Carnegie-Mellon University

Prepared for:

Air Force Office of Scientific Research  
Advanced Research Projects Agency  
National Science Foundation

September 1973

DISTRIBUTED BY:

**NTIS**

National Technical Information Service  
U. S. DEPARTMENT OF COMMERCE  
5285 Port Royal Road, Springfield Va. 22151

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD 76 9973

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER <b>AFOSR - TR - 73 - 2001</b>	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle)  COMPUTER SCIENCE RESEARCH REVIEW 1972-73	5. TYPE OF REPORT & PERIOD COVERED  Interim	
	6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s)	8. CONTRACT OR GRANT NUMBER(s)  F44620-70-C-0107	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Department of Computer Science Pittsburgh, Pennsylvania 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS  61101D AO 827	
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209	12. REPORT DATE September 1973	
	13. NUMBER OF PAGES 64 (63)	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Air Force Office of Scientific Research (NM) 1400 Wilson Blvd Arlington, Virginia 22209	15. SECURITY CLASS. (of this report) UNCLASSIFIED	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)  This report is a review and summary of the research conducted by the Department of Computer Science, Carnegie-Mellon University, during the academic year 1972-1973.  Reproduced by NATIONAL TECHNICAL INFORMATION SERVICE U S Department of Commerce Springfield VA 22151		

DD FORM 1473  
1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Computer Science  
Research Review  
1972-73

AD 769973

ACCESSION for	
HTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION .....	
BY .....	
DISTRIBUTION/AVAILABILITY STATEMENT .....	
Dist.	Avail. Statement
<b>A</b>	

**AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)**  
**NOTICE OF TRANSMITTAL TO DDC**  
 This technical report has been reviewed and is approved for public release IAW AFR 190-12 (7b).  
 Distribution is unlimited.

**D. W. TAYLOR**  
**Technical Information Officer**

EQSR 16-78-2001

2

Approved for public release;  
distribution unlimited.

Computer Science Research Review  
1972-73

An Annual Report  
published by the  
Department of Computer Science  
Carnegie-Mellon University  
Pittsburgh, Pennsylvania

DDC  
RECEIVED  
NOV 29 1973  
REGISTERED  
E

Editing by Barbara Anderson  
Book Design by Darlene Covaleski

The work reported here was largely supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (Contract number F44620-70-C-0107) and is monitored by the Air Force Office of Scientific Research. This work was also supported in part by the National Science Foundation (Contract numbers GJ 32111, GJ 32758x, GJ 23457x, GJ 32259, GJ 30127, GJ 32784), the Office of Naval Research (Contract numbers N00014-67-A-0314-0010, N00014-67-A-0314-0018), and numerous private grants for which we are deeply grateful. For a complete listing of these grants and contracts, see page 61 of this report.

Pittsburgh, Pennsylvania  
September, 1973

Illustrations by the Engineering Lab

- 4 -

## Contents

<b>Annual Review Introduction</b> Joseph F. Traub	5
<b>Design Augmentation</b> Charles M. Eastman	7
<b>On the Scheduling Aspects of Timing Concurrent Processes</b> A. Nico Habermann	17
<b>Some Practical Uses for Analytical Models in the Study of Computing Systems</b> John W. McCredie	25
<b>Lessons from Perception for Chess-Playing Programs</b> Herbert A. Simon	35
<b>Faculty</b>	43
<b>Departmental Staff</b>	46
<b>Graduate Students</b>	47
<b>Publications</b>	51
<b>Research Reports</b>	56
<b>Colloquia</b>	59
<b>Gifts, Grants and Contracts</b>	61
<b>Ph.D. Dissertations</b>	62

## Annual Review Introduction

We continue to engage in large multi-person research projects and in individual research. Two of our largest projects are the design of hardware and software for our multi-mini-processor computer system (C.mmp) and the building of the Hearsay speech understanding system.

By June 1973 C.mmp had grown to three processors and four memory ports. By the end of summer the 16 x 16 switch will be ready, giving us the potential of a 16-processor configuration. The kernel of the operating system is running on the prototype and is being driven by test programs. A piece of the speech system is now up on the C.mmp.

The Hearsay system is now operational and was demonstrated live at several workshops. This system demonstrates the use of context, syntax, and semantics in a speech recognition task and represents a significant milestone in the cooperative speech understanding research effort that is presently underway at several universities and research institutions.

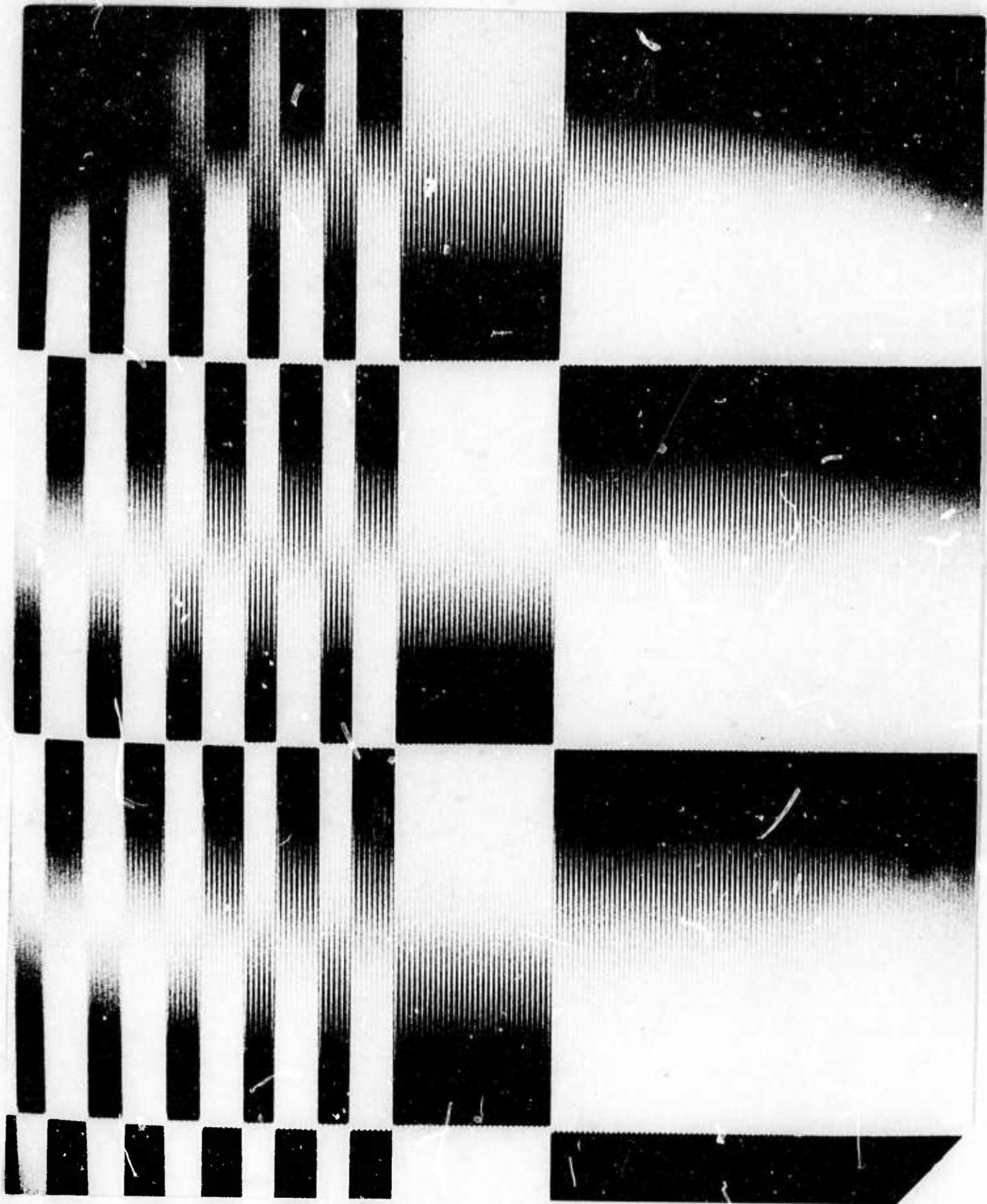
We view workshops and symposia as one of our major links for research communication with the rest of the world.

A nine day Workshop (joint with Psychology) explored New Techniques in Cognitive Research. The "new techniques" are programming systems that embody within themselves significant psychological theory which one explores and uses to construct new theory interactively. The nine days were spent on-line and a major purpose of the Workshop was to assess the advantages of this sort of scientific communication rather than the usual talk-intensive workshops. The Workshop was a success and we are engaging in seven smaller but similar workshops this summer.

Other workshops dealt with Architecture and Application of Digital Modules and with Segmentation and Classification of Connected Speech.

A group of IBM scientists and managers visited for a two day CMU-IBM Minisymposium on current computer science research at the two institutions. A Symposium on Complexity of Sequential and Parallel Numerical Algorithms provided a forum for the presentation and discussion of recent research results and surveys on topics such as the interdependence of machine organization and algorithms, and algebraic and analytic computational complexity.

J.F.T.



## Design Augmentation

Charles M. Eastman

In 1963, Steven Coons described the potential of the computer in design as follows:

"We outlined . . . a system that would in effect join man and machine in an intimate cooperative complex, a combination that would use the creative and imaginative powers of the man and the analytical and computational powers of the machine each with the greatest possible economy and efficiency. We envisioned even then the designer seated at a console, drawing a sketch of his proposed device on the screen of an oscilloscope tube with a "light pen", modifying his sketch at will, and commanding the computer slave to refine the sketch into a perfect drawing, to perform various numerical analyses having to do with structural strength, clearances of adjacent parts, and other analyses as well . . ." [2].

It is now ten years later; a wide variety of graphic terminals have become available, yet this conception has been realized in only very limited areas. One can attribute a variety of causes to the failure of Coons' image being realized. Among them must be included:

- a. *poor understanding of most design tasks.* In most areas we still lack a clear picture of the information typically available for use in decision-making, the sequences of decisions required due to external requirements, and the mode of problem solving normally used by designers in a particular field.
- b. *restricted system designs.* Little effort has been devoted to the matching of design tasks to CAD system capabilities. The generality of data structures, operations, and forms of analysis has been limited, at least in part, to technical problems of software organization.

c. *an arbitrarily restricted view of the contribution of the computer to design.* Most efforts in CAD have begun by partitioning design into two sets of tasks, those algorithmically defined subtasks, and all others, as Coons has done above. The machine undertakes the first set while the human designer "fills in" to complete all the others. This partitioning is often an inefficient use of both man and machine and inevitably leads to questionable systems organization assumptions.

Since about 1967, a group of faculty and students at CMU has been addressing the above technical issues associated with computer-aided design, particularly as applied to architecture, civil engineering, and industrial equipment design. In this paper, I review these efforts and outline what I believe to be their contribution to date.

### Task Analysis

Design is often considered an art, particularly if it is oriented towards buildings or other public products. Case studies and more rigorous analyses of the process of design, as carried out traditionally, have only begun to clarify for computer system designers the tasks involved in design and their possible organizations.

Design is a process of long duration; a complex design may take several years to complete. The range of activity involved in such an extensive process requires a carefully structured analysis. Analyses of the design process have been attempted at three levels of detail. The first and most general level might be called the *molar* level. Its duration is the total length of design (months or years), and the decisions it examines are usually collected through a case study (*i.e.* recall) or diary format; the actions characterized are most often those of a group. The kinds of information normally collected at the molar level include the general sequence in which major design decisions are made, what those decisions are, the sequence in which important information is received, and from whom.

It is also possible to analyze subsets of design decisions. Design problems can be defined that are the appropriate province of a single decision-maker.

In this research context, the information brought to bear, the external representation of information, and other processing of information by an individual designer can be carefully monitored. This level of detail might be called the *molecular* level of design analysis. Usually this level of analysis is characterized as design "problem-solving" and is amenable to the techniques of analysis Newell has developed for problem solving research [15]. Design actions at the molecular level assume as primitives both standard methods of analysis, where these are formalized, and cognitive processes, where no formalization exists. Each design problem analyzed at this level also is assumed to be a single element in the analysis of case studies. Thus this intermediate level of analysis relates primitive perceptual and cognitive processes to the global organization of design.

The lowest level at which design has been studied may be called the *atomic* level (in accordance with our physical science analogy). At this level, the researcher is concerned with the primitive capabilities required to analyze and synthesize physical systems. Many of the analytic primitives have received much attention, *e.g.*, procedures for predicting the behavior of a structure to static loads. Others are of a psychological nature and have only begun to be explored. These include:

- a. the structure of human memory about the physical and visual world and strategies for accessing information within this structure;
- b. matching a verbal description of a condition with a graphic pattern corresponding to that condition, or vice versa, deriving a verbal description of a graphic pattern — in general, creating a description or deriving a correspondence in one kind of language from another language;
- c. testing visually if an object fits within a given space.

The study of primitive operations normally involves laboratory experimentation.

Several studies here have contributed to the body of knowledge regarding the process of design. Eastman and Yessios have undertaken type two, or molecular level studies [3,4,5,18]. Ballay and Moran have studied type three processes [1,13]. Studies undertaken elsewhere have focused on molar studies. These studies have allowed us to elaborate our understanding of design and to determine the context for future studies.

Rather than relate the results of particular studies, I shall attempt to generalize from them. Of necessity, these generalizations are interpretive, but suggest important criteria for the design of CAD systems. In particular, no single sequential structure is likely to be adequate for use by different designers in different contexts. While a common set of operations may eventually evolve, the unique information gained from the application of each will lead to a different, possibly unique sequence [4].

Also, design problems are usually both ill-structured and ill-defined. That is, they are not easily characterized within any one representation and they initially are only partially defined. The designer is responsible for both structuring the problem and completing its definition. He normally does so today through an iterative process of partial definition and resolution. Solutions are used to prompt his experience for the purpose of elaborating the problem definition. This method of problem solving benefits from displays of partial solutions in multiple representations [3].

The strategies used by designers correspond closely to the general problem solving processes called heuristic search. Generate-and-test, means-ends analysis, and planning all can be observed in design protocols collected at the molecular level; often they are intermixed. These results suggest that a CAD system should incorporate capabilities for tests, means-ends tables, and the mapping capabilities needed for planning [3].

As recognized by others, intuitive design is hierarchical and sequential; the subset of variables having global effects are abstracted for early decisions, while others of only local significance are generally resolved later. Decision sequences are also influenced by the external constraints upon variables posed by a particular context. Because each design problem comes with a unique set of constraints, different variables are initially bounded in different problems. The sequence of assignments to variables is partially determined by their binding; those tightly bounded are assigned early (before they become overconstrained). Thus the intuitive sequence of decision-making used by humans in each design problem may vary [3,13].

Many design problems are underconstrained and have no precise objective function. Without greater information regarding goals, a great range of solutions is possible. Moreover, the search of the problem space for a specific solution is potentially inefficient, due to the lack of constraints for partitioning the domain down to manageable size. In this context, designers often add constraints to simplify their own problem solving. These constraints reflect subjective concerns and are a major component in the art of design. Traditionally, the adding of constraints has been an important prerogative of designers.

The mental representations of form and the operations on them used by individuals correspond closely to their perceptual and manipulative experience. Sculptors manipulate forms in terms of the carving operations required to generate them from a simple block, draftsmen use projective geometry, and an art historian is likely to use historical analogies. The internal representations used by humans in design thus evolve from perceptual and tactile experience [1]. These representational differences are an important source of variation in human design.

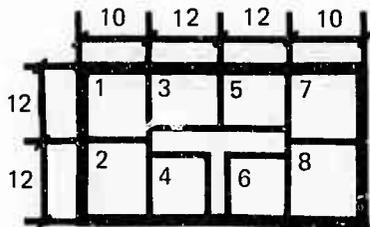
#### System Configurations for Computer-Aided Design

System design research here at CMU has followed an evolution represented by a sequence of programs for computer-aided design. To facilitate later reference to them, I shall first give their names. The first large effort completed here was Grason's GRAMPA, implemented in 1970. This was followed by Eastman's GSP [7,8] and Pfeifferkorn's DPS [16]. In 1972, Yessios implemented FOSPLAN [20], then in 1973, SIPLAN [21]. Several small programs have also been implemented during the same period. Below I review each of these systems in terms of their representation of space and treatment of constraints. Lastly, I outline research in design languages.

A basic issue to be resolved in the design of any computer system is the organization of data for easy manipulation. The issue has broad implications, as different characterizations of the original design task lead not only to different data structures, but also to nonisomorphic operators that may cause drastic differences in problem solving difficulty. I am speaking, of course, of the ubiquitous representation issue. A variety of representations of the physical elements and space involved in design have been developed and explored here at CMU. A common property of all of them has been their explicit treatment of the integer constraint regarding allocations in the space-time continuum — any point in space may be occupied by only one element at a time.

One of the earliest representations used was the *variable domain array* [6,8]. See Figures 1a and 1b. It is a two- or three-dimensional array, each variable with non-zero subscripts representing a rectangular domain. The dimensions of the domain were defined in the zero vectors in each dimension; the X and Y dimensions of  $\alpha_{ij}$  were  $\alpha_{i0}$  and  $\alpha_{0j}$ , respectively. The values of the non-zero variables characterized the state of each space, *e.g.*, whether empty or filled and, if filled, by what object. This representation, while limited to rectangular domains, incorporates certain features which seem highly desirable for CAD systems and which have been incorporated into representations developed later. Both filled and empty space are characterized, allowing the easy locating of new objects in non-overlapping arrangements. The value stored to depict an occupied domain is also a pointer that may be used to reference properties, spatial or non-spatial, not defined in the array. It also shows the relation between domain locations and sizes; a complete description of size allows derivation of location through proper summations. The converse is not true; in the general case, all locations do not allow derivation of sizes. The mapping from sizes to locations is from many variables to one. An early program using the variable domain array was written by Moran in LISP [14]. Later work has relied on ALGOL and FORTRAN [8].

An alternative representation was developed in John Grason's thesis research [11] and consisted of a dual, colored, and directed graph. See Figure 1c. Instead of representing domains, each variable depicts adjacencies between empty or filled spaces. The dashed edges depict west-east adjacencies, while the solid edges depict south-north adjacencies. Direction of the edges, *e.g.*, to or from a node, depict orientation. A node depicts a space. Locations are altered by reconnecting edges. Overlaps never occur as long as the graph remains planar. This colored and directed graph is the dual of a graph in which edges depict walls in the standard manner. The coloring and directions impose a one-to-one mapping between a floorplan and this form of graph. This representation has many similarities to the variable domain array. Notice the correspondence between edge values in the dual graph and the zero vector values in the variable domain array. Yet the dual graph introduces many unique efficiencies not available in the array. These will be described more fully later. Both of these representations are limited to rectangular approximations of more complex shapes. Both are also easily extended to three dimensions.

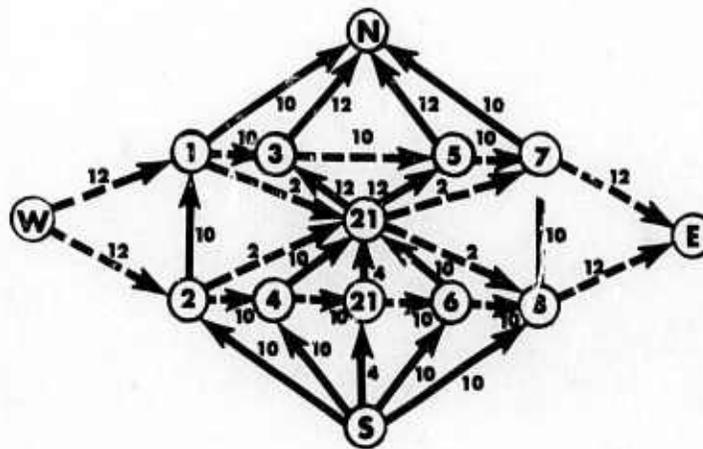


(a) Orthographic Drawing

	10	10	2	2	10	10
10	1	3	3	5	5	7
2	1	21	21	21	21	7
2	2	21	21	21	21	8
10	2	4	21	21	6	8

(b) Variable Domain Array

10



(c) Dual Colored Graph

Figure 1 Spatial Representations

More recently, Charles Pfefferkorn developed a general two-dimensional representation in DPS, as part of his Ph.D. thesis [16]. It consisted of a set of convex domains, each described in terms of its perimeter edges. The map of the data structure used for a single element is shown in Figure 2. A new domain is added by entering its edges one at a time to partition the current domains. This representation may spatially characterize any two-dimensional shape and checks overlaps by restricting the partitioning of domains to those that are empty.

Each of these representations has associated with it facilities for describing objects and spaces, and operators for generating arrangements. While each is conceptually quite simple, each provides quite distinct capabilities when particular types of problems are considered. In terms of a two-dimensional repre-

sentation, Pfefferkorn's is general and provides the capabilities needed for CAD. Only integrating the representation of objects in a way that compliments the treatment of constraints — as Grason's GRAMP has done — would be an improvement.

All problem formulations used to date have been in terms of constraints, that is, tests which return a Boolean predicate. These tests may reflect technological requirements (maximum distance between a memory box and CPU), public safety or building code criteria (width of a stairway), or good design practice (all offices should have windows). In the most general case, these constraints are Boolean functions of unlimited complexity and undefined internal structure. Constraints regarding adjacency, access, distances, sightlines, and orientation have been implemented in this fashion.

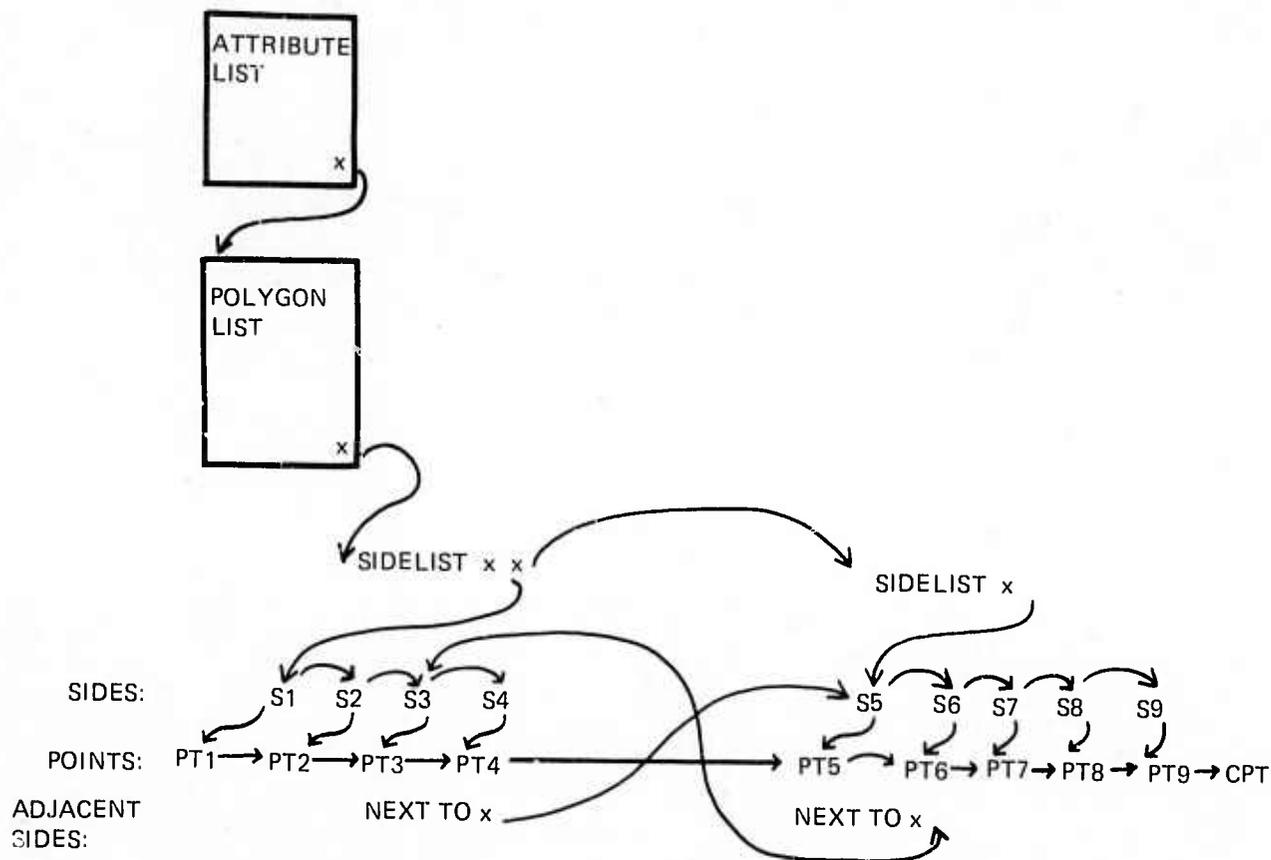


Figure 2 A Map of a Single Object Description in Pfefferkorn's DPS [16]

It was quickly learned that at least two structurally distinct kinds of constraints were involved in design problems. Consider a constraint regarding adjacency. Once two elements are adjacent they will only become not adjacent if one of the adjacent elements is moved relative to the location of the other. Consider now a sightline constraint between two elements. The relocation of any element may alter the value of this constraint. We call the first type *local* and the second *global* [9]. Local constraints are much easier to deal with; they need be tested only when an object which is a predicate of the test is altered. Corrective operations when a local constraint fails are also much simpler to diagnose. It seems possible in many cases to redefine global constraints so that they become local, without loss of generality. For example, once a sightline required between two locations is satisfactory, the program may assign the space required to be clear between them as (in a

sense) solid. No other objects can then be located there and the test need not be repeated. Development of a general set of local constraints is an important objective in both computer-aided and automated design.

Each of these Boolean functions can be computationally expensive. Moreover, it seems unlikely that one can define a reasonably small set of tests that would satisfactorily define different design problems, even if they were all limited to a restricted domain. An alternative approach was incorporated into Grason's GRAMPA. The properties of the dual graph have an interesting relation to a particular set of spatial constraints. Specifically, there is a one-to-one correspondence between many constraints and single or small sets of variables within the dual graph representation. Adjacency, in the general sense, is denoted by the existence of an edge. The value of an edge denotes the length of common border among ad-

jacent elements. Dimensions of a space are denoted by the sum of edges of one direction and color attached to a node. Orientation is denoted by color. In this representation, a problem is defined as a partially specified graph. A solution is a complete planar graph satisfying constraints regarding the value and ordering of edges to a node. This representation reduces constraint testing to triviality, but with the added cost of a more complicated evaluation of planar feasibility.

A third approach for dealing with constraints is called Constraint Projection. Instead of a Boolean function for each constraint, the system incorporates procedures for defining the spatial domains of feasible locations and the corresponding range of feasible orientations. Multiple constraints are treated by defining the domain and orientation range for each constraint, then the appropriate set function, combining them to result in a final feasible domain. Thus, a set of constraints can be reduced (without ever applying them to the arrangement) to a single one. Reduction is a very desirable capability for automated design systems, as it is for other types of problem solvers.

Each of the above methods of treating constraints imposes strict restrictions on the design representation. Generality of the shapes characterized by a representation is only the first criterion in the development of data structures for CAD. Another issue is the base language for its implementation. Yessios has explored a range of data structures for computer-aided design, their specifications regarding shape and arrangement, and various grammars for combining them [20,21]. His work can be considered in two different but equally valuable perspectives. One is that these languages will provide the primitives for higher level CAD systems; this is a traditional perspective. The second view is that a major task in design is translation. A design problem first is an existence question regarding the mapping of a set of statements in one representation into a spatial one. If a mapping exists, *e.g.*, the design is feasible, then the iterative step is one of (a) examining the spatial realization of the first design problem and redefining the original problem statement based on this new information, or (b) applying more complex analyses to the statement and using the results to generate a new problem statement. This second view marks an advance in the conception of man-machine organization, for it partitions design tasks according to the formal definition of their complicatedness, *e.g.*, those problems that are syntactically resolvable within restricted grammars and all others.

### Expansion of the Contributions of the Computer to Design

The description by Coons at the beginning of this paper implicitly partitions the tasks between man and machine. The machine does analysis and numerical studies; the man uses his "creativity" to solve design problems. This *a priori* conception of the two partners' contribution is too limited. We believe that a computer has at least the potential for providing the same skills as a "dumb" draftsman and that some analyses will forever remain the province of visual examination. In the former case, the computer should be able to respond to the description of simple, well structured design problems and generate solutions for them. It should be able to modify its solutions as new information is received from the designer "looking over its shoulder".

A good portion of our research has focused on the automatic generation of the spatial arrangement of physical elements. The general formulation is given:

$s$  ::= a space, bounded or unbounded;  
 $b_1, b_2, \dots, b_m$  ::= a set of elements of fixed or variable shapes;  
 $c_1, c_2, \dots, c_n$  ::= a set of constraints defining required relations between two or more elements and the shape of single elements;  
 $d_1, d_2, \dots, d_p$  ::= a set of operators for mapping elements into the space in different ways and possibly for altering their shape;  
 $e^0$  ::= an initial arrangement, which may simply be  $s$ ;

find:  $(e^i, b^{i+1}, d^{i+1}) \rightarrow (e^{i+1} | e^{i+1} \Leftrightarrow (c_1, c_2, \dots, c_n))$

where  $\Leftrightarrow$  is a matching operation. This formulation presents space planning as a state space problem involving a search through the ubiquitous OR tree. The task is the efficient search of this tree. In contrast with other heuristic search tasks, at least four unique issues are involved in the above formulation:

- a. Location operators – if there is more than one, there are a countably infinite number of locations for any element within a space. Which subset of locations is worth considering at any state of a design problem, that is, how should the location operators be specified? Manual design gives no direct answer to this problem.
- b. Similarly, there may be countably infinite shapes satisfying the shape constraints of an element, but far fewer when all are considered in a single arrangement. What shape operations are effective in finding this subset, and how should they be combined with the location operations?
- c. Given effective operators, what search strategy is most likely to lead to a solution quickly, with minimal states being generated?
- d. Given the large number of variables required to describe a state (six for location of each object in 3-space plus an undefined number for its shape), what bookkeeping procedures are most effective in guaranteeing that search will proceed without looping?

Each of these issues has received attention in CAD research here at CMU.

The location problem has been treated by a variety of heuristic methods and one exact one. Pfefferkorn's DPS, for instance, identifies each convex corner of the empty space as a possible location and places its reference on a list to try [16]. Grason's program tries adjacencies (of rectangular objects) with corners aligning [12]. Both of these are heuristic. Constraint Projection provides an exact method for dealing with the location problem [9]. It derives a reduced domain from the set of domains characterizing the constraints of an element. The reduced domain depicts a homogeneous region within which any location satisfying the orientation requirement is equally acceptable.

The shape definition problem is the feasible solution to two sets of constraints, one set defining the "internal" and constant requirements delimiting acceptable shapes, and another imposed exogenously on this set by context, delimiting the locations the shape may occupy. Two types of shape generation operations have been tried. The first was initially implemented by Sutherland in SKETCHPAD and consisted of a set of (possibly non-linear) equations specifying properties of the set of points used to define the

perimeter of an object. Whenever a new context delimits the location of one or more points, the shape is redefined using a least-squares, iterative convergence method [17]. We at CMU have explored an alternative method of variable shape definition based on generative assumptions. Using a primitive which enlarges a portion of an object so that the resulting form satisfies a group of (variable) tests within the primitive, we have been able to develop sequences of calls to this expansion operator so that one, or a whole set, of variable shaped elements are formed that satisfy both internal and relational criteria. The sequence of expansion is again a tree (AND - OR) and the objective is to search it with minimal backtracking. Our efforts have been directed toward pipe and duct layout, circulation, and room arrangement [10].

Given the large set of variables which describe a design and the complex relations among some of them, an important question arises concerning the general method for bounding them that will fulfill a set of constraints imposed by a user. As described earlier, this question has been formulated within a state-space heuristic search representation.

Any OR-tree is easily considered as a Boolean function. Using minimal assumptions regarding the final distribution of elements, we have developed search decision rules which minimize the cost of evaluating this form of Boolean function. The search is efficient in finding a solution if one exists; this is the criterion driving the search process. But if no solution exists, our procedures resort to implicit enumeration and may waste much time fruitlessly [9]. Currently, we are trying to develop a practical failure criterion.

A very large number of variables is required to describe any state in CAD. In order to guarantee that a program does not generate equivalent states and therefore loop, some trace of past states is required. A single general approach has been used in the programs developed at CMU, with different variations. All have only considered arrangement variables with no shape variation and are based on an assumption of a depth-first search. Given a lexicographic ordering of locations for each element and a fixed sequence for manipulating each element (corresponding to a level in the tree) a pointer to the current location of each element defines both the current state and all others that have been considered.

Pfefferkorn relied precisely on this technique in DPS. When initially considering each element, a TRY-list was generated and ordered heuristically. Backtracking requires only a pointer to the current location of each element and an ordering of the elements. Different orientations of an element were tried at each location. Eastman's program relied on location operators which automatically generate a single next alternative in a lexicographic order. Book-keeping requires that each operator internally identify whether or not it is able to define a location, and that the program keep track of the first location generated when the process is moving down the search tree.

These types of approaches greatly simplify the state description but lead to other complications. In particular, the location operators we have used generate different locations for each arrangement of elements. This means the TRY-list must be regenerated each time an element higher in the search tree is relocated. This is done in both of the above programs. But it also means that different orders of objects generate different locations and thus result in different search trees. Eastman's GSP does not allow element reordering and is limited to searching arrangements resulting from the program's estimation of the most efficient ordering. Pfefferkorn's allows limited reordering.

#### The Direction of Future Research in Computer-Aided Design

Few of the problems reviewed above have been completely resolved. We have only begun to consider the requirements for CAD systems implied by analyses of the tasks of design.

Current research is proceeding in a variety of areas described above, including data structures for three-dimensional objects, the development of a constraint language for describing any kind of spatial relationship between elements, and problem decomposition. In addition, we are exploring alternative methods for bounding the search process in large arrangement problems. That is, when should a program "give up" looking for (a) feasible arrangement(s). Two approaches to the bounding problem show merit. The

first is to use information found in a partial enumeration of the tree to generate a proof that a solution cannot exist in other parts. This requires that axioms be induced from a set of failed search states. For example, in Figure 3 it is intuitively easy to see that if the sum of the areas of A,B,C,D, is smaller than  $X + Y$  but greater than  $X$ , then one or more of the objects must fit in space Y for an arrangement to be feasible. We are exploring how arithmetic analysis over various partitions of the problem space may be used to guide and bound search in CAD.

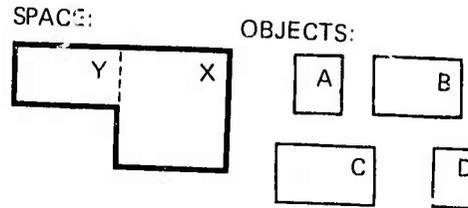


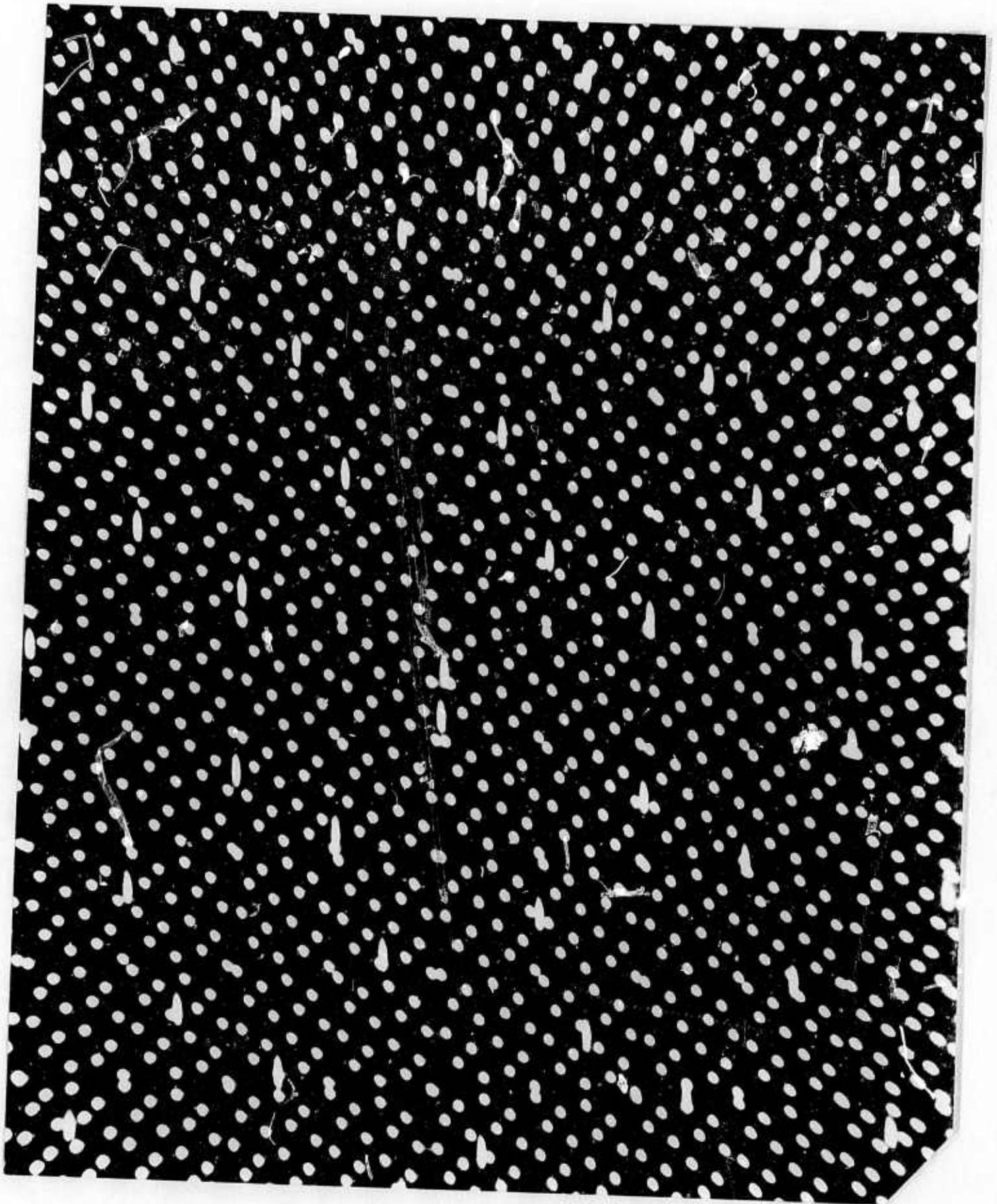
Figure 3

A second approach to bounding search is in terms of cost effectiveness. Can the probability of finding a solution be dynamically estimated as search proceeds to allow derivation of an expected cost of search? If so, this also would be an effective criterion for stopping search after partially enumerating the tree of possibilities.

Computer-aided design, particularly when it includes synthesis capabilities and spatial considerations, has a richness of issues possibly unparalleled among the problems now being investigated by the AI community. Moreover, results have many applications, including the direct ones for CAD, but also for robotology (representations of the physical environment, the planning of manipulation tasks) in both space and industrial applications. The design implications range from architecture, to computer design, to regional land use planning, to controlling pollution effects. We at CMU expect to continue our program of research in augmentation of the design process.

## References

1. Ballay, J. M., "Visual Information Processing in Problem Solving Situations," *EDRA, Proceedings of the 1st Environmental Design Research Association Conference*, H. Sanoff and S. Cohn (eds.), North Carolina State University, 1970.
2. Coons, S. A., "An Outline of the Requirements for a Computer-Aided Design System," *Proc. SJCC*, Detroit, Mich., 1963, 229-304.
3. Eastman, C. M., "Explorations of the Cognitive Processes of Design," ARPA Report, Computer Science Dept., CMU, DDC 671-158, 1968.
4. Eastman, C. M., "Cognitive Processes and Ill-Defined Problems: A Case Study from Design," *Proceedings International Joint Conference on Artificial Intelligence*, Washington, D.C., D. Walker and L. Norton (eds.), May 1969.
5. Eastman, C. M., "On the Analysis of Intuitive Design Processes," *Emerging Techniques of Environmental Design and Planning*, G. Moore (ed.), MIT Press, Cambridge, Mass., 1969.
6. Eastman, C. M., "Representations for Space Planning," *CACM*, Vol. 13, No. 4, April 1970, 242-250.
7. Eastman, C. M., "GSP: A System for Computer Assisted Space Planning," *Design Automation Workshop Proceedings*, Atlantic City, N. J., 1971.
8. Eastman, C. M., "Preliminary Report on a Language for General Space Planning," *CACM*, Vol. 15, No. 2, February 1972, 76-87.
9. Eastman, C. M., "Automated Space Planning," *Journal of Artificial Intelligence*, Vol. 4, 1973, 41-64.
10. Eastman, C. M., and M. Schwartz, "A Generative Method for Creating Variable Shaped Objects in Design," *Institute of Physical Planning Research Report No. 34*, CMU, January 1973.
11. Grason, J., "Methods for the Computer-Implemented Solution of a Class of 'Floor Plan' Design Problems," unpublished Ph.D. Dissertation, Computer Science Dept., CMU, March 1970.
12. Grason, J., "An Approach to Computerized Space Planning Using Graph Theory," *Design Automation Workshop Proceedings*, Atlantic City, N. J., 1971.
13. Moran, T., "A Grammar for Visual Imagery," CIP Working Paper, CMU, 1969.
14. Moran, T., "Structuring Three-Dimensional Space for Computer Manipulation," Computer Science Dept., Working Paper, CMU, June 1968.
15. Newell, A., and H. A. Simon, *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, N. J., 1972.
16. Pfefferkorn, C., "Computer Design of Equipment Layouts Using the Decision Problem Solver," unpublished Ph.D. Dissertation, Computer Science Dept., CMU, May 1970.
17. Sutherland, I. E., "SKETCHPAD: A Man-Machine Graphical Communication System," *Proc. SJCC*, Detroit, Mich., 1963, 329-346.
18. Yessios, C., "Design Protocol Analysis," Architecture Working Paper, CMU, 1969.
19. Yessios, C., "Synthetic Structure for Site Planning," *Environmental Design Research*, W. Preiser (ed.), Dowden, Hutchinson, and Ross, Inc., Stroudsburg, Pa., 1973.
20. Yessios, C., "FOSPLAN: A Formal Space Planning Language," W. Mitchell (ed.), in *Environmental Design: Research and Practice, Proc. Fourth Environmental Design Association Conference*, UCLA, Los Angeles, Ca., January 1972.
21. Yessios, C., "Synthetic Structures and Procedures for Computable Site Planning," unpublished Ph.D. Dissertation, School of Urban and Public Affairs, CMU, April 1973.



## On the Scheduling Aspects of Timing Concurrent Processes

A. N. Habermann

### Introduction

The design and development of operating systems has enriched computer science with interesting studies on control and data structures. Two such contributions are the studies of phenomena associated with concurrency and of the design and implementation of scheduling strategies.

The purpose of this paper is to examine briefly the impact of scheduling on programming timing constraints in concurrent processes. A variety of aspects related to this topic have been discussed in a series of papers and reports in recent years; this paper reviews and summarizes the overall result of this work.

The first section shows what sort of flexibility is desirable in programming timing structures on behalf of process scheduling. In the next section two general timing structures are discussed that allow implementation of arbitrary scheduling rules. Subsequently some of the verification methods are reviewed that are based on using the properties of timing rules and the structure of control programs. Finally, the class of problems is considered that asks for an implementation of priority rules by means of timing structures. A recent study showed that these problems can be solved by means of one unifying principle of representation.

### Timing Concurrent Processes

The fact that concurrent processes share resources in the form of devices, programs, and data gives rise to possible conflicts of interest. Dijkstra has shown how such conflicts can be resolved using critical sections [5]. It was shown in a series of papers [4] that these can be implemented using the "read/write cycle" of a machine as the most elementary critical

section. Critical sections of arbitrary length are often programmed by means of two simpler critical sections: one at the beginning guarding the entrance and one at the end controlling the exit. The function of the simple one at the entrance should be to grant or deny its caller permission to enter. The function of the one at the exit should be to record that a process is leaving, and to grant entrance permission, if possible, to one or more of the processes which was denied earlier.

Because of this general structure it seems appropriate to devise two standard critical sections, one for entrance and one for exit, and to use these for programming critical sections of arbitrary length. Various proposals to this effect have been considered and a variety of such "primitive critical sections" have been implemented. There are even proposals to base the whole timing issue on such primitives [1]. Representatives of two major categories are the operations LOCK and UNLOCK [23] and P,V operations [6]. An advantage of primitives is that a waiting process does not waste any time of a processor that it possibly shares with the very process that will wake it up. Another advantage of P,V operations is that these can easily be extended to handle critical sections of which several may be executed simultaneously (Dijkstra's counting semaphores), whereas LOCK and UNLOCK do not allow such an extension. Finally, a difference between the two (which might not be seen as an advantage) is that the order in which processes pass a P operation is fixed by the chosen implementation, so programs could rely upon that order, whereas it is hard to predict which process will pass a LOCK operation when several are trying to do so.

The use of standard primitives, however, is absolutely inadequate for large critical sections such as those needed for allocation and use of resources. Using the primitives is inappropriate, generally speaking, if the situation has one of the following three characteristics:

1. it matters which process is selected when one of several is considered for entrance permission;
2. it may not be wise to grant permission because of a possible deadlock;
3. the decision to grant permission may be regretted if later permission must be withheld from a process for which entering is more urgent.

An example of resource management illustrates such characteristics. Suppose ten identical magnetic tape drives are pooled among three types of processes:

P-type processes need one tape unit at a time;

Q-type processes need two units during some period of time (*e.g.*, for copying);

R-type processes need three units during some period of time (*e.g.*, for updating or tape correction).

18

We spot easily the deadlock which will occur if, for instance, five R-type processes should succeed in seizing two drives each [10]. Also, it may be wise to select a process from the processes waiting for tape drives based on an external priority and the number of drives it already has in use. But when such a selection strategy is implemented, another problem may arise, namely, that of permanent blocking [18]; for example, an R-type process may never be selected because a P-type or Q-type process happens to be waiting at all times. Finally, the decision to grant the last free drive to a newly arrived R-type process may be regretted if, shortly afterwards, another R-type process requests its third drive. It is not surprising that the impact of scheduling on timing structures is not always correctly appreciated [17].

LOCK and UNLOCK would not provide any means of dealing with these issues. Programming the use of drives within a critical section  $P(\text{drives}) - V(\text{drives})$ , where "drives" has the initial value 10, would mean that special algorithms for dealing with the particular circumstances would have to be programmed as part of the P,V operations themselves. Thus, extrapolating to other situations, there would be a need for as many versions of P,V operations as there are different circumstances, but this is in conflict with the idea of standard primitives.

On the other hand, there is an important type of critical section for which a standard implementation with primitives is perfectly adequate. This is the type for which the probability that more than one process will be waiting when an earlier one reaches the exit is close to zero. We will use in this paper the attribute "small" for a critical section that has this property. Small critical sections are usually short pieces of code without potential delay or repetition with a large or unknown bound. Any of the primitives is suitable for programming small critical sections, but P,V operations are still preferable when a CPU is multiplexed [6].

### Cooperating Processes

The term "cooperating" describes the situation that a process P may have to wait until another process signals the occurrence of an event E. This is a fairly common relation between processes, *e.g.*, when processes communicate [14] or when one process controls another. Processes related through common critical sections can even be viewed that way, because once a process has entered a critical section, it must cause the event of leaving it before another process can get permission to enter.

Cooperation of processes can be described in terms of operations "wait" and "signal" that operate on eventnames. It is possible to implement these operations as P,V operations, but we must realize that the delay in such a P-operation is even less predictable than when used to enter a critical section. We must assume that the process that waits on an event cannot find out when another process will signal the occurrence of that event; it may not even know from which process a signal can be expected. This means that standard primitives are also inadequate for implementing wait and signal operations because of problems with selection, deadlocks and regrettable decisions.

It has been shown that these problems can generally be solved by applying "private eventnames" [12]. The attribute "private" means that an eventname  $E[i]$  associated with a process  $P[i]$  will exclusively be used by other processes to signal  $P[i]$ , whereas  $P[i]$  is the only process that will ever wait on the occurrence of  $E[i]$ . The point about private eventnames is that the selection problem is entirely separated from the implementation of the wait operation, because, no matter how long a delay is caused by wait ( $E[i]$ ), there is only one process that ever will wait on the occurrence of  $E[i]$  and thus, this is the only one that could be selected!

Greater flexibility for implementing the necessary scheduling is now achieved by either one of the following methods:

1. implement wait and signal not as P,V operations, but as critical sections in which scheduling can be programmed as needed;
2. have the processes involved send requests and completion notices to a controlling agent that acts as a policeman regulating the traffic according to well established rules.

Note that the second method does not eliminate our task of programming cooperation among processes; it only moves the interaction from pairs of processes having equal rights to individual members of the community that must cooperate with a central agency. (It is the difference between placing STOP signs or traffic lights at a street intersection.)

When the first method is applied, entrance is programmed as a small critical section followed by a wait operation on the private eventname of its caller. The program within this critical section is small enough to allow the use of P,V operations for delimiting it. Its function is to investigate whether the process can enter; if not, the process will be delayed in the subsequent wait statement. Exit is also programmed as a small critical section for which P,V operations are adequate open- and close-brackets. Its function is to see whether the change of state, caused by leaving, would allow one of the processes that is waiting on its private eventname (if any) to continue.

It has been shown that such constructs as entrance and exit behave as P,V operations and so these are certainly sufficient to implement arbitrary critical sections [12]. But the great advantage gained over plain P,V operations is that we have not committed ourselves to the programs for permission or selection and, thus, we have not made decisions which are unnecessary for implementing large critical sections. The only restriction on programming permission and selection is that the critical sections for entrance and exit should be small (in the technical sense of this paper).

The second method of dealing with selection, deadlocks, and priority rules by means of a central agent is appealing because it seems to separate those issues nicely from the structure of the programs to which they apply. It is certainly true that those programs will have a simpler structure, but overhead is likely to increase due to the additional calls on the agent, and the possible need to reconstruct lost information. A process may have to call the agent for various reasons, *e.g.*, when requesting a resource and when releasing one. At the place where the agent is called, the reason for calling is perfectly well known.

However, this information must be transmitted explicitly with the call, and the agent must find out for what reason its services are required. Thus, information that is present is lost through a uniform call on the agent and must be reconstructed when the agent is activated. In order to preserve the idea of allowing the environment of the processes to handle selection and other issues, one could split the agency into individual agents each to be called for a particular task. This indeed seems an acceptable solution under some circumstances [7], but in other cases such a solution is not feasible, as for instance in case of peripheral device control. The hardware makes it necessary that only one agent controls a peripheral device and it must regulate all requests for device operations.

Working with an agent, however, still does not remove the task of programming timing structure for processes of unequal rank, because cooperation must then be programmed between the processes and the agents. It seems that the use of an agent is to be recommended for complicated hierarchies or a great variety of ranks or complicated priority rules. But it also seems worthwhile to pay special attention to the effect of timing rules on the cooperation of processes partitioned into a small number of fixed ranks, as is the case with an agent and its callers.

#### Verification of Timing Rules

The additional complexity caused by concurrency prohibits a straight-forward extension of Floyd's method of inductive assertions [8] to concurrent processes [19]. The number of states to be considered explodes even for trivial systems. Not only is there the problem of finding the right assertion, but it could easily be the case that the correctness proof itself is much longer and an order of magnitude more complicated than the programs involved [20].

A more promising method was found in the same spirit as the axiomatic approach for proving program correctness [16] and proving the correctness of APL programs [9]. The approach that these methods have in common is to exploit the structure of the given programs in the correctness proof. In Hoare's system, properties of control structures are expressed in the form of axioms and can be used in that form in a correctness proof. In Susan Gerhart's thesis, properties of APL operators are formulated precisely and thus lead to a more concise and tractable correctness proof [9].

Since "the state" of a system of concurrent processes is a rather vague notion in any case, it makes more sense to show that there is an abstract representation which has certain desired properties that will not get lost when going to more detailed versions. Some success was scored in this way with respect to properties that can be derived from timing structures in concurrent processes [11,12]. It was found that the working of  $P(E)$  and  $V(E)$ , or  $\text{wait}(E)$  and  $\text{signal}(E)$ , can be characterized by the fact that a certain relation remains invariant under these operations. The relation says that the number of times permission was granted to continue after a  $\text{wait}(E)$  equals the minimum of the number of attempted  $\text{wait}(E)$ 's and the number of executed  $\text{signal}(E)$ 's incremented by an initial constant.

The verification method using the invariant relation was applied to a useful communication system. Not only could it be proved that the communication was deadlock free, but other interesting properties also emerged from the analysis. For example, it was shown that senders and receivers could access the bounded communication buffer at the same time without getting into a conflict when the buffer was empty or when a first message was placed. Moreover, a natural simplification of the control programs was found for the cases that either the group of senders, or the group of receivers, or both, were reduced to one process. It was later shown that the invariant could also be used to verify the correctness of a more complicated communication system in which senders, or receivers, or both, may get ahead of one another [24].

The property verification method was modestly successful when applied to the Cigarette Smokers Problem [22]. A solution of this problem was presented in the form of a Petri net and it was shown that this problem could not be solved with a restricted form of Dijkstra's semaphores without the use of some form of conditional statement. But a solution using semaphore arrays was soon found [21] and the property verification method was applied to that solution [13]. The method proved to be rather successful in that a generalization of the problem could be proved as easily as the given one. Moreover,

its application clarified considerably the relation between the problem specification and its solution, with the result that other interpretations of the problem statement could be analyzed as well. The result is nevertheless rated as modest mainly for two reasons: first, the proofs are rather long, and second, there is no precise model or formal description.

It seems not satisfying that the verification is so much longer than the program text it is trying to verify. The cause in this case is not so much the number of states to consider, but the combinatorial problem of showing that participating processes cannot get into, or remain in, certain states when certain events happen. So this problem can ultimately be reduced to the second one: the lack of a precise model or formal description.

Another consequence of this second deficiency is that one is never sure whether or not the proof is complete and exhaustive. It is never clear what may be assumed as obvious and what must be proved. In going over the proof one must be reconvinced each time that nothing has been omitted. A solution for both problems may be found in recent results which offer a representation of some classes of timing problems in an abstract model; this allows a more precise and concise treatment. A brief discussion follows in the next section.

#### An Abstract Model for Some Timing Structures

Considerable activity was aroused recently by the Readers and Writers Problem [2]. The basic characteristic of the problem is to implement certain priority rules by means of a timing structure. The problem is that two groups of processes perform an action exclusively, but one of the groups has preference over the other, or more precisely:

1. when a process  $P$  of group  $P$  performs action  $A$ , no process  $Q$  of group  $Q$  is permitted to perform action  $A$  in an overlapping time interval;
2. if neither a  $P$  nor a  $Q$  is performing  $A$ , any one of them must be able to start action  $A$ ;
3. (preference rule) if there are processes of group  $P$  waiting to perform action  $A$ , at least one of these should get permission to do so as soon as the processes currently executing  $A$  are finished.

In addition to these rules one can specify whether or not processes in one group have to perform action A exclusively among one another. The Readers and Writers Problem is stated in two versions: one in which preference is given to Readers and another one in which Writers have priority. This accounts for two of the four possible cases, because Readers do not have to perform action A exclusively, whereas Writers do. The other two cases are, using the same terminology: two groups of Readers of which one has priority, or two Writer groups one of which has preference over the other.

In the original paper the programs for Readers and Writers were presented and their correctness was reasoned in an informal way [2]. Other solutions were proposed in which an attempt was made to design symmetry in the programs for both groups [15], but the authors of the original paper showed the inadequacy of such a solution [3].

Since solutions of such problems depend on timing structure, the invariant for P,V operations mentioned in the preceding section was tried to show that the presented programs indeed have the necessary properties to enforce the required rule of preference. The result was as in previous cases: analysis clarified some deficiencies of the presented programs, but involved rather long proofs based on an informal model.

More recent investigations may overcome the earlier difficulties. These go quite naturally in the direction of a formal model in which timing structures can be represented. The timing structure of a process is represented in this model as a regular expression in which the terminal symbols are brackets representing the wait and signal operations. The timing rules are very simply expressed in terms of state transitions of an automaton with the ground rule that only one process at a time can cause a state transition by placing a bracket. If counting semaphores are not considered, the additional rules are:

1. an open-bracket can be placed at any time;
2. a close-bracket cannot be placed unless the corresponding open-bracket is present;
3. when a close-bracket is placed, it cancels out the corresponding open-bracket and both are deleted.

Considering counting semaphores means that a multiplicity of brackets of one kind is allowed, and in particular the initial state of the automaton may contain several open-brackets of one kind. Verification of properties due to timing structures can be carried out in a more precise way using this model and the proofs seem to be significantly shorter for all the cases mentioned here.

## Summary and Conclusions

Implementation of timing structure in concurrent processes results in a need for scheduling. The use of standard primitives such as P,V operations is inadequate in circumstances where selection of a process is based on a priority measure, deadlock situations, and likely decisions in the near future. Sufficient flexibility can be achieved, however, when timing operations are programmed as combinations of small critical sections and operations on private eventnames.

Programming a central agent that performs the scheduling task can bring about more clarity in the structure of the system, but the task of implementing cooperation in accordance with certain preference rules remains present. With respect to such an organization, one should consider the sorts of scheduling which can be achieved by timing rules for a small number of preference classes.

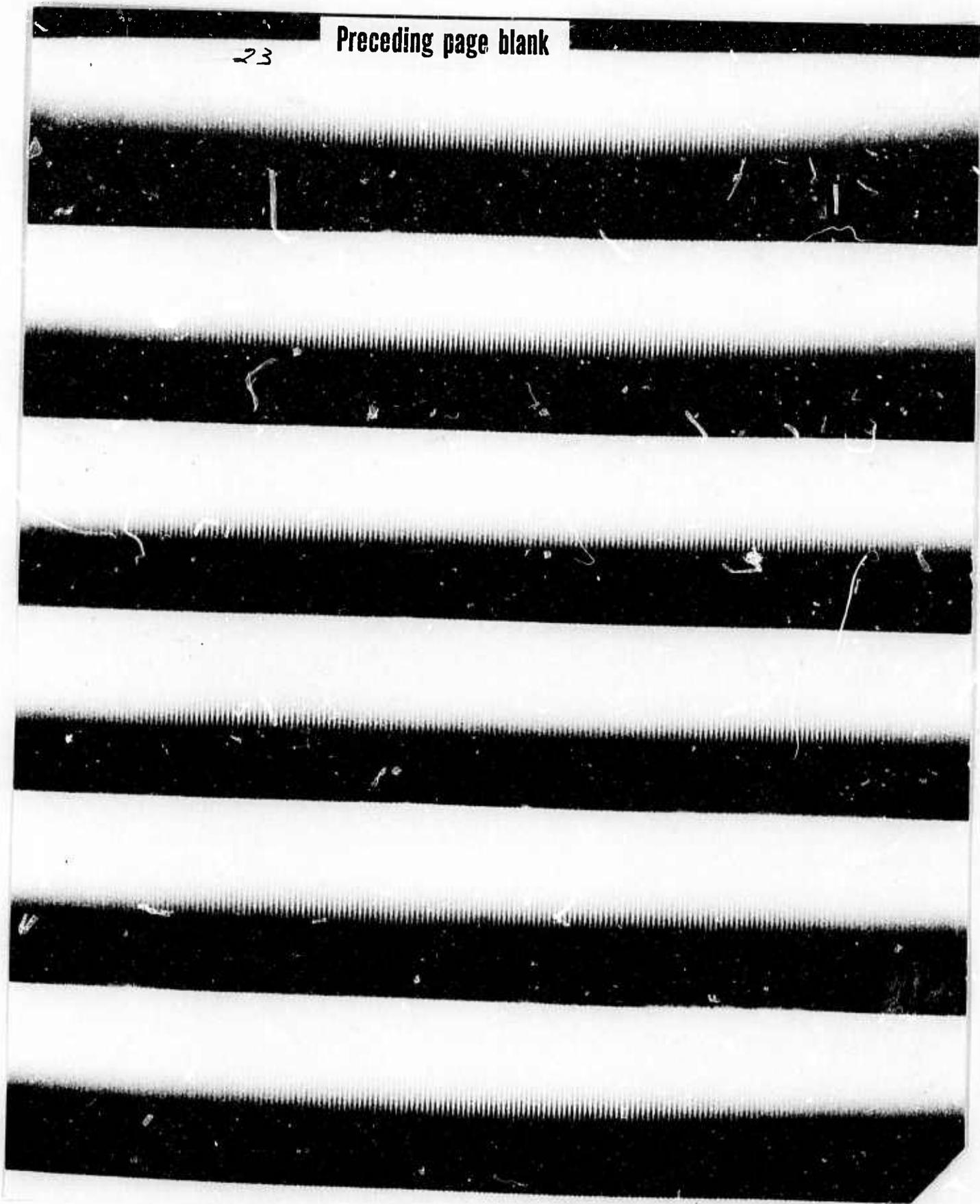
Combinatorial explosion prohibits a useful application of the notion "state" as a composition of the states of the individual processes. For the same reason, there is little hope that Floyd's inductive assertions method can be usefully extended to concurrent processes. Instead, a more promising approach seems to be to prove that an abstract model of the concurrent processes has certain desired properties that will not get lost in more detailed versions of the programs for these processes. Some results were obtained in this way, first by applying an invariance rule to programs with a given timing structure, and more recently by means of an abstract model for the timing structure of concurrent processes and an automaton that simulates their behavior. Satisfactory verification appears to be feasible using this model and it seems worthwhile to investigate what class of timing problems can be treated in this way.

## References

1. Belpaire, G., and J. P. Wilmotte, "An Approach to Concepts of and Tools for a Theory of Parallel Processes," Institut de Mathématique Pure et Appliquée, Université Catholique de Louvain, *Report 57*, December 1972.
2. Courtois, P. J., F. Heymans, and D. L. Parnas, "Concurrent Control with Readers and Writers," *CACM*, Vol. 14, No. 10, October 1971.
3. Courtois, P. J., F. Heymans, and D. L. Parnas, "Comments on a Comparison of Two Synchronizing Concepts," *Acta Informatica*, Vol. 1, No. 4, 1972.
4. Dijkstra, E. W., "Solution of a Problem in Concurrent Programming Control," *CACM*, Vol. 8, No. 9, September 1965; Knuth, D. E., "Comments on a Problem in Concurrent Programming," *CACM*, Vol. 9, No. 5, May 1966; de Bruyn, N. G., "Additional Comments on a Problem in Concurrent Programming Control," *CACM*, Vol. 10, No. 3, March 1967; Eisenberg, M. A., and M. R. McGuire, "Further Comments on Dijkstra's Concurrent Programming Control Problem," *CACM*, Vol. 15, No. 11, November 1972.
5. Dijkstra, E. W., "The Structure of the THE Multiprogramming System," *CACM*, Vol. 11, No. 5, May 1968.
6. Dijkstra, E. W., "Cooperating Sequential Processes," in *Programming Languages*, Genuys (ed.), Academic Press, London, 1968.
7. Dijkstra, E. W., "Hierarchical Ordering of Sequential Processes," *Acta Informatica*, Vol. 1, No. 2, 1971.
8. Floyd, R. W., "Assigning Meanings to Programs," *Proc. Amer. Math. Soc. Symposium in Applied Mathematics*, Providence, R. I., Vol. 19, 1967.
9. Gerhart, S., "Verification of APL Programs," Ph.D. Dissertation, Computer Science Dept., CMU, 1972.
10. Habermann, A. N., "Prevention of System Deadlocks," *CACM*, Vol. 12, No. 7, July 1969.
11. Habermann, A. N., "An Operating System Modeled as a Set of Interacting Processes," *Annual Princeton Conference on Computer Science and Systems*, Princeton, N. J., March 1971.
12. Habermann, A. N., "Synchronization of Communicating Processes," *CACM*, Vol. 15, No. 3, March 1972.
13. Haberman, A. N., "On a Solution and a Generalization of the Cigarette Smokers' Problem," Computer Science Dept., CMU, August 1972.
14. Hansen, P. B., "The Nucleus of a Multiprogramming System," *CACM*, Vol. 13, No. 4, April 1970.
15. Hansen, P. B., "A Comparison of Two Synchronizing Concepts," *Acta Informatica*, Vol. 1, No. 3, 1972.
16. Hoare, C. A. R., "An Axiomatic Approach to Computer Programming," *CACM*, Vol. 12, No. 10, October 1969.
17. Hoare, C. A. R., "Towards a Theory of Parallel Programming," in *Operating System Techniques*, C. A. R. Hoare (ed.), Academic Press, London, 1972.
18. Holt, R. C., "Comments on Prevention of System Deadlocks," *CACM*, Vol. 14, No. 1, January 1971.
19. Lauer, H. C., "Correctness in Operating Systems," Ph.D. Dissertation, Computer Science Dept., CMU, September 1972.
20. Levitt, K. N., "The Application of Program-Proving Techniques to the Verification of Synchronization Processes," *Proc. FJCC*, Anaheim, Ca., 1972.
21. Parnas, D. L., "On a Solution of the Cigarette Smokers' Problem (without Conditional Statement)," Computer Science Dept., CMU, June 1972.
22. Patil, S. S., "Limitations and Capabilities of Dijkstra's Semaphore Primitives for Coordination among Processes," Project MAC, Computational Structures Group, *Memo 57*, February 1971.
23. Spier, M. J., and E. I. Organick, "The Multics Interprocess Communication Facility," *Proc. of the 2nd ACM Symposium on Operating System Principles*, Princeton, N. J., October 1969.
24. Wodon, P., "Synchronization of Communicating Processes," (private communication), 1972.

23

Preceding page blank



## Some Practical Uses For Analytic Models in the Study of Computer System Performance

John W. McCredie

### Introduction

At the 1973 national meeting of the ACM Special Interest Group on Measurement and Evaluation, authors presenting analytic models of computer systems were under constant attack from a large group of practitioners. Two labels ("academics", and "those in the ditches") quickly entered the local jargon. In public sessions and in small informal groups, debaters presented classical arguments about the relative merits of theoretical and empirical studies. The focus of these discussions was the analysis of computer system performance, but many of the arguments had been presented time and again in other domains. The argument that overly simplified analytic models are misleading was countered with the charge that reels of experimental data without an underlying theory are useless. The "Scientific Method" presents a fundamental interaction between theory and empiricism that is apparently lacking in many current performance evaluation studies.

Two reasons why "academic" analytic computer system models often remain unused by "those in the ditches" are: (a) reports describing them seldom contain discussions about their validity for describing empirical observations and (b) often the results are so complicated that users are not willing to invest the time needed to understand the model and its behavior. The main purpose of descriptive models is to account for observed phenomena of physical systems. However, the complexity of most actual systems requires that any particular model must address a limited and constrained subset of state variables. Thus, each model is an abstraction of a particular set of important features of interest to an analyst or designer. Simplifications required to make an abstrac-

tion manageable by a particular solution technique limit both scope and power. Since analytic models are characterized by symbolic formulations and deductive derivations, they require many simplifying assumptions. The consequences of these assumptions must be explored before one applies such models. The following paragraph outlines some of the general ways analytic models may be useful in computer system performance analysis, and the body of the article contains a number of specific analytic examples developed at Carnegie-Mellon.

Probably the two most common techniques used by performance evaluation practitioners are:

- (1) the design, implementation, and analysis of empirical investigations;
- (2) the construction and use of specific, large, complex simulation models.

These two methodologies have areas of applicability which interact with those of analytic models. For example, analytic models often expand to the point where large computational effort is required to calculate results. Often a point is reached when a modest simulation may be a more cost-effective approach. Large simulations may grow into system prototypes. Empirical investigations can provide insight required to design better models, and these models can indicate which of many possible parameters or subsystems are good candidates for more detailed study via simulation and experimentation. Other important uses for analytic models are as reference systems to aid in both the debugging and statistical analysis of simulation experiments.

To be really useful, analytic formulations should include the essential features of a system, or subsystem, and should have solutions that are readily understandable. The necessity of spending excessive computer effort to solve for each parameter value of an analytic model casts doubt upon its usefulness since simulations typically can handle more detailed cases with similar effort. The conclusion from these considerations is that analytic models, empirical investigations, and simulation studies should complement one another. Each technique serves a useful purpose when applied properly.

The goal of this article is to illustrate, with three specific examples, that even though most analytic computer system models are highly simplified abstractions of actual systems, they can be very useful in performance evaluation studies. The first example is a discrete-time Markov model that illustrates the effects of priority scheduling in a closed cyclic service network. The primary uses of this class of model are for educational or demonstration purposes. The second model is a modification of a classic multi-server queueing system. This model is helpful in studying different scheduling algorithms for load leveling in computer networks. The final model allows an analyst to explore economically part of the large design space of a multiprocessing computer system in order to focus attention on areas which need further study.

The level of detail in the following paragraphs varies with each model. The purpose is not to present detailed derivations, but to describe the structures of three different types of models and to summarize the results of the detailed analysis. Since the first model is not too complicated the interested reader should be able to derive the results presented in equations (1) through (7). The second model is more involved, but the reader with some background in queueing theory should be able to derive equations (8) through (10) with little effort. The last model is an application of an important theorem concerned with networks of queues. Both the classic reference for this theorem, and the details of the theorem's application to this model are rather involved. Thus the results of this analysis are presented as an ALGOL procedure so that the interested reader may use the model directly and then check the derivations from the references.

#### Discrete Markov Model

The basic concepts of a Markov process are *system state* and *state transition*. For a discrete-time Markov process it is convenient to assume that the time between transitions is a constant equal to unity. Let there be  $N$  states in the system numbered from 1 to  $N$ . Then for a simple Markov process the probability of a transition to state  $j$  during the next time interval, given that the system now occupies state  $i$ , is a function only of  $i$  and  $j$  and not of any history of the system before its arrival in  $i$ . Thus one may specify a set of conditional probabilities,  $p_{ij}$ , which are the probabilities that a system which now occupies state  $i$  will occupy state  $j$  after its next transition. The transition matrix for a Markov process is the  $N$  by  $N$  matrix whose elements,  $p_{ij}$ , satisfy the following equations.

$$(1) \quad \sum_{j=1}^N p_{ij} = 1$$

$$(2) \quad 0 \leq p_{ij}$$

Consider the following model of a simple multi-programming system. At every point in time there are two jobs in the system receiving, or waiting for, service from one of two subsystems: (a) a central processing unit (Pc), and (b) an input/output (M.drum) system with characteristics similar to a drum. The entire system is synchronized to schedule jobs at the end of every timing interval which is equal to one revolution of the drum. The two jobs which cycle through the system come from two different priority classes. If there is a job from priority class 1 at the Pc, the probability that it will require another interval of Pc time is  $(1-u_1)$  and the probability that it will make a request to the drum subsystem is  $u_1$ . The corresponding probabilities for jobs of priority class 2 are  $(1-u_2)$  and  $u_2$ . If a job makes a drum request, it is blocked from additional Pc processing until the request is satisfied. When a job from class 1 is receiving service from the M.drum system, the probability that it will require another interval of service is  $(1-w_1)$  and the probability that it will finish and return to the Pc for additional processing is  $w_1$ . The corresponding probabilities for jobs from priority class 2 are  $(1-w_2)$  and  $w_2$ .

Whenever a job completes its work and leaves the system, it is immediately replaced by a new job from the same priority class having identical parameters  $u_i$  and  $w_i$ . Figure 1 illustrates the structure of this model.

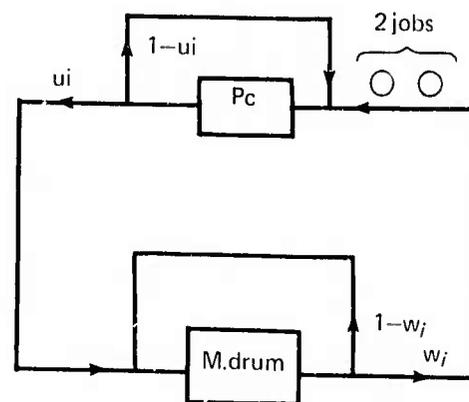


Figure 1 Discrete-Time Markov Model

Define the following configurations as the states of the system:

- S1: Both jobs are requesting service from the Pc.
- S2: Both jobs are requesting service from the M.drum system.
- S3: A job from priority class 1 is requesting service from the Pc and one from class 2 is requesting service from the M.drum system.
- S4: A job from priority class 2 is requesting service from the Pc and one from class 1 is requesting service from the M.drum system.

To specify the system completely we must determine a scheduling rule to decide which job will receive service from a subsystem if two jobs are simultaneously requesting service for the next service interval. Assume that class 1 has the higher priority and whenever two jobs are waiting for service the one from class 1 will be chosen for processing.

The following matrix contains the transition probabilities for this system. Each element,  $p_{ij}$ , is the probability that if the system is in state  $i$  at the end of a timing interval it will be in state  $j$  at the end of the next interval.

j \ i	1	2	3	4
1	$(1-u_1)$	0	0	$u_1$
2	0	$(1-w_1)$	$w_1$	0
3	$(1-u_1)w_2$	$u_1(1-w_2)$	$(1-u_1)(1-w_2)$	$u_1w_2$
4	$(1-u_2)w_1$	$u_2(1-w_1)$	$u_2w_1$	$(1-u_2)(1-w_1)$

The steady state probabilities,  $p_j$ , that this system will be in state  $j$ , after a large number of transitions may be calculated from the following equations.

$$(3) \quad p_j = \sum_{i=1}^N p_i \cdot p_{ij} \quad , j = 1, \dots, N$$

$$(4) \quad \sum_{j=1}^N p_j = 1$$

One may now eliminate variables so that all of the steady state probabilities may be expressed in terms of just one state probability. The following equations are the results of expressing all state variables of this system in terms of  $p_4$ .

$$(5) \quad p_1 = (u_2 + w_1 - u_2w_1 - u_1u_2) p_4 / u_1$$

$$(6) \quad p_2 = \frac{(u_1u_2 - u_1u_2w_2 + u_2w_2 - u_2w_1w_2) p_4}{(w_1w_2)}$$

$$(7) \quad p_3 = u_2p_4 / w_2$$

These results may be used in equation (4) to determine  $p_4$  directly. One may now compute various performance parameters of the system. For example, Pc utilization (the probability that the Pc is busy) is  $p_1 + p_3 + p_4 = 1 - p_2$  and M.drum utilization is  $p_2 + p_3 + p_4 = 1 - p_1$ .

The effects of different scheduling algorithms may be illustrated with this model by assigning different parameter values to the different priority classes. As examples consider the cases of "expected-shortest-job-first" and "expected-longest-job-first" scheduling disciplines. Since jobs from priority class 1 are always processed first we can model these two scheduling algorithms by properly assigning  $u_1$ ,  $u_2$ ,  $w_1$  and  $w_2$ . The mean number of service quanta a job will receive from the Pc and M.drum systems are  $1/u_i$  and  $1/w_i$  respectively. If  $u_1 > u_2$  the Pc will process the job with the shorter mean service request when both jobs are requesting Pc service. If  $u_1 < u_2$  the Pc will process the longer request. When  $u_1 = w_1 = .5$  and  $u_2 = w_2 = .1$  (case 1) Pc utilization and M.drum utilization are both .75. When the scheduling is reversed by letting  $u_1 = w_1 = .1$  and  $u_2 = w_2 = .5$  (case 2) the Pc and M.drum utilization drop to .58. One measure of job throughput is the steady state probability that at the end of a timing interval a job will be leaving the Pc to request M.drum service. When the shorter jobs are given high priority (case 1) the probability is .25 that a class 1 job will be completing Pc service and .025 that a class 2 job will be finishing. When the longer jobs are given high priority (case 2) these probabilities become .05 and .04. Thus in the latter case, short job throughput is reduced by a factor of five, total throughput is reduced by a factor of three, long job throughput is increased by sixty percent, and Pc and M.drum utilization decrease by more than twenty percent.

There have been a number of empirical and simulation studies that have demonstrated the effects of "shortest-job-first" scheduling rules in multiprogramming systems. A recent article by Sherman, Baskett, and Browne<sup>[6]</sup> reviews the results of some of these experiments. Using a trace of real service requests as input, they built a simulation model of an actual multiprogramming system. They found no counter example to the hypotheses that the "best" way to schedule the Pc in such a system is to give it to the job that will compute for the shortest period of time before issuing an M.drum request and the "worst" way is to give the Pc to the job that will compute for the longest period. The objective functions used for their evaluation studies were based on utilization and throughput measures. They did not consider dispatching rules which delayed tasks when resources were available to process them.

The model developed in this section illustrates important fundamental ideas of scheduling theory in a way that is easy to understand, derive, and manipulate. The model may be easily modified to become a continuous-time Markov model of a fully preemptive scheduling policy. Many other illustrative variations are possible. This model has been used successfully in classes at Carnegie-Mellon as the focal point for lectures on scheduling and as the basis for more complicated simulation assignments.

#### Scheduling Model

One of the important goals of networks of computer systems is load sharing. Jobs from a heavily utilized facility may be shipped to a lightly loaded one in order to improve overall system performance. There are many interesting problems concerned with the properties of different scheduling policies for such configurations. A recent paper by Balachandran, McCredie, and Mikhail<sup>[1]</sup> outlines a number of mathematical programming approaches to some of these problems. The analytic model presented in this section focuses upon one small problem from the general area of load leveling policies in computer networks.

Consider a simple network of two computers having processing rates  $u_1$  and  $u_2$  jobs per unit time. Each of these machines may process any job submitted to the network. The machines are functionally homogeneous, but their rates differ. Although the following scheduling policy seems rather complicated, it is conceptually very simple. The basic idea is to process all work at machine 1 until the backlog at this processor is equal to  $(C-1)$  jobs and then utilize machine 2. This policy seems counter productive at

first glance because network capacity will be idle when there are jobs waiting for service. However, if the rate of machine 1, ( $u_1$ ), is greater than the rate of machine 2, ( $u_2$ ), it may be advantageous to build up a backlog at machine 1 before utilizing machine 2. By setting  $C=2$  the policy will direct an arriving job to machine 1 if it is idle, and will immediately assign a new job to machine 2 if it is idle and machine 1 is busy. By setting  $C=1$ , machine 1 will process all work and machine 2 will always be idle. Figure 2 illustrates how this system works.

The scheduling policy under study is the following: at arrival time, schedule a job (1) for machine 1 if it is idle or if there are less than  $(C-1)$  jobs in the queueing system; (2) for machine 2 if there are  $(C-1)$  jobs in the queueing system, machine 2 is idle and machine 1 is busy; (3) for machine 1 if there are  $(C-1)$  jobs in the queueing system and machine 2 is busy; (4) for no particular machine if there are  $C$  or more jobs in the queueing system (for this last case the job will be kept in an "order-of-arrival" waiting line until there are less than  $C$  jobs in the queueing system and then it will be dispatched according to the rules presented above). The basic decision problem for this scheduling algorithm is to choose  $C$ , as a function of  $u_1$ ,  $u_2$ , and the job-arrival rate, so that some measure of system performance is maximized.

An analytic model may be used to investigate this scheduling policy to determine under what circumstances, if at all, it is advisable to allow some system capacity to remain idle when work is available for processing. Let the input to the system be from a Poisson process with rate  $\lambda$ , and let the service times at each machine be exponentially distributed random variables with parameters  $u_1$  and  $u_2$ . Define the following system state probabilities:

$p_{k,0}$  = probability that there are  $k$  jobs waiting for service from machine 1, and machine 2 is idle ( $k=0,1, \dots C-1$ )

$p_{k,1}$  = probability that there are  $k$  jobs waiting for service from machine 1, and machine 2 is busy ( $k=0,1, \dots C-2$ )

$p_k$  = probability that there are a total of  $k$  jobs in the system waiting in the common ordered queue and waiting for or being serviced by machines 1 and 2 ( $k=C,C+1, \dots$ )

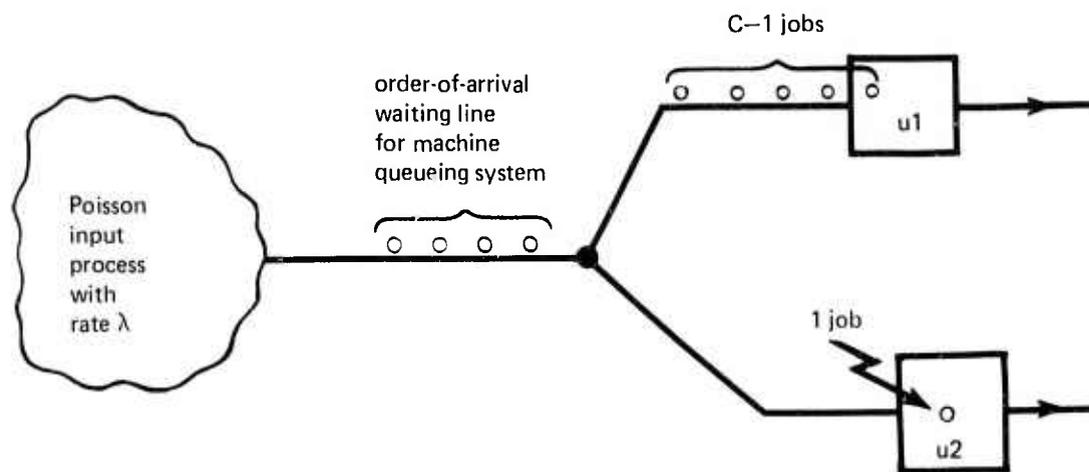


Figure 2 Scheduling Model

Using standard techniques for the analysis of exponential queueing systems (*e.g.*, as presented in the book by Saaty<sup>[5]</sup>) one may derive the following steady state recurrence equations for the state probabilities. These equations apply only when a steady state exists (*i.e.* when the input rate is less than the total processing capacity  $u_1 + u_2$ ).

$$(8) \quad (\lambda + u_1)p_{k,0} = u_1 p_{k+1,0} + \lambda p_{k-1,0} + u_2 p_{k,1} \\ k=1, \dots, C-2$$

$$(9) \quad (\lambda + u_1 + u_2)p_{k,1} = \lambda p_{k-1,1} + u_1 p_{k+1,1} \\ k=1, 2, \dots, C-2$$

$$(10) \quad (\lambda + u_1 + u_2)p_k = \lambda p_{k-1} + (u_1 + u_2)p_{k+1} \\ k=C+1, C+2, \dots$$

It is beyond the scope of this article to describe the solution of these equations in detail. However, one may use generating functions to reduce the infinite set of equations of relation (10) to one simple expression. Then all that is required is to solve  $2C$  linear equations by standard numerical techniques. Although a simple closed form for the result has not been found, the computations are straight-forward and easy to perform.

It is also beyond the scope of this article to describe in detail the many interesting results which may be obtained by solving these equations for various parameter settings. However, a few conclusions are easy to summarize. The expected value of the time spent in the system, both waiting for and receiving service, was the performance index used for the following comparisons. When the ratio of the rate of machine 1 to machine 2 is small (2 to 4), then the optimum value for  $C$  is also small (2 to 4) and the performance curve is relatively flat. Thus the improvement one could expect from implementing this type of policy in this type of situation is only a few percent. But as the ratio of the processing rates increases to ten for example, improvements of the order of twenty to twenty-five percent are possible by increasing  $C$  from 2 to six or seven. Around the optimum value of  $C$  the performance curve is again relatively flat. In this latter type of situation it is often better never to use machine 2 than to set  $C=2$ .

The model described in this section may be used to examine a number of theoretical scheduling questions. The results of this kind of analysis can help to formulate realistic policies. The performance of operational scheduling algorithms should be examined by simulations and measurements of prototype systems. However, the construction of these more expensive studies can be guided by insights gained from the analytic results.

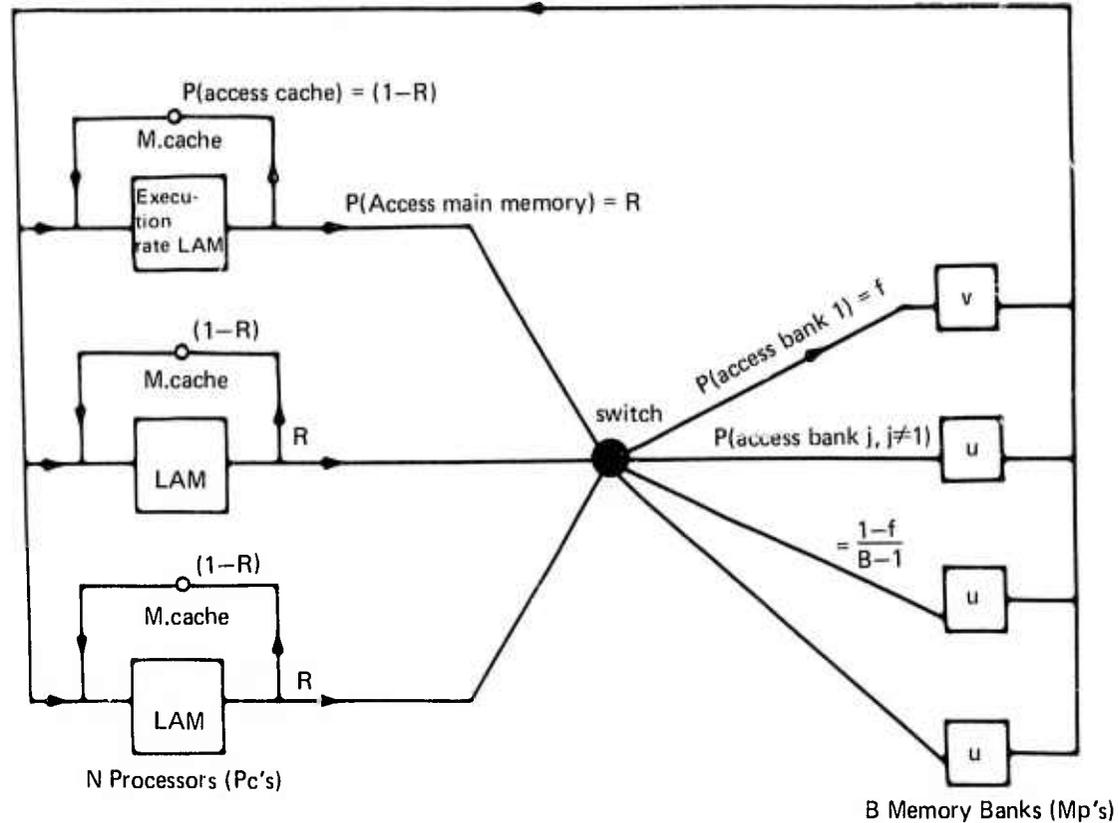


Figure 3 Memory Interference Model

### Memory Interference Model

One of the crucial problems in the design of a multiprocessing computing system is the interference which occurs when more than one processor requests information from the same shared memory (see Wulf [8]). Performance will be degraded in such circumstances due to queueing delays. Strecker [7] studied this problem and presented a number of models to approximate the effects of memory interference. Bhandarkar and Fuller [2] have recently surveyed techniques for analyzing this type of interference in multiprocessor systems. The analysis which follows differs from these other reports in a number of ways and represents an alternative framework for analytical study. The present model is based upon different assumptions than those used by Strecker, Bhandarkar and Fuller. It allows one to consider the effects of a cache memory for each of the processors, as well as the situation in which one of the memory modules has a different speed and probability of being ac-

cessed than all other modules. The model discussed in this section was presented in greater detail in a paper by McCredie [4].

Figure 3 illustrates the structure of the model. There are  $N$  processors each of which may access any one of  $B$  memory banks through an  $N$  by  $B$  cross point switch. Strecker [7] defines an abstraction called the "unit instruction", and shows how more complicated instructions may be synthesized from various combinations of unit instructions. Each unit instruction consists of one memory reference and a random interval of processor activity. The performance of a particular organization of memories and processors may be measured in terms of the mean unit execution rate (UER) which is the mean speed at which the configuration can execute unit instructions.

For the model of Figure 3, the time required to decode and execute a unit instruction will be an exponentially distributed interval having a mean of

1/LAM nanoseconds. At the end of this time the processor must access memory. The probability that one of the B banks of main memory will be referenced is R and the probability that the reference will be to the cache memory associated with each processor is (1-R). The cache will be assumed to have an access time that is much smaller than the processor delay and will be ignored. Since the probability of accessing the cache is independent of state information (such as how many accesses have already been directed to the cache) the number, X, of consecutive unit instructions the processor may execute before referencing main memory is geometrically distributed with mean 1/R.

The sum of a geometrically distributed number of exponential random variables is another exponentially distributed random variable. Thus, for each processor, the time from the completion of one reference to main memory until the next access to main memory is exponentially distributed with mean  $1/(R \cdot LAM)$ . If there is no cache memory for each processor, R is equal to unity. The value of R will decrease as the size of the cache increases.

Assume that the time that a module of main memory is blocked while an access is completed is an exponentially distributed random variable with mean  $1/v$  nanoseconds for all memory banks but the first which will have a mean of  $1/v$  nanoseconds. This exponential delay represents the total cycle time of main memory for the different classes of accesses as well as any switching delay required to link N processors with B memories. Define f to be the probability that a request to main memory will be to the first memory bank. Assume that requests to all other (B-1) modules are uniformly distributed and thus the probability that a request goes to any particular memory bank is:

(11) P (reference to module j when a reference is to main memory)

$$= \begin{cases} f, & j=1, 0 \leq f \leq 1 \\ \frac{1-f}{B-1} & j=2, \dots, B \end{cases}$$

A processor may not issue a request to any memory module until it has received and processed the information from the preceding memory access.

The assumptions stated in the previous paragraphs may be modified slightly to adjust the model to more realistic situations such as processor utilization of words from memory immediately after memory access and during memory rewrite. However, most of the assumptions are required to keep the mathematics reasonable. Using a powerful theorem originally presented by J. R. Jackson [3] one may solve this model to determine the mean unit execution rate, UER, as a function of all of the parameters defined above. The details of the application of this theorem to the present situation are contained in the previously referenced paper by McCredie. Although the equations are rather complicated, the solution may be evaluated by a straight-forward, fast algorithm which has a running time of a few milliseconds and is proportional to  $(N^2)$ . The algorithm is presented below in ALGOL to demonstrate that it is computationally quite simple.

31

```

REAL PROCEDURE UER(LAM, V,U,N,B,R,F);
REAL LAM,V,U,R,F;
INTEGER N,B;
COMMENT LAM,V,U, and N must be positive, R and
F must be probabilities and B, the number of mem-
ory banks, must be greater than 1;
BEGIN
  REAL REFPROB,DENOM, EM;
  REAL ARRAY W(0:N),T(0:N),A(0:N),P(0:N);
  INTEGER K,J;
  REFPROB:=(1.0-F)/(B-1);
  W(0):=T(0):=A(0):=DENOM:=1.0;
  EM:=0.0;
  FOR K:=1 STEP 1 UNTIL N DO
  BEGIN
    A(K):=A(K-1)*(B+K-2)/K;
    W(K):=W(K-1)*R*LAM*(N-K+1);
    T(K):=0.0;
    FOR J:=0 STEP 1 UNTIL K DO
      T(K):=T(K)+A(J)*(F/V)^(K-J)*
        (REFPROB/U)^J;
    DENOM:=DENOM+W(K)*T(K);
  END;
  P(0):=1.0/DENOM;
  FOR K:=1 STEP 1 UNTIL N DO
  BEGIN
    P(K):=T(K)*W(K)/DENOM;
    EM:=EM+K*P(K);
  END;
  UER:=(N-EM)*LAM;
END OF PROCEDURE UER;

```

Two of the necessary assumptions for this model were that both the processor execution times and the memory cycle times were exponentially distributed random variables. A uniformly distributed processing time and an approximately constant memory cycle time are closer approximations to the hardware performance data of multiprocessor configurations such as Carnegie-Mellon's C.mmp [8]. To check the effects of these assumptions we built a simple simulation of the system and compared the results for different parameters. The simulation curves were similar to the analytic results over a wide range of values.

### Summary

32

The primary goal of this article is to show that analytic models are valuable in the overall study of computer system performance. Even though they are usually simplified abstractions of actual systems and constitute only one dimension of the total space of available techniques, they do have advantages in certain areas. To capitalize on these advantages, systems analysts should be exposed to both the power and limitations of current analytic techniques, and researchers in the area should strive to communicate their results in more usable ways.

### References

1. Balachandran, V., J. W. McCredie, and O. I. Mikhail, "Models of the Job Allocation Problem in Computer Networks," *Proc. Seventh Annual IEEE Computer Society International Conference*, 1973, 211-214.
2. Bhandarkar, D. and S. Fuller, "A Survey of Techniques for Analyzing Memory Interference in Multiprocessor Computer Systems," Computer Science Dept., CMU, April 1973.
3. Jackson, J. R., "Jobshop: Like Queueing Systems," *Management Science*, Vol. 1, No. 1, 1963, 132-142.
4. McCredie, J. W., "Analytic Models as Aids in Multiprocessor Design," *Proc. Seventh Annual Princeton Conference on Information Science and Systems*, Princeton University, 1973.
5. Saaty, T. L., *Elements of Queueing Theory with Applications*, McGraw Hill Book Co., Inc., New York, N. Y., 1961.
6. Sherman, S., F. Baskett and J. C. Browne, "Trace-Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System," *CACM*, Vol. 15, No. 12, December 1972, 1063-1069.
7. Strecker, W. D., "An Analysis of the Instruction Execution Rate in Certain Computing Structures," Ph.D. Dissertation, Computer Science Dept., CMU, 1971.
8. Wulf, W. A., "C.mmp: A Multi-Mini-Processor," *Computer Science Research Review 1971-72*, Computer Science Dept., CMU, 37-69.

Preceding page blank

54

The image displays a grid of 20 columns and 20 rows of vertical bar patterns. Each cell in the grid contains a unique sequence of vertical bars of varying heights and widths, arranged in a regular, repeating pattern across the entire page. The patterns are dense and uniform, creating a textured, barcode-like appearance. The grid is centered on the page, with the text 'Preceding page blank' and the number '54' positioned at the top left and top center, respectively.

## Lessons from Perception for Chess-Playing Programs (*and Vice Versa*)

Herbert A. Simon

### Introduction

For nearly twenty years, artificial intelligence and cognitive psychology have maintained a close symbiotic relationship to each other. It has often been remarked that their cooperation stems from no logical necessity. That a human being and a computer are both able to perform a certain task implies nothing for the identity, or even similarity, of their respective performance processes. Each may have capabilities not shared by the other, and may build its performances on those peculiar capabilities rather than upon those they hold in common.

In spite of this logical possibility of total irrelevance of the one field for the other, during the last two decades there has been massive borrowing in both directions. Artificial intelligence programs capable of humanoid performance in particular task domains have provided valuable hypotheses about the processes that humans might use to perform these same tasks, and some of these hypotheses have subsequently been supported by evidence. Bobrow's [2] STUDENT program, for example, which translated story problems into algebraic equations, provided a model, later tested by Paige & Simon [11], for some of the human syntactic processes in performing that task.

Conversely, hypotheses and data about human performance have been important inputs to artificial intelligence efforts. The General Problem Solver, for example, received its early shape from analyses of human thinking-aloud protocols in a problem solving task [8].

The distance between AI and cognitive psychology has not been the same in all task domains. Until quite recently, for instance, AI research on theorem prov-

ing developed in directions quite different from those suggested by the study of human behavior in theorem proving tasks. There is little that is humanoid about resolution theorem proving.

In the domain of chess playing, the distance between AI and cognitive psychology has been neither so close as in the GPS example, nor so distant as in theorem proving. The early chess playing programs, in their reliance on brute force and machine speed, borrowed little from what was known of human chess playing processes [9]. The clear demonstration by their relatively weak levels of performance, that speed was not enough, produced a gradual movement toward incorporating into the programs some of the selective task-dependent heuristics that humans rely heavily upon in their chess playing. However, the strongest chess programs in existence today still rely heavily upon extensive rapid search, usually over thousands or tens of thousands of branches of the game tree [7].

I should like to describe here some efforts on the other side of the line — attempts to explore chess playing mechanisms that can explain human chess performance. These mechanisms may turn out to have important implications for the future of chess playing programs motivated by AI goals. Their own motivation, however, was largely psychological.

### MATER

The story begins with an examination of those kinds of chess positions where appropriate search will disclose a checkmating combination against which the opponent has no defense. We have good evidence that strong human players discover these checkmates in over-the-board play after exploring trees of positions having (generally) only a few dozen branches. Simon & Simon [15] hand-simulated a program that achieved this kind of performance, and which discovered checkmates as deep as eight moves (16 plies). This program was further developed and implemented by Baylor & Simon [11] in several versions of the MATER program.

MATER relied, first of all, on being able to detect attack and defense relations among pairs of pieces on the board, and to use this information to guide its search. On the offensive side (in its simplest version), it examined only checking moves — that is, moves attacking the king; but on the defensive side, it examined all legal replies. (This is essential in order to demonstrate that the checkmate cannot be escaped.) MATER's second important heuristic was to employ a search-and-scan strategy — at each stage it explored first that branch on the as-yet-unexplored portion of its game tree which allowed the opponent the fewest replies. The combination of its selectivity in considering attacking moves, and its priority ordering for attention to restricting moves gave it great power with modest amounts of search. In one of its most impressive performances — rediscovering the eight-move mate from a game of Edward Lasker against Thomas — the search tree grew to only 108 positions, and in most positions it was much smaller.

#### PERCEIVER

The claim that selective search could account for many aspects of human performance in chess was challenged by a number of psychologists who thought that perceptual processes, enabling a master player to see "at once" a whole multitude of meaningful relations in a position placed before him, held the key to skilled human chess playing. The Russian investigators, Tichomirov and Poznyanskaya [16], for example, recorded eye movements of a strong player for the first five seconds after he was shown a chess position with instructions to find the best move. During these five seconds, there were about twenty eye fixations, and almost all of these fixations were aimed at "important" squares of the board — those that a skilled player would regard as important for the position. The edges and corners of the board received almost no direct attention. Moreover, the sequence of fixations could not be correlated with any possible tree of moves. Saccadic movements of the eyes from one fixation to the next generally passed along lines of potential action between pairs of pieces. Thus, the eyes might move from one piece to another that attacked or defended it, or was attacked or defended by it.

To interpret the results of Tichomirov and Poznyanskaya, we need a few facts about the nature of vision. The eye has a central area, or fovea, about  $1^\circ$  in radius, of very high resolution, surrounded by a much wider peripheral area (about  $7^\circ$ ) in which familiar objects can usually be recognized, but no detailed information about them can be acquired. Since the angle between successive fixations is usually several degrees, the information that directs the saccadic movements must be acquired peripherally.

Simon & Barenfeld [12] set out to demonstrate that a serial processor could simulate the observed eye movement phenomena without requiring the assumption that large amounts of information can be acquired instantaneously and in parallel over the whole visual field. Their simulation program, PERCEIVER, used a stripped-down version of MATER (removing the executive routine that guided its search for mating combinations) to detect attack and defense relations between pairs of pieces. These relations, once detected, drove the eye movements.

More specifically, PERCEIVER assumed the eye to be fixated, initially, on some prominent piece in the position. The attack and defense relations between that piece and other pieces would be detected (presumably by a combination of foveal and peripheral vision), and the eye would then move to a new fixation at one of the squares so related to the point of previous fixation. Successive saccadic movements would carry the eyes around the board, but would tend to move them most often to those parts of the board where the network of chess relations among pieces was densest. Hence, PERCEIVER had many fixations on the "important" squares, and seldom strayed out to the corners of the board. In fact, its fixations and their sequence were indistinguishable from the human eye movements.

PERCEIVER showed that the basic perceptual processes required for the initial reconnaissance of a chess position were just like those that had already been incorporated in MATER for the search of the tree of moves. The amount of visual information to be acquired during the initial "perceptual" phase was not more than could be accounted for by this kind of scanning process. There was no evidence that the Gestalt of the position was seized "instantaneously".

### Reconstructing Chess Positions

Another chess perception phenomenon, first discovered in 1925 in Moscow [5], studied in detail by de Groot [4] in Amsterdam in the 1930's, and replicated again in our laboratory within the past couple of years, raised a different set of questions about how the mechanisms incorporated in MATER and PERCEIVER could account for the perceptual abilities of skilled chess players. This phenomenon was the remarkable ability of chess masters and grandmasters to reproduce a position from an actual game (not previously known to them) after they had seen it for only five or ten seconds.

In brief, the empirical findings are these: take a position (typically, with about 25 pieces on the board) from a game between strong players. Allow a master to examine it for five seconds. He will then be able, with about 80% accuracy, to replace the pieces correctly on the board. Let a weak player examine the same position for five seconds, and then try to reconstruct it. He will be able to place only six or seven pieces correctly on the board: about 25%.

But an equally surprising result is obtained if we now perform the same experiment with a board on which the pieces have been placed at random. Now the performance of the master falls to the level of the amateur, while the latter does slightly less well than before. That is to say, both master and amateur will now recall the positions of only about one quarter of the pieces, and the master will do no better than the weak player.

The first part of the experiment might seem to suggest that the chess master has unusual powers of visual imagery — a hypothesis about chess players that has been widely believed. But the second part of the experiment shows that these visual powers evaporate when the situations are different from those encountered in actual chess play. Evidently, the chess master's superior perceptive powers rest on special chess knowledge, and not on any unusual properties of his visual or imaging system.

This experiment seems at first to conflict with what we know about short-term memory [10]. There is a large body of evidence to show that one can hold only about a half dozen "chunks" of information in short-term memory. The information — up to about that amount — can be kept there indefinitely, but transferring it to long-term memory (to free up the short-term memory for other information) requires about five or more seconds for each chunk.

The term "chunk" in this theory is not quite as vague as might appear. A "chunk" is any unit of information that is already familiar to the subject, and which he can therefore recognize as an old friend. Thus, for a native speaker of a language, any common word is (at most) a single chunk, and even common idiomatic phrases (*e.g.* "make or break") may be chunks. Hence it is often possible to estimate in advance the number of chunks contained in a given stimulus — a string of words or numbers, say.

The findings in the chess perception experiments could be reconciled with the hypothesis of limited short-term memory if the chess master could recognize a chess position as a configuration of a half-dozen chunks of three or four pieces each, while the amateur recognized each piece as a separate chunk. The master's chunks would be configurations familiar to him from having seen the same arrangements of pieces in many previous positions.

This hypothesis has been explored by Chase & Simon [3] in a series of experiments in which they videotaped players reconstructing positions and timed the intervals between successive placements of pieces. Long intervals (over two seconds) were assumed to represent chunk boundaries; short intervals (less than two seconds) were assumed to be within-chunk intervals. The data gave support to several aspects of the hypothesis: the chunks so defined were in fact clusters of pieces of kinds that occur with high frequencies in games. Several kinds of evidence reinforced the plausibility of the two-second criterion for chunk boundaries.

The master's chunks were, in fact, larger than those of the weaker players — perhaps fifty per cent larger, on average. To that extent the short-term memory hypothesis was supported. However, contrary to the hypothesis, Chase & Simon found that the master held more chunks in memory (also by a margin of about fifty per cent) than did weaker players. Hence the master appeared to have a somewhat larger short-term memory capacity, measured in chunks, than did the others. This discrepancy between theory and data remains unexplained at present, and constitutes one of the important targets of our continuing research on this subject.

## MAPP

If we take, for the moment, an optimistic position, and assume that further investigation will reconcile the chunking hypothesis with the observed data, we still have to discover what kind of organization of processes would produce these phenomena. In the interest of parsimony, we don't want to invent explanations *ad hoc* for this purpose, but wish to limit ourselves to processes that are already known to exist from other psychological experiments.

38

The MAPP program was written by Simon & Gilmartin [14] to simulate the phenomena of the position-reconstruction experiment with the help of well substantiated mechanisms. MAPP can be regarded as the offspring of a marriage between the PERCEIVER program, used to simulate the eye movements, and EPAM, a venerable simulation program first devised by Feigenbaum to explain the main results from a whole range of standard rote-learning experiments [6].

Since the 19th century, psychologists have been studying the processes for memorizing syllables, either in the form of paired associates (stimulus = BYX, response = GOV) or in the form of series (CEV, DAR, CUJ, *et cetera*). Meaningfulness and familiarity of items have been shown to have major facilitative effects on learning (as much as a three-to-one increase in learning rate for meaningfulness); similarity of items, a deterrent effect. In a list, the items at the ends are generally learned with fewer errors than the middle items (serial position curve). For materials of a given kind, amount of learning is roughly proportional to total time. These are illustrative of some of the main findings from rote-learning experiments.

The EPAM program gives correct predictions — and in many cases quantitatively correct predictions — of the effects of these and other learning variables [13]. It is a reasonably well verified first approximation to a theory of rote learning. The MAPP program combines the main EPAM mechanisms with the mechanisms embodied in PERCEIVER in an endeavor to explain the chess position-recognition data. Since not all of the detail of the two parent programs is relevant to these data, MAPP incorporates somewhat stripped-down versions of EPAM and PERCEIVER.

MAPP has two main components: (1) a learning program and (2) a performance program. The learning program is exposed to many configurations of chess pieces (two to seven pieces each) of kinds that occur frequently in chess games. It grows, through this exposure, a large discrimination net that allows it to recognize these configurations when it encounters them again, and which stores the information needed to reconstruct each of them. The net-growing processes are essentially the processes of EPAM, and the configurations that become recognizable through this learning are the chunks to be held in short-term memory.

The performance program of MAPP scans a chess position that is presented to it, looking for salient pieces. It fixates on each salient piece, and uses the previously grown EPAM net to recognize the largest possible configuration of pieces around it. If it succeeds in recognizing a configuration, it stores in short-term memory the address in the EPAM net where the information about the configuration can be found. Up to six (or whatever number is specified by the parameter) such chunks can be stored simultaneously in short-term memory.

After short-term memory has been filled — or all salient pieces have been scanned, whichever occurs first — information about the board is removed, and MAPP is instructed to reconstruct the position. It takes the chunk addresses stored in short-term memory, recovers from the EPAM net the configurations corresponding to each of these chunks, and reconstructs the position (or as much of it as it has stored in memory) on the board.

How successful is MAPP in accounting for the superior ability of chess masters to reconstruct positions? The largest EPAM net that MAPP has grown thus far contains 1,144 configurations, of two to seven pieces each, selected more or less unsystematically from diagrams in standard chess works. We cannot be sure that these are the configurations that occur most frequently in chess games, but they certainly include a large fraction of the configurations of high frequency. Using this EPAM net, MAPP was able to replace 55% of the pieces in nine positions. In experiments with the same nine positions, a master replaced 81% of the pieces, while a Class A player replaced 49%.

Thus, given familiarity with 1,144 common configurations of pieces, MAPP performs twice as well as a beginner, a little better than a Class A player, and not nearly so well as a master. We can now ask how much the EPAM net would have to be expanded to bring the performance of MAPP up to master level. Since the net already contains the configurations that occur most frequently, each new configuration we add will be somewhat more rare than those already in the net — hence will make a less than proportional contribution to performance. We cannot estimate what that contribution will be without making some assumption about the frequency distribution of patterns. It is probably not unreasonable to assume that this distribution is much like the frequency distribution of words in natural language. The latter distribution is highly skewed, and is closely approximated by the so-called harmonic, or Zipf, distribution. In the harmonic distribution, when words are arranged by the frequency of their occurrence, the  $k$ th most frequent word occurs about  $1/k$  times as often as the most frequent word:  $f_k = (1/k) f_1$ . (Interestingly enough, when authors are ranked by the numbers of their publications, or cities by their populations, the distributions also conform approximately to the harmonic law.)

If we assume that the frequency distribution of patterns of chess pieces is also a harmonic distribution, then we can estimate the size of the EPAM net required to match the master performance. Taking the continuous approximation to  $f_i/i$ , the cumulative distribution is the log function:  $F_j = k \log_e i$ . From the MAPP simulation data,  $.55 = k \log_e 1144$ . Solving this equation, we find  $k = .078$ . Using this value of  $k$ , we now calculate the size of the net for a performance level of  $.81$  by  $\log_e N = .81/.078$ , whence  $N = 32,000$ .

How reasonable is it to assume that a chess master is familiar with 32,000 configurations of chess pieces? First, there are a number of other indirect ways for estimating the size of the net, all of which yield estimates of the same order of magnitude. Further, the estimate computed above is of about the same size as the natural language vocabulary of a college-educated adult. Such a person might be expected to have a recognition vocabulary in his native language of 25,000 to 100,000 words. When we consider that no one becomes a chess master without some years of intensive application to the game (grandmaster status is never achieved in less than a decade), the estimate becomes quite plausible; for, a chess master has spent about as many hours staring at chess positions as other educated adults have spent staring at the printed page.

There are other tests of MAPP besides the relation between its vocabulary of chess patterns and quantitative performance as on the recognition task. We can compare the nature of the chunks it recognizes with those recognized by human players in the same positions. The agreement is generally good. Hence, MAPP must be taken seriously as an explanation of the phenomena, and it would be desirable, as soon as possible, to test it with an EPAM net grown to 25,000 or 50,000 configurations. Since the smallest net grown for the experiment occupied about 100,000 words of PDP-10 memory, and since the time required to grow the net was more than an hour, the experiment will probably not be attempted until memories become somewhat larger, faster, and cheaper.

39

### Prospects

To understand the implications of the research on chess perception for the design of chess-playing programs, one other phenomenon should be discussed. It is well known that when strong chess players engage in rapid-transit games, taking only a few seconds for each move, their play is weaker, but only moderately weaker, than when they take a longer time for their moves. Masters and grandmasters can play dozens (or even hundreds) of simultaneous games against strong amateurs, and win almost all of them.

Drawing upon what has been learned about chess perception, we can provide a plausible, though as yet untested, explanation for such feats. Consider a production system programmed to play chess. The condition part of each production is a configuration of pieces on the board — just such a configuration as is stored in the EPAM net. The action part of the production is a move that is to be considered whenever that configuration occurs. The productions are arranged in priority order, with the most important at the head of the list. Thus, an attack on a queen will be noticed before an isolated pawn. The program then takes the first action whose condition is satisfied.

Such a program will undoubtedly not play good chess. It will certainly play rapid chess. What would have to be added to it to permit it to play plausible chess must be determined by experiment. Notice that for a "fair" test, a very large number of productions – tens of thousands – would have to be provided. But the real point at issue is not whether a program that is "nothing but" such a production system can be a strong chess player. Rather, the point at issue is whether any program that does not incorporate a range of chess knowledge like that imbedded in the production system can play good chess.

The experiments I have described bring us face-to-face again with one of the central issues of artificial intelligence: to what extent can intelligence be made general and independent of knowledge about particular subject-matter fields? To the extent that artificial intelligence is to be modelled on human intelligence, these experiments suggest that general mechanisms, however powerful and indispensable, are no complete substitute for the ability to recognize a very large number of quite specific features imbedded in complex situations: if the skilled man is an intelligent man, he is also a learned man.

#### Acknowledgements

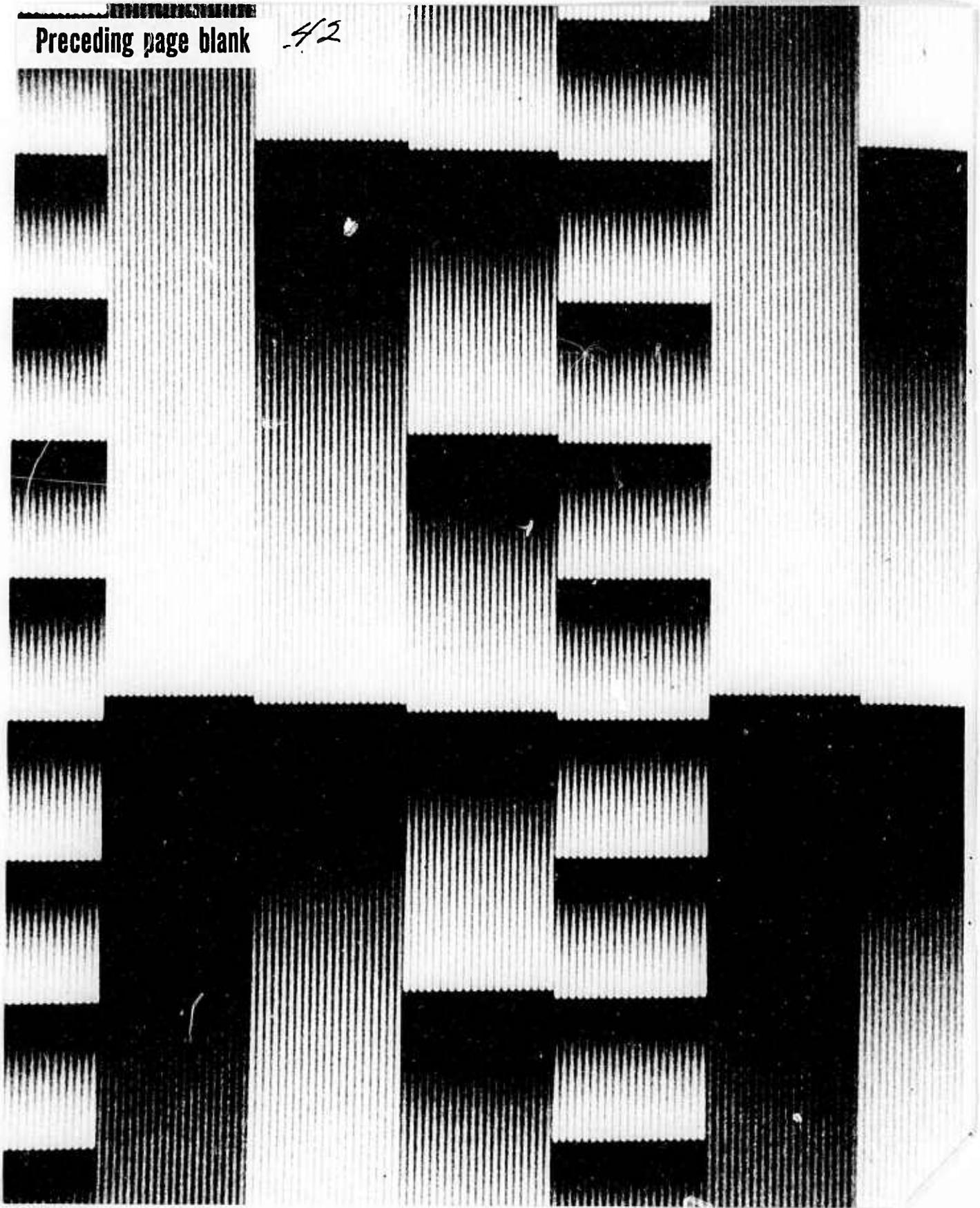
\* I am grateful to the colleagues who have collaborated with me over the years in this research on human and computer chess playing: Michael Barenfeld, George Baylor, William Chase, Kevin Gilmartin, Allen Newell, Clifford Shaw, and Peter Simon. This research has been supported in part by Public Health Service Grant MH-07722 from the National Institute of Mental Health, and by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0107), which is monitored by the Air Force Office of Scientific Research.

#### References

1. Baylor, G. W., Jr., "Report on a Mating Combinations Program," Santa Monica, Ca.: SP-2150, System Development Corporation, 1965; Baylor, G. W., Jr. and H. A. Simon, "A Chess Mating Combinations Program," *Proc. SJCC*, Atlantic City, N. J., 1966, 431-447.
2. Bobrow, D. C., "Natural Language Input for a Computer Problem-Solving System," in M. Minsky (ed.), *Semantic Information Processing*, MIT Press, Cambridge, Mass., 1968.
3. Chase, W. G. and H. A. Simon, "Perception in Chess," *Cognitive Psychology*, Vol. 4, January 1973, 55-81.
4. de Groot, A. D., *Het denken van den Schaker*, translated as *Thought and Choice in Chess*, Mouton & Company, the Hague, 1965.
5. Djakow, I. N., N. W. Petrowski and P. A. Rudik, *Psychologie des Schachspiel*, Walter de Gruyter, Berlin, 1927.
6. Feigenbaum, E. A., "The Simulation of Verbal Learning Behavior," *Proc. Western Joint Computer Conference*, Los Angeles, Ca., 1961, 121-132.
7. See Monroe Newborn (ed.), "A Review of the 1st and 2nd United States Computer Chess Championships and a Review of Recent Developments in Computer Chess," published for *ACM 25th Annual Conference*, August 1972.
8. Newell, A. and H. A. Simon, *Human Problem Solving, Part 3*, Prentice-Hall, Englewood Cliffs, N. J., 1972.
9. See the survey of these early systems in Newell & Simon, Chapter 11.
10. Norman, D. A., *Memory and Attention*, Wiley, New York, 1969.
11. Paige, J. M. and H. A. Simon, "Cognitive Processes in Solving Algebra Word Problems," in B. Kleinmuntz (ed.), *Problem Solving*, Wiley, New York, 1966.
12. Simon, H. A. and M. Barenfeld, "Information-Processing Analysis of Perceptual Processes in Problem Solving," *Psychological Review*, Vol. 76, 1969, 473-483.
13. Simon, H. A. and E. A. Feigenbaum, "An Information-Processing Theory of Some Effects of Similarity, Familiarization, and Meaningfulness in Verbal Learning," *Journal of Verbal Learning and Verbal Behavior*, Vol. 3, 1964, 385-396; Gregg, L. W. and H. A. Simon, "An Information-Processing Explanation of One-Trial and Incremental Learning," *Journal of Verbal Learning and Verbal Behavior*, Vol. 6, 1967, 780-787.
14. Simon, H. A. and K. Gilmartin, "A Simulation of Memory for Chess Positions," *Cognitive Psychology*, Vol. 4, 1973, 55-81.
15. Simon, H. A. and P. A. Simon, "Trial and Error Search in Solving Difficult Problems: Evidence from the Game of Chess," *Behavioral Science*, Vol. 7, 1962, 425-429.
16. Tichomirov, O. K. and E. D. Poznyanskaya, "An Investigation of Visual Search as a Means of Analyzing Heuristics," *Soviet Psychology*, Vol. 5, 1966, 2-15, [Translated from *Voprosy Psikhologii*, Vol. 2, No. 4, 39-53.]

PRECEDING PAGE BLANK

42



## Faculty

43

C. Gordon Bell  
Professor of Computer Science and  
Electrical Engineering  
S.B., Massachusetts Institute of  
Technology (1956)  
S.M., Massachusetts Institute of  
Technology (1957)  
Carnegie, 1966: Computers and Computer  
Networks

Jack R. Buchanan  
Assistant Professor of Computer Science and  
Industrial Administration  
B.S., University of Utah (1965)  
M.A., University of Utah (1967)  
Ph.D., Stanford University (1973)  
Carnegie, 1972: Complex Information  
Processing

Robert N. Chanon  
Instructor  
B.S., Carnegie-Mellon University (1967)  
Carnegie, 1968: Software Engineering

Charles M. Eastman  
Associate Professor of Architecture,  
Computer Science, and Urban and  
Public Affairs  
B.Arch., University of California  
at Berkeley (1964)  
M.Arch., University of California  
at Berkeley (1966)  
Carnegie, 1967: Computer-Aided Design,  
Simulation of Human Problem-Solving,  
and Urban Models

Lee D. Erman  
Research Associate  
B.S., University of Michigan (1966)  
M.S., Stanford University (1968)  
Carnegie, 1970: Artificial Intelligence

Samuel H. Fuller  
Assistant Professor of Computer Science and  
Electrical Engineering  
B.S.E., University of Michigan (1968)  
M.S., Stanford University (1969)  
Ph.D., Stanford University (1972)  
Carnegie, 1972: Performance Evaluation,  
Measurement of Computer Systems,  
Computer Architecture

A. Nico Habermann  
Associate Professor of Computer Science  
B.S., Free University, Amsterdam (1953)  
M.S., Free University, Amsterdam (1957)  
Ph.D., Technological University, Eindhoven,  
The Netherlands (1967)  
Carnegie, 1968: Operating Systems and  
Programming Languages

Michael J. Knudsen  
Research Associate  
B.S., Pennsylvania State University (1966)  
M.S., Massachusetts Institute of  
Technology (1968)  
Ph.D., Carnegie-Mellon University (1972)  
Carnegie, 1973: Computer Systems Architecture  
and Description Languages

H. T. Kung  
 Research Associate  
 B.S., National Tsing Hua University,  
 Taiwan (1968)  
 M.S., The University of New Mexico (1970)  
 Carnegie, 1973: Computational Complexity  
 and Numerical Mathematics

Victor R. Lesser  
 Research Associate  
 A.B., Cornell University (1966)  
 M.S., Stanford University (1970)  
 Ph.D., Stanford University (1972)  
 Carnegie, 1972: Design of Speech Understanding,  
 Raster Display Devices, Computer  
 Architecture (Micro-programming,  
 multiprocessor systems)

Donald W. Loveland  
 Associate Professor of Computer Science and  
 Mathematics  
 A.B., Oberlin College (1956)  
 S.M., Massachusetts Institute of  
 Technology (1958)  
 Ph.D., New York University (1964)  
 Carnegie, 1967: Logic (Recursive Function  
 Theory), Mechanical Theorem Proving,  
 and Computational Complexity

Philip H. Mason  
 Instructor  
 B.S., Carnegie-Mellon University (1967)  
 Carnegie, 1969: Systems Programming and  
 Parallel Processing Systems

Nelson L. Max  
 Visiting Assistant Professor of Computer  
 Science and Mathematics  
 B.S., Johns Hopkins University (1960)  
 M.S., Harvard University (1963)  
 Ph.D., Harvard University (1967)  
 Carnegie, 1972: Computer Animation

John W. McCredie  
 Assistant Professor of Computer Science  
 B.E., Yale University (1962)  
 M.S.E.E., Yale University (1964)  
 Ph.D., Carnegie-Mellon University (1972)  
 Carnegie, 1968: Analytical Modeling,  
 Simulation, and System Performance  
 Evaluation

James Moore  
 Research Associate  
 B.S., Massachusetts Institute of  
 Technology (1964)  
 Ph.D., Carnegie-Mellon University (1971)  
 Carnegie, 1971: Artificial Intelligence  
 and Semantic Nets

Richard B. Neely  
 Research Associate  
 B.A., University of Oregon (1966)  
 M.S., Stanford University (1968)  
 Carnegie, 1970: Artificial Intelligence

Allen Newell  
 University Professor  
 B.S., Stanford University (1949)  
 Ph.D., Carnegie Institute of Technology (1957)  
 Carnegie, 1961: Artificial Intelligence,  
 Psychology of Human Thinking, Programming  
 Systems, and Computer Structures

David L. Parnas  
 Associate Professor of Computer Science  
 B.S., Carnegie Institute of Technology (1961)  
 M.S., Carnegie Institute of Technology (1964)  
 Ph.D., Carnegie Institute of Technology (1965)  
 Carnegie, 1966: Software Engineering, Computer  
 Languages, and Computer System Design

Thomas A. Porsching  
 Senior Lecturer  
 B.S., Carnegie Institute of Technology (1957)  
 M.S., Carnegie Institute of Technology (1958)  
 Ph.D., Carnegie Institute of Technology (1964)  
 Carnegie, 1972: Numerical Solution of  
 Partial Differential Equations

Srinivasan Ramani  
Visiting Research Associate  
B.E., University of Madras (1962)  
M.Tech., Indian Institute of Technology (1964)  
Ph.D., Indian Institute of Technology (1969)  
Carnegie, 1971: Problem-Solving by Machine  
and Natural Language Systems

D. Raj Reddy  
Associate Professor of Computer Science  
B.E., University of Madras (1958)  
M.Tech., University of New South  
Wales (1961)  
M.S., Stanford University (1964)  
Ph.D., Stanford University (1966)  
Carnegie, 1969: Artificial Intelligence,  
Computer Graphics, and Man-Machine  
Communication

Ronald M. Rutledge  
Assistant Professor of Computer Science,  
Chief Officer of the Operations Division,  
and Director of the Computation Center  
S.B., University of Georgia (1957)  
S.M., University of Georgia (1960)  
Ph.D., University of Tennessee (1964)  
Carnegie, 1968: Computer Science  
Management and Evaluation of  
Operating Systems

Mario Schkolnick  
Assistant Professor of Computer Science  
Electrical Engineer, University of  
Chile (1965)  
M.S., University of California (1967)  
Ph.D., University of California at  
Berkeley (1969)  
Carnegie, 1973: Formal Languages and  
Automata Theory

Mary Shaw  
Assistant Professor of Computer Science  
B.A., Rice University (1965)  
Ph.D., Carnegie-Mellon University (1972)  
Carnegie, 1971: Programming Systems and  
Concrete Computational Complexity

Linda Shockey  
Visiting Research Associate  
B.S., Ohio State University (1967)  
Ph.D., Ohio State University (1973)  
Carnegie, 1972: Linguistics and  
Automatic Speech Recognition

Daniel P. Siewiorek  
Assistant Professor of Computer Science and  
Electrical Engineering  
B.S., University of Michigan (1968)  
M.S., Stanford University (1969)  
Ph.D., Stanford University (1972)  
Carnegie, 1972: Computer Architecture,  
Digital System Design, Reliability  
Modeling, Testing, and Fault  
Tolerant Computer Design

Herbert A. Simon  
Richard King Mellon Professor of Computer  
Science and Psychology  
A.B., University of Chicago (1936)  
Ph.D., University of Chicago (1943)  
D.Sc.(Hon.), Case Institute of  
Technology (1963)  
D.Sc.(Hon.), Yale University (1963)  
LL.D.(Hon.), University of Chicago (1964)  
Fil.D.(Hon.), University of Lund,  
Sweden (1968)  
LL.D. (Hon.), McGill University (1970)  
Carnegie, 1949: Computer Simulation of  
Cognitive Processes, Artificial  
Intelligence, and Management Science

Franco Sirovich  
 Visiting Research Associate  
 M.S., Politecnico of Milano (1967)  
 Ph.D., Institudi Elaborazione della  
 Informazione, Pisa (1969)  
 Carnegie, 1971: Artificial Intelligence and  
 Machine Learning Techniques

Gilbert W. Stewart  
 Associate Professor of Computer Science and  
 Mathematics  
 B.A., University of Tennessee (1962)  
 Ph.D., University of Tennessee (1968)  
 Carnegie, 1972: Numerical Mathematics

Joseph F. Traub  
 Professor of Computer Science and  
 Mathematics, and Head of the  
 Department of Computer Science  
 B.S., City College of New York (1954)  
 Ph.D., Columbia University (1959)  
 Carnegie, 1971: Numerical Mathematics and  
 Computational Complexity

William A. Wulf  
 Associate Professor of Computer Science  
 B.S., University of Illinois (1961)  
 M.S.E.E., University of Illinois (1963)  
 D.Sc., University of Virginia (1968)  
 Carnegie, 1968: Programming Systems:  
 Compiler Optimization, Operating  
 Systems, Systems Programming  
 Languages, and Multiprocessor  
 Systems

## Departmental Staff

### *Engineering*

William Broadley—Manager of Engineering Design  
 Paul Buck—AI Video Engineer  
 Paolo Coraluppi—Research Engineer  
 Michael Doreau—Digital Design Engineer  
 Tim Kirby—Staff Engineer  
 Stan Kriz—Engineer  
 Richard A. Lang—Technician  
 Paul Newbury—Engineering Lab Property Manager  
 John M. Powell—Engineer  
 Charles Rapach—Staff Engineer  
 Brian Rosen—Staff Engineer  
 Kenneth Stupak—Technician  
 James Teter—Manager of Engineering Production  
 William Vogler—Draftsman

### *Office Staff*

Nancy Barron—Secretary to Department Head  
 Mary Caldwell—Departmental Secretary  
 Beverly Howell—Documentation Librarian  
 Dorothy Josephson—Faculty Secretary  
 Mercedes Kostkas—Departmental Secretary  
 Carol Kustra—Departmental Secretary  
 Beryl O'Connell—Secretary to Business Manager  
 Mildred Sisko—Secretary to Prof. Newell  
 Paul Stockhausen—Business Manager

### *Programming and Operations*

Barbara Anderson—Operator/Programmer  
 Diana Bajzek—Programmer  
 Donn J. Bihary—Programmer  
 Christopher Cooper—Programmer  
 Gregory S. Gill—Visiting Research Assistant  
 Donald McCracken—Research Programmer  
 Susan Nist—Operator  
 Charles Pierson—Research Programmer  
 George Robertson—Research Programmer  
 Harold Van Zoeren—Senior Research Programmer  
 Howard Wactlar—Manager of Programming  
 Carl L. Wears—Operator/Programmer  
 Eric Werme—Programmer  
 Paula Wilson—Operator

## Graduate Students

Durga Agarwal  
B.E., Birla Institute of Technology and  
Science (1969)

Electronics  
M.Tech., Indian Institute of Technology (1970)  
Computer Science

Jerry Apperson  
B.A., University of Virginia (1965)  
Mathematics

Gideon Ariely  
B.A., Hebrew University (1969)  
Mathematics, Philosophy, Computer Science

Marshall A. Atlas  
B.S., Rensselaer Polytechnic Institute (1967)  
Mathematics  
M.S., University of Illinois (1968)  
Mathematics

Birol Aygun  
B.S.M.E., Newark College of Engineering (1965)  
M.S., Columbia University (1968)  
Mathematical Methods in Engineering and  
Operations Research

James Baker  
A.B., Princeton University (1967)  
Mathematics

Janet M. Baker  
B.S., Tufts University (1969)  
Biology and German

Mario Barbacci  
B.S., U.N.I., Lima, Peru (1966)  
Electrical Engineering  
Engineer, U.N.I., Lima, Peru (1968)  
Electrical Engineering

Madeline Bauer  
A.B., Cornell University (1968)  
Mathematics  
M.A., University of Michigan (1970)  
Computing and Communications Sciences

Hans Berliner  
B.A., George Washington University (1954)  
Psychology

Sushil Bhatia  
B.Tech., Indian Institute of Technology (1966)  
Electrical Engineering  
M.S., Carnegie-Mellon University (1969)  
Electrical Engineering

Hsiau-Chung Chang  
B.S., National Taiwan University (1971)  
Physics

Robert Chanon  
B.S., Carnegie Institute of Technology (1967)  
Mathematics

Robert Chen  
B.E.E., Rensselaer Polytechnic Institute (1966)  
Electrical Engineering  
S.M., Massachusetts Institute of Technology (1968)  
Electrical Engineering

Douglas W. Clark  
B.S., Yale University (1972)  
Engineering and Applied Science

Ellis Cohen  
B.S., Drexel Institute of Technology (1970)  
Mathematics

Lee W. Coopridger  
B.A., Oberlin College (1969)  
Mathematics

William M. Corwin  
B.S., Carnegie-Mellon University (1972)  
Physics

John Dills  
B.S., Clarkson College (1968)  
Mathematics

David C. Eklund  
B.A., Harvard University (1968)  
Applied Mathematics

Roger Fajman  
B.S., California Institute of Technology (1967)  
Mathematics  
M.S., Stanford University (1969)  
Computer Science

Arthur Farley  
B.S., Rensselaer Polytechnic Institute (1968)  
Mathematics

Richard Fennell  
B.S., Rensselaer Polytechnic Institute (1969)  
Physics

Lawrence E. Flon  
B.S., SUNY at Stony Brook (1972)  
Physics

Charles L. Forgy  
B.S., University of Texas at Arlington (1972)  
Mathematics

John G. Gaschnig  
B.S.E.E., Massachusetts Institute of  
Technology (1972)  
Computer Science

Susan Gerhart  
B.A., Ohio Wesleyan University (1965)  
Mathematics  
M.S., University of Michigan (1967)  
Communication Sciences

Charles Geschke  
A.B., Xavier University (1962)  
Latin  
M.S., Xavier University (1963)  
Mathematics

James Gillogly  
B.A., UCLA (1967)  
Mathematics  
M.S., University of Wisconsin (1970)  
Computer Science

Henry Goldberg  
S.B., Massachusetts Institute of  
Technology (1968)  
Mathematics

Gilbert Hansen  
B.S., Case Institute of Technology (1962)  
Physics  
M.S., Case Institute of Technology (1964)  
Computer Science

Don Heller  
B.S., Carnegie-Mellon University (1971)  
Mathematics

George M. Hicks  
B.A., David Lipscomb College (1971)  
Physics

Teruo Hikita  
B.S., University of Tokyo (1970)  
Mathematics  
M.S., University of Tokyo (1972)  
Computer Science

Steven O. Hobbs  
A.B., Dartmouth College (1969)  
Mathematics  
M.A., University of Michigan (1972)  
Mathematics (Computer Science Option)

Wing-Hing Huen  
B.S., University of Hong Kong (1966)  
Physics  
M.S., University of Alberta (1969)  
Computer Science

David R. Jefferson  
B.S., Yale University (1970)  
Mathematics

Richard Johnson  
B.E., Vanderbilt University (1970)  
Electrical Engineering

Anita Jones  
B.A., Rice University (1964)  
Mathematics  
M.A., University of Texas (1966)  
English

Edwin B. Kaehler  
B.S., Stanford University (1972)  
Physics

Philip Karlton  
 B.A., University of California,  
 Santa Barbara (1971)  
 Mathematics

Philip Koltun  
 B.S., University of Illinois (1971)  
 Mathematics and Computer Science

Sai-Ming Lee  
 B.A., University of California at  
 Berkeley (1970)  
 Mathematics and Computer Science

Roy Levin  
 B.S., Yale University (1970)  
 Mathematics

Robert Lieberman  
 B.S., SUNY at Stony Brook (1968)  
 Mathematics

Richard Lipton  
 B.S., Case Western Reserve (1968)  
 Mathematics

Bruce Lowerre  
 B.S., Case Institute of Technology (1965)  
 Chemistry  
 B.S., Case Western Reserve (1970)  
 Mathematics

Amund Lunde  
 M.Sc., University of Oslo (1966)  
 Mathematics

William Mann  
 B.S., Lehigh University (1956)  
 Electrical Engineering  
 M.E.A., George Washington University (1964)  
 Engineering Administration

Madhav Marathe  
 B.S., University of Bombay (1971)  
 Physics  
 M.S., Indian Institute of Technology,  
 Kanpur (1972)  
 Physics

Thomas Moran  
 B.Arch., University of Detroit (1965)  
 Architecture  
 M.S., Cornell University (1967)  
 Architectural Structures

Joseph Newcomer  
 B.A., St. Vincent College (1967)  
 Mathematics

John D. Oakley  
 B.S., Harvey Mudd College (1970)  
 Physics  
 M.S., University of Wisconsin (1972)  
 Computer Science

Ronald Ohlander  
 B.S., St. Mary's College (1962)  
 Psychology

Frederick Pollack  
 B.S., University of Florida (1970)  
 Mathematics

Keith Price  
 B.S., Massachusetts Institute of  
 Technology (1971)  
 Electrical Engineering

William Price  
 B.A., Lehigh University (1969)  
 Mathematics

Robert Ramey  
 B.S., Northwestern University (1954)  
 Chemistry  
 M.A., Northwestern University (1956)  
 Biochemistry  
 M.S., Northwestern University (1957)  
 Mathematics

Satish Rege  
 B.Tech., Indian Institute of Technology,  
 Bombay (1966)  
 Electrical Engineering  
 M.S., University of Pittsburgh (1969)  
 Electrical Engineering

Elaine Rich  
A.B., Brown University (1972)  
Formal Language Theory

Lawrence Robinson  
B.S., Yale University (1971)  
Engineering and Applied Science

Michael Rychener  
A.B., Oberlin College (1969)  
Mathematics  
M.S., Stanford University (1971)  
Computer Science

Steven Saunders  
S.B., Massachusetts Institute of  
Technology (1972)  
Computer Science

Steven J. Schlesinger  
B.S., Cornell University (1968)  
Mathematics

Edward Schneider  
B.S., Carnegie-Mellon University (1970)  
Mathematics

Ross Scroggs  
B.S., North Carolina State University  
at Raleigh (1971)  
Mathematics and Computer Science

Richard Smith  
B.S., Houghton College (1971)  
Physics and Mathematics

Larry Snyder  
B.A., University of Iowa (1968)  
Mathematics

David K. Stevenson  
B.A., Wesleyan University (1969)  
English and Mathematics  
M.A., University of Oregon (1972)  
Mathematics

Mark Stickel  
B.S., University of Washington (1969)  
Mathematics  
M.S., University of Washington (1971)  
Computer Science

Richard J. Swan  
B.A., University of Essex (1972)  
Computing Science

Ray Teitelbaum  
S.B., Massachusetts Institute of  
Technology (1964)  
Mathematics

Ronald Tugender  
B.S., SUNY at Stony Brook (1971)  
Mathematics

Narayanan Vasudevan  
B.S., Engineering College, Madras (1966)  
Electrical Engineering  
M.Tech., Indian Institute of Technology,  
Bombay (1969)  
Electrical Engineering

Charles Weinstock  
B.S., Carnegie-Mellon University (1970)  
Mathematics

David S. Wile  
Sc.B., Brown University (1967)  
Applied Mathematics

## Publications

July 1, 1972 to June 30, 1973

These publications are given in alphabetical order according to the name of the first author listed for each publication. In cases of multiple authorship where more than one author is in the Computer Science Department, a cross reference is made to that first listing under the name of each departmental author.

No cross-references are made for non-departmental authors.

- Aygun, B. O., "Environments for Monitoring and Dynamic Analysis of Execution," *Proc. Symposium on the Simulation of Computer Systems*, National Bureau of Standards, Gaithersburg, Md., June 1973.
- Balachandran, V., J. W. McCredie and O. I. Mikhail, "Models of the Job Allocation Problem in Computer Networks," *Proc. COMPCON 73*, New York, N. Y., 1973, 211-214.
- Barbacci, M., C. G. Bell and A. Newell, "ISP: A Language to Describe Instruction Sets and Other Register Transfer Systems," *COMPCON 72*, San Francisco, Ca., September 1972.
- Barbacci, M., C. G. Bell and D. P. Siewiorek, "ISP: A Notation to Describe a Computer's Instruction Set," *Computer*, Vol. 6, No. 3, March 1973, 22-24.
- For other references by M. Barbacci, see D. P. Siewiorek, —, and C. G. Bell.
- Bauer, M. J. and J. W. McCredie, "AMS: A Software Monitor for Performance Evaluation and System Control," *First Annual SIGME Symposium on Measurement and Evaluation*, Palo Alto, Ca., February 1973, 147-160.
- Bell, C. G., L. Gale and C. Kaman, "Some Effects of LSI on Minicomputers," *NEREM 72*, August 1972.
- Bell, C. G., J. Grason and A. Newell, *Designing Computers and Digital Systems Using PDP-16 Register Transfer Modules*, Digital Press, Maynard, Mass., September 1972.
- Bell, C. G. and P. Freeman, "C.ai—A Computer Architecture for AI Research," *Proc. FJCC*, Anaheim, Ca., 1972, 779-790.
- Bell, C. G. and M. Knudsen, "PMS: A Notation to Describe Computer Structures," *Proc. COMPCON 72*, San Francisco, Ca., September 1972, 227-230.
- Bell, C. G., R. C. Chen, S. H. Fuller, J. Grason, S. Rege and D. P. Siewiorek, "The Architecture and Applications of Computer Modules: A Set of Components for Digital Systems Design," *Proc. COMPCON 73*, New York, N. Y., February 1973.
- For other references by C. G. Bell, see D. P. Bhandarkar *et al.*, and D. P. Siewiorek, —, and J. Grason, and D. P. Siewiorek, M. Barbacci and —, and W. A. Wulf and —.
- Bhandarkar, D. P., C. G. Bell, P. Goel and J. Grason, "A Simulator for Register Transfer Modules," *Proc. Fourth Annual Pittsburgh Conference on Modeling and Simulation*, Pittsburgh, Pa., April 1973.
- For references by D. Bihary, see D. R. Reddy *et al.*
- For references by W. Broadley, see D. R. Reddy *et al.*
- Brooks, C., L. D. Erman and R. B. Neely, "JABBER-WOCKY: A Semiautomated System for the Transcription of Verbal Protocols," *Behavioral Research Methods and Instrumentation*, May 1973.
- Chase, W. G. and H. A. Simon, "Perception in Chess," *Cognitive Psychology*, Vol. 4, January 1973, 55-81.
- For references by R. C. Chen, see C. G. Bell *et al.*
- Cohen, E. S., "Symmetric Multi-Mini Processors: A Better Way To Go," *Computer Decisions*, January 1973.
- Computer Science Departmental Review Committee, "The Computer Science Ph.D. Program at CMU," *SIGSCE Bulletin*, Vol. 5, No. 2, June 1973, 33-40.

- Eastman, C., "Automated Space Planning," *Artificial Intelligence*, April 1973, 41-64.
- Eastman, C., "Requirements for Man-Machine Collaboration in Design," to appear in *Environmental Design Research*, W. Preiser (ed.), Dowden, Hutchinson, and Ross, Inc., Stroudsburg, Pa.
- Eastman, C., "Automated Space Planning and a Theory of Design," *Proc. International Computing Symposium*, Venice, Italy, April 1972.
- Erman, L. D., D. R. Reddy and R. B. Neely, "The CMU Hearsay Speech Understanding System," invited Paper, *Seminar on Artificial Intelligence*, American Nuclear Society, November 1972.
- For other references by L. D. Erman, see C. Brooks, —, and R. B. Neely, and D. R. Reddy, —, and R. B. Neely, and D. R. Reddy *et al.*
- Fajman, R. and J. Borgelt, "WYLBUR: An Interactive Text Editing and Remote Job Entry System," *CACM*, May 1973.
- Fleisig, S., D. Loveland, D. Smiley and D. Yarmush, "An Implementation of the Model Elimination Proof Procedure," to appear in *JACM*.
- Fuller, S. H. and F. Baskett, "An Analysis of Drum Storage Units," to appear in *JACM*.
- Fuller, S. H., "An Optimal Drum Scheduling Algorithm," *IEEE Trans. on Computers C-21*, November 11, 1972, 1153-1165.
- Fuller, S. H., "Performance of an I/O Channel with Multiple Paging Drums," *ACM SIGMET Symposium on Measurement and Performance Evaluation*, Palo Alto, Ca., March 1973, 13-21.
- Fuller, S. H., R. J. Swan and W. A. Wulf, "The Instrumentation of C.mmp: A Multi-Mini-Processor," *Proc. COMPCON 73*, New York, N. Y., March 1973, 173-176.
- For other references by S. H. Fuller, see C. G. Bell *et al.*, and H. S. Stone and —.
- For references by C. M. Geschke, see D. S. Wile and —.
- Gillogly, J. J., "The Technology Chess Program," *Artificial Intelligence*, March 1973, 145-163.
- Habermann, A. N., "Integrated Designer," presented at the *SIGPLAN/SIGOPS Interface Meeting*, Harri-man, N. Y., April 1973.
- For other references by A. N. Habermann, see D. L. Parnas and —.
- Hansen, P. B., P. J. Courtois, F. Heymans and D. L. Parnas, Comments on "A Comparison of Two Synchronizing Concepts," *Acta Informatica 1*, 1972, 375-376.
- Johnsson, K. O. and R. K. Johnsson, "Stopping Engine Run-on," *Popular Electronics*, Vol. 3, No. 3, March 1973.
- For references by R. K. Johnsson, See R. O. Johnsson and —, and D. R. Reddy *et al.*
- Klein, S., J. D. Oakley, D. J. Suurballe and R. A. Ziesemer, "A Program for Generating Reports on the Status and History of Stochastically Modifiable Semantic Models of Arbitrary Universes," *Statistical Methods In Linguistics*, August 1972, 64-93.
- For references by M. Knudsen, see C. G. Bell and —.
- Kung, H. T., "A Bound on the Multiplication Efficiency of Iteration," *Proc. Fourth Annual ACM Symposium on Theory of Computing*, Denver, Colo., May 1972, also in *Journal of Computer and System Sciences*, June 1973.
- Kung, H. T., "The Computational Complexity of Algebraic Numbers," *Proc. Fifth Annual ACM Symposium on Theory of Computing*, March 1973.
- Lesser, V. R., "The Design of an Emulator for a Parallel Machine Language," presented at *ACM SIGPLAN/SIGMICRO Interface Meeting*, Harri-man, N. Y., May 1973.

Lesser, V. R., "A Dynamically Reconfigurable Multiple Microprocessor," presented at International Workshop on Computer Architecture, Grenoble, France, June 1973.

For references by D. Loveland, see S. Flesig *et al.*

McCredie, J. W., "A Tandem Queueing Model of a Time-Shared Computer System," *Proc. of the ACM*, Vol. 2, August 1972, 991-1000.

McCredie, J. W., "Analytic Models as Aids in Multi-processor Design," *Proc. Seventh Princeton Conference on Information Sciences and Systems*, Princeton, N. J., 1973.

For other references by J. W. McCredie, see V. Balachandran, O. I. Mikhail and —, and M. Bauer and —.

Moran, T. P., "The Cognitive Structure of Spatial Knowledge," *Proc. Fourth Environmental Design Research Association Conference*, Vol. 2, Dowden, Hutchinson & Ross, Inc., Stroudsburg, Pa., 1973.

For references by R. B. Neely, see C. Brooks, L. D. Erman, and —, and L. D. Erman, D. R. Reddy and —, and D. R. Reddy, L. D. Erman and —.

For references by J. Newcomer, see D. R. Reddy *et al.*

Newell, A., "A Note on Process-Structure Distinctions in Developmental Psychology," in Sylvia Farnham-Diggory (ed.), *Information Processing in Children*, Academic Press, New York, N. Y., 1972, 125-139.

Newell, A., "A Theoretical Exploration of Mechanisms for Coding the Stimulus," in A. W. Melton and E. Martin (eds.), *Coding Processes in Human Memory*, Winston and Sons, Washington, D. C., 1972, 337-434.

Newell, A., J. Barnett, J. W. Forgie, C. Green, D. Klatt, J. C. R. Licklider, J. Munson, D. R. Reddy and W. A. Woods, "Speech Understanding Systems: Final Report of a Study Group," (Published for *Artificial Intelligence*), North-Holland/American Elsevier Publishing Co., New York, N. Y., 1973.

Newell, A., "You Can't Play 20 Questions with Nature and Win: Projective Comments on the Papers of this Symposium," in William G. Chase (ed.), *Visual Information Processing*. (In press)

Newell, A., "Production Systems: Models of Control Structures," in William G. Chase (ed.), *Visual Information Processing*. (In press)

For other references by A. Newell, see C. G. Bell, J. Grason and —, and M. Barbacci, C. G. Bell and —.

For references by J. D. Oakley, see S. Klein *et al.*

For references by R. B. Ohlander, see D. R. Reddy *et al.*

Parnas, D. L., "On The Criteria to be used in Decomposing Systems into Modules," *CACM*, Vol. 15, No. 12, December 1972.

Parnas, D. L., "Sample Man-Machine Interface Specification — A Graphics Based Line Editor," in *Display Use for Man-Machine Dialog*, Wolfgang Handler, Joseph Weizenbaum, Carl Hanser (eds.), Springer-Verlag, Munchen, 1972.

Parnas, D. L., "Information Distribution Aspects of Design Methodology," *Information Processing 71*, North-Holland Publishing Co., New York, N. Y., 1972.

Parnas, D. L. and A. N. Habermann, "Comment on Deadlock Prevention Method," *CACM*, Vol. 15, No. 9, September 1972.

For other references by D. L. Parnas, see P. B. Hansen *et al.*

Reddy, D. R., D. Bihary, W. J. Davis and R. B. Ohlander, "Computer Analysis of Neuronal Structure," in Kater and Nicholson, *Neuronal Analysis and Dye Injection Techniques*, Springer-Verlag, New York, N. Y., February 1973.

Reddy, D. R., "Segment-Synchronization Problem in Speech Recognition," presented at *78th Meeting of the Acoustical Society, JASA*.

- Reddy, D. R., L. D. Erman and R. B. Neely, "A Model and a System for Machine Recognition of Speech," *IEEE Trans. on Audio and Electroacoustics*, AU-21, No. 3, June 1973, 229-238.
- Reddy, D. R., W. Broadley, L. D. Erman, R. K. Johnson, J. Newcomer, G. Robertson and J. Wright, "XCRIBL - A Hardcopy Scan Line Graphics System for Document Generation," *Information Processing Letters*, 1972, 246-251.
- For other references by D. R. Reddy, see L. D. Erman, —, and R. B. Neely and A. Newell *et al.*
- For references by S. Rege, see C. G. Bell *et al.*
- For references by G. Robertson, see D. R. Reddy *et al.*
- Shaw, M., "A System for Structured Programming," *SIGPLAN Notices*, Vol. 8, No. 6, June 1973.
- Shaw, M., "Immigration Course in Computer Science: Teaching Materials and 1972 Schedule," *SIGSCE Bulletin*, Vol. 5, No. 2, June 1973, 26-32.
- Shaw, M. and J. F. Traub, "On the Number of Multiplications for the Evaluation of a Polynomial and All its Derivatives," *Proc. Thirteenth Annual Symposium on Switching and Automata Theory*, October 1972, 105-107.
- For other references by M. Shaw, see W. A. Wulf and —.
- Siewiorek, D. P., C. G. Bell and J. Grason, "Register Transfer Modules (RTMs) for Understanding Digital Systems Design," *Proc. COMPCON 72*, San Francisco, Ca., September 1972, 305-308.
- Siewiorek, D. P., P. Goel and J. Grason, "Structural Factors in Fault Dominance and Test Bounds for Combinational Logic Circuits," *1973 International Symposium on Fault Tolerant Computing*, Palo Alto, Ca., June 1973.
- Siewiorek, D. P. and E. J. McCluskey, "Switch Complexity in Systems for Hybrid-Redundancy," *IEEE Trans. on Computers (Special Issue on Fault-Tolerant Computing)*, Vol. C-22, March 1973, 276-282.
- Siewiorek, D. P. and E. J. McCluskey, "An Iterative Cell Switch Design for Hybrid Redundancy," *IEEE Trans. on Computers (Special Issue on Fault-Tolerant Computing)*, Vol. C-22, March 1973, 290-297.
- Siewiorek, D. P., M. Barbacci and C. G. Bell, "PMS: A Notation to Describe Computer Structures," *Computer*, Vol. 6, No. 3, March 1973, 19-21.
- For other references by D. P. Siewiorek, see C. G. Bell *et al.*
- Simon, D. P. and H. A. Simon, "Alternative Uses of Phonemic Information in Spelling," *Review of Educational Research*, Vol. 43, April 1973, 115-137.
- Simon, H. A., "The Sizes of Things," in Judith M. Tanur, *et al.* (eds.), *Statistics: A Guide to the Unknown*, Holden-Day Publishing Co., San Francisco, Ca., 1972, 195-202.
- Simon, H. A., "On the Development of the Processor," in Sylvia Farnham-Diggory (ed.), *Information Processing in Children*, Academic Press, Inc., New York, N. Y., 1972, 3-22.
- Simon, H. A., Review of "Explanation in the Behavioral Sciences: Confrontations," Robert Borget and Frank Cioffi (eds.), *The Philosophical Quarterly*, Vol. 22, July 1972, 278-280.
- Simon, H. A., "What is Visual Imagery? An Information Interpretation," in L. W. Gregg (ed.), *Cognition in Learning and Memory*, John Wiley & Sons, Inc., New York, N. Y., 1972, 183-204.
- Simon, H. A., "Complexity and the Representation of Patterned Sequences of Symbols," *Psychological Review*, Vol. 79, September 1972, 369-382.
- Simon, H. A., "Central Issues in Designing Management Information Systems," in L. Radanovic (ed.), *Organizations and Computers*, Center for Advanced Studies, Belgrade, 1972, 88-127.

- Simon, H. A., "Social Change and Human Behavior: Perceiving and Thinking," *Proc. Conference on the 25th Anniversary of the National Mental Health Act, Mental Health Challenges: Past and Future*, 44-46.
- Simon, H. A., "Cognitive Control of Perceptual Processes," in George V. Coelho and Eli Rubinstein (eds.), *Social Change and Human Behavior*, National Institutes of Mental Health, Rockville, Md., 1973, 91-109.
- Simon, H. A., "Sistemu Sekkei to Soshikiron (System Planning and Organization Theory)," *Soshiki Kagaku (Organizational Science)*, Vol. 6, Winter, 1972, 27-34.
- Simon, H. A., "Mao's China in 1972," *Items (Social Science Research Council)*, Vol. 27, No. 1, March 1973, 1-4.
- Simon, H. A., "Applying Information Technology to Organization Design," *Public Administration Review*, Vol. 33, May-June 1973, 268-278.
- For other references by H. A. Simon, see W. G. Chase and —, and D. P. Simon and —.
- Sirovich, F., "Some Ideas on Semantic Memory in Automatic Learning of Heuristics," in *Atti del Convegno di Informatica Teorica, AICA*, March 1973, 413-439.
- Stewart, G. W., "On the Sensitivity of the Eigenvalue Problem  $Ax = \lambda Bx$ ," *SIAM Journal of Numerical Analysis*, Vol. 9, 1972, 669-686.
- Stone, H. S. and S. H. Fuller, "On the Near-Optimality of the Shortest-Latency-Time-First Drum Scheduling Discipline," *CACM*, Vol. 16, No. 6, June 1973.
- For references by R. J. Swan, see S. H. Fuller, —, and W. A. Wulf.
- Teitlebaum, T., "Context-Free Error Correction by Evaluation of Algebraic Power Series," *5th Annual ACM Symposium on Theory of Computing*, 1973.
- Traub, J. F. (ed.), *Complexity of Sequential and Parallel Numerical Algorithms*, Academic Press, New York, N. Y., 1973.
- Traub, J. F., "Numerical Mathematics and Computer Science," *CACM*, Vol. 15, No. 7, July 1972, 537-541.
- For other references by J. F. Traub, see M. Shaw and —.
- Wile, D. S., "A Generative, Nested-Sequential, Basis for General Purpose Programming Languages," *SIGPLAN Notices*, Vol. 8, No. 6, June 1973.
- Wile, D. S. and C. M. Geschke, "An Implementation Base for Efficient Data Structuring," *International Journal of Computer and Information Sciences*, September 1972.
- Wulf, W. A., "A Case Against the Goto," *Proc. ACM National Conference*, 1972.
- Wulf, W. A. and M. Shaw, "Global Variable Considered Harmful," *SIGPLAN Notices*, Vol. 8, No. 2, February 1973.
- Wulf, W. A., "Systems for System Implementors: Some Experience from Bliss," *Proc. FJCC*, Anaheim, Ca., 1972, 943-948.
- Wulf, W. A. and C. G. Bell, "C.mmp: A Multi-Mini-Processor," *Proc. FJCC*, Anaheim, Ca., 1972, 765-778.
- For other references by W. A. Wulf, see S. H. Fuller, R. J. Swan and —.

## Research Reports

July 1, 1972 to June 30, 1973

Support for the work reported here came largely from the Advanced Research Projects Agency (F44620-70-C-0107) and in part from the National Science Foundation (GJ 32111, GJ 32758x, GJ 28457x, GJ 32259, GJ 30127, GJ 32784) and the Office of Naval Research (N00014-67-A-0314-0010, N00014-67-A-0314-0018).

These reports are registered with the Defense Documentation Center. Accession numbers assigned as of July 1973, are listed after the report titles.

56

In cases of multiple authorship where more than one author is a Faculty member or Research Associate, a cross reference is made to the listing under the name of the principal author.

No cross references are made for graduate students or non-departmental authors.

Barbacci, M. R., "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems," March 1973.

Barbacci, M., C. G. Bell and A. Newell, "ISP: A Language to Describe Instruction Sets and Other Register Transfer Systems," November 1972. (AD 751296)

Bell, C. G., D. P. Bhandarkar, D. L. Feucht, S. L. Rege and D. P. Siewiorek, "Large Scale Integration - A Designer's Viewpoint," November 1972. (PB 221013)

Bhandarkar, D. P. and S. H. Fuller, "A Survey of Techniques for Analyzing Memory Interference in Multiprocessor Computer Systems," April 1973. (AD 762524)

Bhatia, S., "Techniques for Designing Balanced Extensible Computer Systems," Ph.D. Dissertation, October 1972.

Computer Science Departmental Review Committee, "The Computer Science Ph.D. Program at CMU," January 1973.

Eastman, C. and C. Yessios, "An Efficient Algorithm for Finding the Union, Intersection and Differences of Spatial Domains," September 1972. (AD 755958)

Farley, A., J. Dills and M. Shaw (eds.), "CMU PDP-10 Introductory Users Manual," Second edition, September 1972.

Fuller, S. H., "Random Arrivals and MTPT Disk Scheduling Disciplines," *Tech. Report 29*, Digital Systems Lab., Stanford University, Stanford, Ca., August 1972.

Fuller, S. H., "The Expected Difference Between the SLTF and MTPT Drum Scheduling Disciplines," *Tech. Report 28*, Digital Systems Lab., Stanford University, Stanford, Ca., July 1972.

Fuller, S. H., "The Analysis and Scheduling of Devices Having Rotational Delays," Ph.D. Dissertation, Stanford University, Stanford, Ca., August 1972.

Fuller, S. H. and F. Baskett, "An Analysis of Drum Storage Units," *Tech. Report 26*, Digital Systems Lab., Stanford University, Stanford, Ca., August 1972.

Fuller, S. H. and R. C. Chen, "The I/O Port Architecture for Computer Modules," Computer Science and Electrical Engineering Dept.'s, CMU, March 1973.

For other references by S. H. Fuller see D. P. Bhandarkar and —.

Gerhardt, D. and D. L. Parnas, "Window: A Formally Specified Graphics-Based Text Editor," June 1973. (AD 763838)

Gerhart, S. D., "Verification of APL Programs," Ph.D. Dissertation, November 1972. (AD 754856)

Geschke, C. M., "Global Program Optimizations," October 1972. (AD 762621)

Gibbons, G. D., "Beyond REF-ARF: Toward an Intelligent Processor for a Nondeterministic Programming Language," Ph.D. Dissertation, August 1972. (AD 755811)

Habermann, A. N., "Parallel Neighbor Sort," July 1972. (AD 759248)

- Habermann, A. N., "On a Solution and a Generalization of the Cigarette Smokers' Problem," August 1972. (AD 750539)
- Habermann, A. N., "Critical Comments of the Programming Language PASCAL," February 1973.
- Heller, D., "A Determinant Theorem with Applications to Parallel Algorithms," March 1973. (AD 758717)
- Johnsson, R. K., "A Survey of Register Allocation," May 1973. (AD 761529)
- Jones, A., "Protection in Programmed Systems," Ph.D. Dissertation, June 1973. (AD 765535)
- Knudsen, M. J., "PMSL: An Interactive Language for System-Level Description and Analysis of Computer Structures," Ph.D. Dissertation, April 1973. (AD 762513)
- Kung, H. T., "Fast Evaluation and Interpolation," January 1973. (AD 755451)
- Kung, H. T., "The Computational Complexity of Algebraic Numbers," March 1973. (AD 757678)
- Kung, H. T. and J. F. Traub, "Optimal Order of One-Point and Multipoint Iteration," February 1973. (AD 757679)
- Kung, H. T. and J. F. Traub, "Computational Complexity of One-Point and Multipoint Iteration," May 1973. To appear in the *Proc. of Symposium on the Complexity of Computation*, R. Karp (ed.), American Mathematical Society, 1973.
- Lauer, H. C., "Correctness in Operating Systems," Ph.D. Dissertation, September 1972. (AD 753122)
- Lipton, R., "On Synchronization Primitive Systems," Ph.D. Dissertation, June 1973.
- Loveland, D. L. and M. E. Stickel, "A Hole in Goal Trees: Some Guidance from Resolution Theory," June 1973.
- McCredie, J. W., "Analytic Models of Time-Shared Computing Systems: New Results, Validations, and Uses," November 1972.
- Neely, R., "On the Use of Syntax and Semantics in a Speech Understanding System," Ph.D. Dissertation, Stanford University, Stanford, Ca., May 1973.
- Newell, A., H. A. Simon, R. Hayes and L. Gregg, "Report on a Workshop in New Techniques in Cognitive Research," Departments of Psychology and Computer Science, CMU, June 1972. (AD 755813)
- Newell, A., D. R. Reddy, S. Evans, J. Gillogly, C. Hedrick, P. Koltun, T. Moran, R. Ohlander and G. Robertson, "AI Study Guide," October 1972.
- For other references by A. Newell, see D. A. Waterman and —.
- Parnas, D. L., "On the Problem of Producing Well Structured Programs," Computer Science Research Review 1971-1972.
- Parnas, D. L., "Some Conclusions from an Experiment in Software Engineering Techniques," June 1972. (AD 759249)
- Parnas, D. L., "On a Solution to the Cigarette Smokers' Problem (without Conditional Statements)," July 1972.
- Parnas, D. L., "Response to Detected Errors in Well Structured Programs," July 1972.
- Parnas, D. L. and D. P. Siewiorek, "Use of the Concept of Transparency in the Design of Hierarchically Structured Systems," September 1972. (AD 753694)
- For other references by D. L. Parnas, see D. Gerhardt and —, and L. Robinson and —.
- Price, W. R., "Implications of a Virtual Memory Mechanism for Implementing Protection in a Family of Operating Systems," Ph.D. Dissertation, June 1973.
- Reddy, D. R., "Eyes and Ears for Computers," March 1973. (AD 760153)

- For other references by D. R. Reddy, see A. Newell *et al.*
- Robinson, L., "Design and Implementation of a Multi-Level System Using Software Modules," June 1973.
- Robinson, L. and D. L. Parnas, "A Program Holder Module," June 1973. (AD 765536)
- 58 Shaw, M., "Reduction of Compilation Costs Through Language Contraction," July 1972.
- Shaw, M., "IC Study Problems," Second Edition, August 1972.
- Shaw, M., "Immigration Course in Computer Science: 1972 Schedule," Septemoer 1972.
- Shaw, M. and J. F. Traub, "On the Number of Multiplications for the Evaluation of a Polynomial and Some of its Derivatives," August 1972. To appear in *JACM*.
- For other references by M. Shaw, see A. Farley, J. Dills and —.
- Siewiorek, D. P. and D. P. Bhandarkar, "The Effect of Checker on System Reliability," September 1972. (AD 751493)
- Siewiorek, D. P. and A. Ingle, "Extending the Error Correction Capability of Linear Codes," April 1973. (AD 760154)
- For other references by D. P. Siewiorek, see C. G. Bell *et al.*, and D. L. Parnas and —.
- For references by H. A. Simon, see A. Newell *et al.*
- Sirovich, F., "Memory System of a Problem Solver Generator," September 1972. (AD 755812)
- Snyder, L., "An Analysis of Parameter Evaluation for Recursive Procedures," June 1973.
- Stewart, G. W., "Conjugate Direction Methods for Solving Systems of Linear Equations," November 1972.
- Stewart, G. W., "The Convergence of Multipoint Iterations to Multiple Zeros," January 1973.
- Stewart, G. W., "Some Iterations for Factoring a Polynomial II. A Generalization of the Secant Method," February 1973.
- Stewart, G. W., "Modifying Pivot Elements in Gaussian Elimination," March 1973. (AD 758065)
- Traub, J. F., "Iterative Solution of Tridiagonal Systems on Parallel or Vector Computers," January 1973. (AD 762053). To Appear in *Complexity of Sequential and Parallel Numerical Algorithms*, Academic Press, 1973.
- Traub, J. F., "Theory of Optimal Algorithms," June 1973. (AD 764101). To appear in *Numerical Mathematical Software*, Institute of Mathematics and its Applications, England, 1973.
- For other references by J. F. Traub, see H. T. Kung and —, and M. Shaw and —.
- Waterman, D. A. and A. Newell, "Preliminary Results with a System for Automatic Protocol Analysis," CIP Working Paper No. 211, Departments of Psychology and Computer Science, CMU, May 1972.
- Wodon, P. L., "Still Another Tool for Synchronizing Cooperating Processes," August 1972. (AD 750538)
- Wulf, W. A., J. L. Apperson, C. M. Geschke, R. K. Johnsson, C. B. Weinstock, D. S. Wile, R. F. Brender, P. A. Kneeven and M. R. Pellegrini, "Bliss-11 Programmers Manual," December 1972.
- Wulf, W. A., E. Cohen, W. Corwin, A. Jones, R. Levin, C. Pierson and F. Pollack, "HYDRA: The Kernel of a Multiprocessor Operating System," June 1973. (AD 762514)

## Colloquia

### September

Jeffrey D. Ullman, Princeton University  
"Code Optimization and Reducible Flow Graphs"  
September 12, 1972

Richard M. Karp, University of California,  
Berkeley  
"Reducibility Among Combinatorial Problems"  
September 14, 1972

Philip S. Dauber, IBM T. J. Watson Research Center  
"Computer Science Research in IBM"  
September 20, 1972

Joyce Friedman, University of Michigan  
"Computer Modeling of Linguistic Theories"  
September 22, 1972

Harold S. Stone, Stanford University  
"Parallel Computers and Parallel Algorithms"  
September 28, 1972

### October

Michael J. Flynn, Johns Hopkins University  
"Microprogramming and Directly Executable  
Languages"  
October 2, 1972

Marvin Minsky, Massachusetts Institute of  
Technology  
"Research in Artificial Intelligence at MIT"  
October 4, 1972

Joel Moses, Massachusetts Institute of  
Technology  
"Algebraic Manipulation"  
October 9, 1972

Douglas McIlroy, Bell Laboratories  
"What Makes Programs Intelligible"  
October 11, 1972

John C. Reynolds, Syracuse University  
"Definitional Interpreters for Higher-Order  
Programming Languages"  
October 13, 1972

Jacob Schwartz, Courant Institute of  
Mathematical Sciences  
"Programming and the Design of Programming  
Languages"  
October 20, 1972

Lynn H. Quam, Stanford University  
"Computer Techniques for Processing Mariner  
Pictures of Mars"  
October 25, 1972

### November

E. M. Reingold, University of Illinois  
"Balanced Binary Search Trees"  
November 9, 1972

David Farber, University of California, Irvine  
"The Distributed Computing System"  
November 29, 1972

### January

Barry Boehm, The RAND Corporation  
"Future Trends in Computing Technology,  
Applications, and Problems"  
January 24, 1973

Gilbert Daniels, Hunt Institute for  
Botanical Documentation  
"Programs for Bibliographic Research"  
January 31, 1973

*February*

Alan Kay, Xerox Palo Alto Research Center  
"Video Graphics"  
February 7, 1973

Wesley Clark and Charles Molnar,  
Washington University  
"Macro Modules We Have Known"  
February 14, 1973

Gene H. Golub, Stanford University  
"Elliptic Partial Differential Equations"  
February 19, 1973

*March*

James Bunch, Cornell University  
"Graphic Theory and Sparse Matrices"  
March 1, 1973

Heinz Klein, State University of  
New York at Buffalo  
"Game Playing Programs to Study  
Decision Making"  
March 7, 1973

Thomas Szymanski, Cornell University  
"Generalized Bottom-Up Parsing"  
March 12, 1973

Michael Levine, Carnegie-Mellon University  
"Symbol Manipulation in Physics Research"  
March 14, 1973

Harry Hunt, Cornell University  
"Predicates on the Regular Sets"  
March 16, 1973

John Pople, Carnegie-Mellon University  
"Computers in Chemistry Research"  
March 21, 1973

*April*

Edmund C. Berkeley, Berkeley Enterprises  
"Computers in Society"  
April 4, 1973

H. R. Strong, IBM T. J. Watson Research Center  
"Theory of Programming"  
April 9, 1973

H. Kobayashi, IBM T. J. Watson Research Center  
"System Measurement and Evaluation"  
April 9, 1973

W. D. Frazer, IBM T. J. Watson Research Center  
"Analysis of Combinatory Algorithms"  
April 10, 1973

L. A. Belady, IBM T. J. Watson Research Center  
"Security"  
April 10, 1973

Ruth M. Davis, National Bureau of Standards  
"Computers in Public Policy"  
April 11, 1973

W. W. Bledsoe, University of Texas at Austin  
"Man-Machine Theorem Proving"  
April 17, 1973

Charles Kriebel, Carnegie-Mellon University  
"Management Information Systems"  
April 18, 1973

Thomas Hull, University of Toronto  
"Proving Correctness of Numerical Algorithms"  
April 25, 1973

*May*

Michael Harrison, University of California  
at Berkeley  
"Parsing Algorithms for Deterministic  
Languages"  
May 3, 1973

### Gifts, Grants and Contracts

- Mellon Computer Research Funds, Department, Computer Research, unrestricted.
- Digital Equipment Corporation, Prof. W. A. Wulf, BLISS Compiler, unrestricted.
- Advanced Research Projects Agency, Prof. A. Newell and Prof. J. F. Traub, Research in Information Processing, June 1970 - June 30, 1973.
- National Science Foundation, Prof. D. Loveland and Prof. P. Andrews, Proof Procedures in Predicate Calculus-Type Theory, July 1, 1971 - June 30, 1973.
- Office of Naval Research, Prof. J. F. Traub, Analysis of Algorithms, July 1, 1971 - August 31, 1973.
- National Science Foundation, Prof. D. L. Parnas, A Tool for Controlling the Complexity of Software Systems, August 1, 1971 - January 31, 1973 and April 1, 1973 - March 31, 1975.
- Alcoa Foundation, Department, 2 partial Scholarships, 1972-73.
- Computer Science Scaife Grant, Department, 1972-73.
- Ford Motor Company, Prof. W. A. Wulf, unrestricted, 1972-73.
- International Business Machines, Department, 1 full fellowship, 1972-73.
- Shell Companies Foundation, Department, Computer Science Research, 1972-73.
- Xerox Corporation, Prof. A. Newell, Research in Cognitive Processes, 1972-73.
- National Science Foundation, Prof. J. F. Traub, Iteration Algorithms: Theory and Applications, January 15, 1972 - June 30, 1973.
- National Science Foundation, Prof. D. R. Reddy, Computer Models of Neural Networks, March 1, 1972 - February 28, 1974.
- National Science Foundation, Prof. C. G. Bell, Register Transfer Level Computer Systems Design, July 1, 1972 - June 30, 1973.
- National Science Foundation, Prof. D. L. Parnas, System Family Concept, July 1, 1972 - June 30, 1974.
- Office of Naval Research, Prof. G. W. Stewart, Design of Algorithms, October 1, 1972 - September 30, 1973.
- Mathematical Social Sciences Board, Prof. A. Newell, Summer Workshop in New Techniques for Cognitive Research, 1973 Conference (NSF Sub-Grant).

### Ph.D. Dissertations

The following persons have been awarded Ph.D.'s in Computer Science and related areas since the establishment of the Computer Science Department in 1965. The department or program from which each received his Ph.D. is followed by his most recent position.

Support for this work came largely from the Advanced Research Projects Agency under contract F-44620-70-C-0170. The accession numbers follow in parentheses after those dissertations which are registered as reports with the Defense Documentation Center.

62

Balzer, Robert M. (Systems and Communication Sciences), Research Scientist, RAND Corporation, Santa Monica, California, "Studies Concerning Minimal Time Solutions to the Firing Squad Synchronization Problem," 1966, Professor A. Newell. (AD 635056)

Berglass, Gilbert R. (Systems and Communication Sciences), Assistant Professor of Computer Science, State University of New York at Buffalo, Buffalo, New York, "A Generalization of Macro Processing," 1970, Professor A. J. Perlis.

Caviness, B. F. (Mathematics), Assistant Professor of Computer Science, University of Wisconsin, Madison, Wisconsin, "On Canonical Forms and Simplification," 1967, Professor A. J. Perlis. (AD 671938)

Coles, L. Stephen (Systems and Communication Sciences), Senior Research Mathematician, Stanford Research Institute, Menlo Park, California, "Syntax Directed Interpretation of Natural Language," 1967, Professor H. A. Simon. (AD 655923)

Darringer, John A. (Systems and Communication Sciences), IBM T. J. Watson Research Center, Yorktown Heights, New York, "The Description, Simulation, and Automatic Implementation of Digital Computer Processors," 1969, Professor D. L. Parnas. (AD 700144)

Earley, Jay (Computer Science), Research Associate, Department of Computer Science, University of California, Berkeley, California, "An Efficient Context-Free Parsing Algorithm," 1968, Professor R. W. Floyd.

Ernst, George (Systems and Communication Sciences), Associate Professor, Department of Computer Science, Computer Engineering Division, Case Western Reserve University, Cleveland, Ohio, "Generality and GPS," 1966, Professor A. Newell. (AD 809354)

Evans, Arthur, Jr. (Mathematics), Lincoln Laboratory, Lexington, Massachusetts, "Syntax Analysis by a Production Language," 1965, Professor A. J. Perlis. (AD 625465)

Feldman, Jerome A. (Mathematics), Associate Professor of Computer Science, Department of Computer Science, Stanford University, Stanford, California, "A Formal Semantics for Computer Oriented Languages," 1964, Professor A. J. Perlis. (AD 462935)

Fikes, Richard E. (Computer Science), Research Mathematician, Stanford Research Institute, Menlo Park, California, "A Heuristic Program for Solving Problems Stated as Nondeterministic Procedures," 1969, Professor A. Newell. (AD 688604)

Fisher, David (Computer Science), Assistant Professor of Systems and Information Science, Vanderbilt University, Nashville, Tennessee, "Control Structures for Programming Languages," 1970, Professor A. J. Perlis. (AD 708511)

Freeman, Peter A. (Computer Science), Assistant Professor, Department of Information and Computer Science, University of California, Irvine, California, "Sourcebook for OSD - An Operating System Designer," 1970, Professor A. Newell.

Gerhart, Susan L. (Computer Science), Duke University, Durham, North Carolina, "Verification of AFL Programs," 1973, Professor D. Loveland. (AD 754856)

Geschke, Charles M. (Computer Science), Xerox Research Center, Palo Alto, California, "Global Program Optimization," 1973, Professor W. A. Wulf.

Gibbons, Gregory D. (Computer Science), Assistant Professor, Naval Post Graduate School, Monterey, California, "Beyond REF-ARF: Toward an Intelligent Processor for a Nondeterministic Language," 1973, Professor A. Newell. (AD 755811)

- Grason, John (Electrical Engineering [Systems and Communication Sciences]), Assistant Professor of Electrical Engineering, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Methods for the Computer-Implemented Solution of a Class of 'Floor Plan' Design Problems," 1970, Professor H. A. Simon. (AD 717756)
- Haney, Frederick M. (Computer Science), Advanced Systems Staff, Xerox Corporation, El Segundo, California, "Using a Computer to Design Computer Instruction Sets," 1968, Professor C. G. Bell. (AD 671939)
- Iturriaga, Renato (Computer Science), Director of the Computation Center and of the Center for Research on Applied Mathematics, University of Mexico, Mexico City, "Contributions to Mechanical Mathematics," 1967, Professor A. J. Perlis. (AD 660127)
- King, James C. (Computer Science), Research Staff, IBM T. J. Watson Research Center, Yorktown Heights, N. Y., "A Program Verifier," 1970, Professor R. W. Floyd. (AD 699248)
- Knudsen, Michael J. (Computer Science), Research Associate, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "PMSL, An Interactive Language for System-Level Description and Analysis of Computer Structures," 1973, Professor C. G. Bell.
- Ladron DeCegama, Angel (Electrical Engineering [Systems and Communication Sciences]), Senior Research Engineer, National Cash Register Company, Hawthorne, California, "Performance Optimization of Multiprogramming Systems," 1970, Professor A. J. Perlis.
- Lauer, Hugh C. (Computer Science), Lecturer, Computing Laboratory, University of Newcastle, Newcastle Upon Tyne, England, "Correctness in Operating Systems," 1973, Professor W. A. Wulf. (AD 753122)
- Lindstrom, Gary (Computer Science), Assistant Professor of Computer Science, University of Pittsburgh, Pittsburgh, Pennsylvania, "Variability in Language Processors," 1970, Professor A. J. Perlis. (AD 714695)
- London, Ralph L. (Mathematics), Information Sciences Institute, University of Southern California, Marina Del Rey, California, "A Computer Program for Discovering and Proving Sequential Recognition Rules for Well-Formed Formulas Defined by a Backus Normal Form Grammar," 1964, Professor A. Newell. (AD 840036)
- Manna, Zohar (Computer Science), Associate Professor, Applied Mathematics Department, Weizmann Institute of Science, Rehovot, Israel, "Termination of Algorithms," 1968, Professor R. W. Floyd. (AD 670558)
- McCredie, John W. (Systems and Communication Sciences from Graduate School of Industrial Administration), Assistant Professor of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Analytic Models of Time-Shared Computing Systems: New Results, Validations, and Uses," 1972, Professor C. Kriebel.
- McCreight, Edward M. (Computer Science), Research Scientist, Xerox Research Center, Palo Alto, California, "Classes of Computable Functions Defined by Bounds on Computation," 1970, Professor A. R. Meyer. (AD 693327)
- Mitchell, James G. (Computer Science), Xerox Research Center, Palo Alto, California, "The Design and Construction of Flexible and Efficient Interactive Programming Systems," 1970, Professor A. J. Perlis. (AD 712721)
- Moore, James A. (Computer Science), Research Associate, Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "The Design and Evaluation of a Knowledge Net for MERLIN," 1971, Professor A. Newell.
- Mullin, James K. (Systems and Communication Sciences), Associate Professor, Computer Science Department, University of Western Ontario, London, Ontario, Canada, "A Computer Optimized Question Asker for Aiding Bacteriological Species Identification COQAB," 1967, Professor B. Green.

- Parnas, David L. (Systems and Communication Sciences), Professor, Technical University of Darmstadt, West Germany, "System Function Description ALGOL - A Language for the Description of the Functions of Finite State Systems, the Simulation of Finite Systems, and the Automatic Production of the State Tables of Such Systems," 1965, no advisor. (AD 467633)
- Pfefferkorn, Charles (Computer Science), ILLIAC-IV Project, Evans and Sutherland Corp, San Francisco, California, "Computer Design of Equipment Layouts Using the Design Problem Solver (DPS)," 1971, Professor H. A. Simon.
- Quatse, Jesse T. (Electrical Engineering and Systems and Communication Sciences), University of California, Berkeley, California, "A Highly-Modular Organization of General Purpose Computers," 1969, Professor A. Newell and Professor C. G. Bell.
- Quillian, M. Ross (Psychology), Associate Professor, Social Sciences Department, University of California, Irvine, California, "Semantic Memory," 1967, Professor H. A. Simon.
- Richardson, Leroy (Systems and Communication Sciences), Staff Scientist, Information Sciences Institute, University of Southern California, Marina Del Rey, California, "Specification Techniques for Interactive Computer Systems," 1972, Professor D. L. Parnas.
- Shaw, Mary M. (Computer Science), Assistant Professor of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, "Language Structures for Contractible Compilers," 1972, Professor A. J. Perlis. (AD 744117)
- Shoup, Richard (Computer Science), Xerox Research Center, Palo Alto, California, "Programmable Cellular Logic Arrays," 1970, Professor C. G. Bell. (AD 706891)
- Siklossy, Laurent (Computer Science), Computer Sciences Department, University of Texas, Austin, Texas, "Natural Language Learning by Computer," 1968, Professor H. A. Simon. (AD 671937)
- Snyder, Lawrence (Computer Science), Assistant Professor of Computer Science, Yale University, New Haven, Connecticut, "An Analysis of Parameter Evaluation for Recursive Procedures," 1973, Professor A. N. Habermann.
- Standish, Thomas A. (Computer Science), Senior Scientist, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, "A Data Definition Facility for Programming Languages," 1967, Professor A. J. Perlis. (AD 658042)
- Strauss, Jon C. (Systems and Communication Sciences), Associate Professor of Computer Science, Washington University, St. Louis, Missouri, "Identification of Continuous Dynamic Systems by Parameter Optimization," 1965, Professor A. Lavi. (AD 660887)
- Strecker, William D. (Electrical Engineering), Research and Development Group, Digital Equipment Corporation, Maynard, Massachusetts, "An Analysis of the Instruction Execution Rate in Certain Computer Structures," 1970, Professor C. G. Bell. (AD 711408)
- Wagner, Robert A. (Computer Science), Associate Professor, Department of Systems and Information Science, Vanderbilt University, Nashville, Tennessee, "Some Techniques for Algorithm Optimization with Application to Matrix Arithmetic Expressions," 1969, Professor A. J. Perlis. (AD 678629)
- Waldinger, Richard J. (Computer Science), Research Mathematician, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, "Constructing Programs Automatically Using Theorem Proving," 1969, Professor H. A. Simon. (AD 697041)
- Williams, Donald S. (Systems and Communication Sciences), Member Technical Staff, Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, "Computer Program Organization Induced by Problem Example," 1969, Professor H. A. Simon. (AD 688242)
- Winikoff, Arnold W. (Systems and Communication Sciences), President, Q.E.D., Inc., Minneapolis, Minnesota, "Eye Movement as an Aid to Protocol Analysis of Problem Solving Behavior," 1967, Professor A. Newell.

**Best  
Available  
Copy**

