# PROJECT MAC

## PROGRESS REPORT VI

JULY 1968 to JULY 1969



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Cambridge Massachusetts 02139



152

# PROGRESS REPORT VI

JULY 1968 to JULY 1969

Work reported herein was supported by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01), (02). The primary support for some of this work came from the M.I.T. departments and laboratories participating in Project MAC, whose research programs are, in turn, sponsored by various government and private agencies. This support is acknowledged by specific mention of agency and contract number in the appropriate sections.



PROJECT MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

545 Technology Square Cambridge, Massachusetts 02139

PROJECT MAC

JULY 1968 to JULY 1969

S. M. Adams
G. K. Adler
J. W. Alsop
E. I. Ancona
R. M. Baecker
A. Bagchi
B. Bailin
M. E. Baker
M. D. Beeler
V. Berardinelli
W. T. Beyer
W. D. Bilofsky
T. O. Binford
Prof. M. Blum
M. C. Bogue, III
H. E. Brammer
A. L. Brown
M. C. Burnham
D. E. Burmaster
R. L. Bushkoff
M. L. Cabral
T. F. Callahan
R. H. Campbell
I. R. Campbell-Grant
R. A. Carpenter
L. Cavallaro
J. H. Cecil
E. Charniak
G. F. Clancy
D. D. Clark
Prof. F. J. Corbato
C. A. Dancy, III
R. C. Daley
P. E. deCoriolis
H. M. Deitel
Prof. J. B. Dennis
F. L. DeRemer
Prof. J. J. Donovan
C. P. Doyle
H. R. Drab, Jr.
D. A. Duggento
S. D. Dunten
R. S. Eanes
D. E. Eastlake
J. W. Eddleman
M. Edelberg
Prof. A. Evans, Jr.
Prof. R. M. Fano
M. N. Fateman
R. J. Feiertag
H. Fell
Prof. R. R. Fenichel
L. T. Flynn
J. S. Freiberg
R. L. Gardner
C. Garman
S. L. Geffner
J. L. Gertz
J. P. Golden
R. C. Goldstein
R. W. Gosper
Prof. R. M. Graham
I. G. Greif
R. D. Greenblatt
C. D. Graceffa
R. S. Green
J. M. Greene
A. K. Griffith
J. M. Grochow
J. A. Gunn
Prof. A. Guzman
J. P. Haggerty
M. M. Hammer
P. G. Hebalkar
D. A. Henderson, Jr.
W. H. Henneman
Prof. F. C. Hennie
A. Herskovits
E. Hewitt
K. H. Hill
M. G. Hinchey
P. Hirson
J. T. Holloway
P. Holloway

B. K. P. Horn
W. F. Hui
K. M. Jacobs
M. Jacobson
J. L. Jaroslav
P. Jarvis
J. W. Johnson
D. L. Jones
F. Jones
Prof. M. M. Jones
T. L. Jones
E. I. Kampits
E. Klang
D. J. Kfoury
T. F. Knight
Prof. Z. Kohavi
D. Kontrimus
L. J. Krakauer
J. C. R. Licklider
L. K. Lipman
Prof. C. L. Liu
Prof. F. L. Luconi
R. F. Mabee
L. L. Maddaus
S. E. Madnick
P. S. Malek
R. Mandl
R. G. Mansfield
M. J. Marcus
K. J. Martin
Prof. W. A. Martin
E. W. Meyer
J. Milner
Prof. M. L. Minsky
G. H. H. Mitchell
S. Montgomery
E. G. Moore
R. C. Moore
S. C. Morr
J. H. Morris, Jr.
N. I. Morris
Prof. J. Moses
R. E. Neubauer
R. Noftsker
S. Ohayon
Prof. E. I. Organick
M. A. Padlipsky
L. G. Pantalone
Prof. S. A. Papert
S. S. Patil
D. N. Perkins, Jr.
J. E. Pinella
L. K. Platzman
C. Ramchandi
D. H. Randall
R. L. Rappaport
H. A. Rideout
E. Roderick
J. S. Roe
S. L. Rosenbaum
N. L. Ross
L. J. Rotenberg
Prof. J. H. Saltzer
P. R. Samson
D. C. Scanlon
R. R. Schell
M. D. Schroeder
R. C. Schroeppel
A. Sekino
L. Seligman
L. L. Selwyn
T. Seymour
J. M. Shah
T. P. Skinner
S. W. Smoliar
W. Southworth
J. W. Spall
M. Speciner
M. J. Spier
W. A. Spies
M. A. Stallings
N. F. Stone
G. J. Sussman
J. E. Sussman
R. H. Thomas
M. R. Thompson
R. C. Thurber
H-M D. Toong
D. H. Vanderbilt
T. H. VanVleck
K. D. Venezia
C. A. Vogt

V. L. Voydock
C. T. Waldrop
D. L. Waltz
M. E. Wantman
M. B. Weaver
M. W. Webber
S. H. Webber
Prof. J. Weizenbaum
D. M. Wells
C. M. White
J. L. White
T. Winograd
P. H. Winston
E. M. Wolman
Prof. J. M. Wozencraft
F. Wright
C. Ying
K. Young
M. L. Young
S. N. Zilles

Guests

N. Adelman
Prof. C. K. Chow
T. G. Evans
Prof. E. Fredkin
H. Hegna
Prof. H. N. Mahabala
Prof. M. S. Paterson
A. Sasaki
G. Voyat

PROGRESS REPORT VI

Contents

## PROJECT MAC ADMINISTRATION

| | |
|---|---|
| Prof. J. C. R. Licklider | Director |
| Prof. M. M. Jones | Assistant Director |
| David E. Burmaster | Assistant Director for Student Activities |
| Dorothea C. Scanlon | Assistant to the Director |
| H. E. Brammer | Assistant to the Director (to June 1969) |
| M. L. Cabral | Business Manager (to March 1969) |

L. Cavallaro
D. A. Duggento
R. C. Goldstein
C. D. Graceffa
J. M. Greene
J. A. Gunn
D. Kontrimus
R. G. Mansfield
L. L. Maddaus
E. G. Moore
L. G. Pantalone
E. Roderick
R. T. Stetson
C. M. White
E. M. Wolman
M. L. Young

# PART I

# PROJECT MAC PROGRESS REPORT VI

## INTRODUCTION

Project MAC is an M.I.T. interdepartmental research laboratory for computer science and engineering. The research program of Project MAC is focused upon the new field of "interactive" computing, i.e., computing in which men and computers work together in close partnership in solving problems or making decisions. Some of the research is aimed at making it possible for men and computers to work together more effectively. Some of it takes advantage of interactive computing to facilitate the solution of basic problems in computer science or to develop new applications. But almost all of its deals with men and computers -- hence the acronym "MAC". (There are other expansions of "MAC"; those we use most often are "multiple-access computers" and "machine-aided cognition".

During the last year, Project MAC has accomplished several research objectives, and it has gone through part of a major transition. Inasmuch as a brief account of the transition will set the stage for a discussion of the research results, let us begin with that account.

### The Course of Project MAC

In order to bring about a partnership between men and computers, one must make it possible -- and economically practicable -- for men and computers to work together directly and effectively. The first major undertaking of Project MAC, beginning in 1963, and building on early work of the M.I.T. Computation Center, was to create a computer system with which many people could work simultaneously and conveniently. The result was the first general-purpose multi-access computer system, CTSS, the Compatible Time-Sharing System. (Actually, two CTSS's were built; one was operated by Project MAC, the other by the Computation Center.)

By 1965, CTSS was the focus of a significant part of the intellectual effort of about 200 research people at M.I.T. and quite a few at other universities. It was obvious that interactive computing opened new horizons. The present computer "time-sharing" industry, with its 150 to 200 service companies, is one of them. CTSS made it evident that a multi-access computer could provide the communication as well as the information storage and processing facilities required for the emergence of a new kind of intellectual community. Partly by having them and partly by lacking them, CTSS made it possible to see what characteristics and features a "community" multi-access system should possess.

The effect of CTSS upon Project MAC was threefold: its quick success, together with the flooding-in upon the designers of ideas about how to build a far better system, led to the determination to create a truly comprehensive community computer system, and, with the General Electric Company and the Bell Telephone Laboratories, Project MAC initiated a major effort toward that goal. Not all the computer scientists and engineers who contributed to the development of CTSS wanted to participate in the development of the more advanced system: many turned to research in other aspects of computer science and engineering. At the same time, CTSS tended to bring its community of substantive users -- people interested in using interactive computing to facilitate research in their various fields -- into Project MAC. Thus Project MAC came to have three main parts: (1) the Computer System Research Group, developing the new multi-access computer system, (2) several smaller research programs in computer science and engineering, and (3) a large number of users of CTSS.

During 1967 and 1968, as the use of CTSS turned from a research experience into an operational routine, and as the development of the new multi-access system required an increasing fraction of the available funds, the third part dwindled. This past year, the CTSS operated by Project MAC was transferred to the Information Processing Center (formerly Computation Center), and Project MAC became a two-part laboratory. That was the first part of the transition.

The second part of the transition began on the first of January 1969. The new multi-access system, called the Multiplexed Information and Computing Service (Multics), had proven to be much more complex and difficult than anticipated, and its development had proceeded at just half the scheduled speed -- until the beginning of the new year. Then everything began to go well, the schedule quit slipping and now (July 1969) the heart of Multics beats, the Multics Operating System operates.

The Multics that now "runs" is a bare-bones system, but an advanced one. The Computer System Research Group is confident that it will be opened for general use on 1 October 1969. That will begin the third part of the transition. The main system research effort will turn from the operating system to the comprehensive library procedures and data required to convert such a "system" into a "service", to make it effectively and conveniently useful to others than computer buffs. This service-building effort will lead, we believe, to a realization of our now long-dreamed dream of an "on-line" intellectual community. But it will not bring the whole community into Project MAC -- only the service builders. Operational responsibility for Multics will be transferred to the Information Processing Center.

## Characteristics and Features of Multics

Inasmuch as the demonstration of the Multics Operating System was one of the main accomplishments of the past year, it is an appropriate time to explain the basis for the claim that Multics provides a new and unique facility for a true on-line community of users, a foundation for unprecedented teamwork in many kinds of undertaking that are based upon information. There are two main areas in which Multics represents a major advance. In the first of these areas, a few other systems, developed since the design of Multics was published, are comparable to Multics. In the second, to the best of our knowledge, Multics is unique.

First, Multics introduces a great simplification into the individual user's concept of the computer system. Heretofore, he had not only to remember the names of his programs and sets of data but also to keep in mind their sizes and, if large, squeeze them (all together or in subsets) into the computer's memory, which was pictured as a single series of little bins, each capable of holding one "word" of procedure or data. Using Multics, he thinks only in terms of the names; the operating system brings into memory those procedures and data required at each moment and does not waste expensive memory space (which can be put to good purpose by other users) on a user's inactive information. Programming within the context of Multics, the user has available to him the equivalent of millions of pieces of writing paper, each just as long as he needs it to be -- but he need not remember how long, or where he put the pieces; he just names them and remembers the names. (If he forgets the names, Multics will of course provide a listing.) Nor does the user have to think about the configuration of the of the "hardware" computer. It may change basically, e.g., in number of processing units or in number of memory blocks, through failure of subsets or through augmentation, but the only way that affects the user is to slow down or speed up the service. His working image of the configuration of the computer system remains the same: a "space" occupied by the names of procedures and data.

Second, Multics greatly facilitates cooperation among its users. It makes it easy for them to work in pairs or groups, using shared as well as individual programs and data sets and communicating with one another through their consoles as well as face-to-face or by telephone. To use another person's procedure or data, one has only to get his permission. One does not have to borrow a copy either overtly or within the system; two or more people can use the same "copy" of a procedure or of a set of data at the same time. Multics keeps one of them (and his programs) from changing it while the other (another) is in the process of "reading" it. Without permission of the owner, access to files is barred. The owner can extend permission to any individual or established group to use one of his files in any one of several ways: "read and change", "read only", "execute only", and "use only through a privileged program". The Multics file system provides the basis for free, voluntary cooperation,

and also the basis for entrepreneurial cooperation for a fee: Multics will keep records of the use of certain programs and data and send bills on behalf of the owner.

Professor F. J. Corbato, leader of the Computer System Research Group, has a list of 30 characteristics and features, of which the foregoing are a few, that define Multics. It will not be long, now, until we see whether they will make the expected difference, whether they will greatly facilitate the individual's use of his own information and, at the same time, turn the computer into a communication network.

Knowledge and Heuristics in the Computer

Computers are so fast and accurate in their execution of procedures that they proved themselves very useful long before any computer was programmed to do anything that seemed at all intelligent. Indeed, to combine the computer's speed and accuracy in applying defined procedures to specified data with man's ability to formulate and evaluate is the essential aim of man-computer interaction. But communication between men and ordinary computer systems is so poor as to frustrate any attempt at teamwork. There can be little man-computer partnership if the computer knows practically nothing and has to be told at every stage precisely what to do and in detail how to do it.

A large and important part of the work of Project MAC is devoted, therefore, to learning how to "educate" the computer to be a better partner. It does no good simply to fill the computer's memories and stores with facts; the uneducated computer can remember but not use them. It does little good to fill the memories and stores with procedures; the uneducated computer has to be told exactly when and to what data to apply each one. The problem is, essentially, to understand the process of being intelligent and to program at least some of that process into the computer. It is clear that the process involves knowledge, which can be defined roughly as facts structured into a model that can be "run" or be interpreted by the processor, and heuristics, which can be defined roughly as guidelines to solution or discovery.

In Project MAC, during the past year, several accomplishments were made in the education of computers. In most instances, the computer system was the Incompatible Time-Sharing System (ITS), a system somewhat smaller than CTSS and Multics and specialized in quite different ways. You have to be a computer buff to use ITS -- but, if you are, you like it so much that you do nothing to make it easy for non-buffs to understand. As a result of the educational accomplishments, ITS can -- among other things -- now:

1) Solve calculus "word problems" of the type, "A ladder 20.0 feet long leans against a house. Find the rate at which the top of the ladder is moving if its

foot is 12.0 feet from the house and moving away from the house at the rate of 2.0 feet per second." (E. Charniak) (But see Part II of this Report for the section on Artificial Intelligence and Intelligent Automata for a discussion of the difficulty of this problem.)

2) Look (through its television-camera "Eye") at a haphazard pile of toy blocks on the floor, analyze the scene into individual blocks (Prof. A. Guzman; B. K. P. Horn), pick them up one-by-one (with its mechanical arm and hand), and stack them up to make a tower.

3) Play Class C chess instead of Class D chess, as last year. (R. Greenblatt)

4) Solve symbolic (non-numeric) integration problems, now even including problems involving logarithms and exponentials, such as the Gaussian ("error") function. (Prof. J. Moses)

The systems of programs that do the foregoing things, as well as others now operating, have some knowledge and some heuristic capability, but not much. They are successful only in quite special and restricted contexts. A large group, led by Professors M. M. Minsky and S. A. Papert, is working intensively to advance the understanding of such problems.

Man-Computer Interaction

Next to the stupidity of ordinary computer systems, the main barrier to effective and convenient communication between men and computers has been the computer "console". In practical fact, the console used in 98 per cent of all on-line man-computer communication these last few years has been merely an electric typewriter with additional electromechanical parts to send codes to and receive codes from a computer. Such typewriters are narrow bottlenecks. Obviously they had to give way to, or at least be supplemented by, graphic displays and non-keyboard computer-input devices such as pencils and microphones.

The past year saw a marked advance in the commercial supply of ultra-typewriter equipment for man-computer interaction. The Advanced Remote Display Station (ARDS), developed by the Electronic Systems Laboratory in conjunction with Project MAC, led an important part of the advance. Its storage cathode-ray screen provides a non-flickering image of fairly good resolution and the capacity to display a standard page of typescript (unfortunately at somewhat reduced size). Most importantly, it can display graphs, diagrams and maps, and it presents information very much faster than a typewriter.

Perceiving the importance of improved man-computer interaction techniques to the future of interactive information processing,

Project MAC recently augmented its research programs in computer graphics and dynamic modeling. As Multics moves from its developmental to its operational phase, increasing effort will be devoted to understanding and development of those areas.

Computer Language

The intrinsic "language" in terms of which a computer carries out its internal processes is hardly a language in the everyday sense; it is just a repertoire of primitive operations upon information in its memory and upon its own configuration. Great difficulties arose when men tried to communicate with the computer in its own "machine language". One of the main practical advances of the early years of computing was the development of "higher level" languages for use in preparing programs: FORTRAN, ALGOL, and so on. Then, when such languages were in widespread use, people began to subject them to linguistic analysis and to try to understand them theoretically. That effort led to a clearer understanding of natural languages as well as computer programming languages. It is a continuing study, but a part of it recently reached a culmination.

Much of the effort of the Computer Linguistics Group of Project MAC, led by Prof. J. M. Wozencraft, had focused on working out a systematic analysis and exposition of the basic concepts of computer programming languages on the basis of a method called the lambda calculus. That effort was completed at the end of 1968. The exposition is presented in the class notes of Computer Linguistics (6.231) and in several publications. The end of 1968 was thus the end of an era. Professor Wozencraft took leave of absence to serve as Associate Head of a division of the Lincoln Laboratory. The group, now under the leadership of Prof. R. M. Graham, is attacking new computer language problems: languages for programming operating systems (Prof. Graham), languages that can be readily extended by the individual programmer or programming team to meet special requirements (Prof. A. Evans, Jr.), and a new way of formalizing computer languages and formulating and providing theorems about them (Prof. J. J. Donovan).

At the same time, it is becoming increasingly clear that there is more to computer language than just language for programming. As the library of programs grows, the ratio of program preparation to program use decreases, and the languages through which people interact with programs ("interaction languages") become more important. In a part of this new area, the area of data description languages, Project MAC and the Special Interest Committee on File Description and Translation of the Association for Computing Machinery together recently initiated a weekly seminar.

## Other Research Areas

Project MAC is active in other research areas, also, and made significant advances in many of them during the past year. Professors J. Weizenbaum and R. R. Fenichel completed the first version of their computer program, TEACH, that teaches computer programming, and "proved" it on an introductory class. This work was joint with the Education Research Center. Professors Donovan and M. M. Jones and their students developed an interactive simulation programming language, SIMPLE. W. T. Beyer found a way to reduce very greatly the amount of computation required in certain kinds of geometrical information processing; he found, for example, that through use of a parallel computer paradigm the number of computational steps required to determine whether or not two figures, represented in a grid, are connected can be made proportional to the number of intervals in one dimension of the grid rather than to that number squared. Harriet J. Fell worked out a way of dealing mathematically with topological problems that arise in discrete (i.e., gridlike) approximations to continuous spaces; in mathematical terms, she showed that the Hausdorff topology is the intersection of all possible grid topologies. And so on. It is not possible to do justice to them in a brief review. (See reports from the various groups.)

Let us conclude the discussion of Project MAC's research, therefore, with a brief word about computer networks. This is, in our view, a new field of very great potential. Project MAC has initiated a new research group to work on computer-network problems and techniques. We are looking forward to participation in the experimental ARPA network, which will link multi-access computers in several universities. A. K. Bhushan has already published two papers in the new field and is planning thesis research in it.

## Administration of Project MAC

Project MAC has a simple structure: about twelve research groups, two computer installations, a document room, a publications office, and a headquarters. It has a director, (now) two assistant directors, and an assistant-to-the-director who is the cement that holds the project together. Since Prof. R. M. Fano retired from the directorship at the end of the Summer of 1968, Prof. J. C. R. Licklider has been Director of Project MAC. Professor Jones and Mr. D. E. Burmaster (*vide infra*) are the Assistant Directors. Miss Dorothea C. Scanlon is the Assistant to the Director. In addition to conducting and publishing research, Project MAC holds seminars and participates in educational undertakings. This past year, there were, in addition to the standard, general Project MAC colloquia, seminars in theory of computation and theory of automata, and the one (mentioned earlier and only recently started) on data description languages. Project MAC initiated two "project laboratories" (laboratories conducted on a research-project basis that yield regular academic credits), one in software engineering and one

(joint with the Education Research Center and the Lincoln Laboratory) in computer graphics.

At the beginning of the Summer of 1969, Burmaster was appointed Assistant Director of Project MAC for Student Activities -- and also Business Manager. As an M.I.T. student, Burmaster was an effective activist for student involvement in research and student access to computers. With a few colleagues, he conducted the campaign that led to the formation and funding of the M.I.T. Student Information Processing Board and, inter alia, to the initiation of the Computer Graphics Projects Laboratory. Under his direction, Project MAC is now dedicating itself to a major increase in student (including undergraduate student) participation. The computer field is a "natural" for young people. They not only learn to understand computer systems faster and better than their elders; they can do significant original research soon after entering the field.

For the five years of its existence preceding this last year, although its terms of reference encouraged diversified support, Project MAC was funded exclusively by the Information Processing Techniques Branch of the Advanced Research Projects Agency (ARPA). This last year, while the Information Processing Techniques Branch of ARPA continued its major support at the established level, small or medium-sized additional research programs were undertaken under the support of the National Aeronautics and Space Administration (Extensible Languages, Prof. Evans; and a Laboratory for Research in Perception, Prof. Minsky), the National Library of Medicine (Features and Costs of Multi-Access Computer Services, Professors Licklider and Corbato), the Office of Naval Research (Interactive Problem Solving and Decision Making, Prof. Jones), and the Behavioral Sciences Branch of ARPA (Dynamic Modeling, Prof. Licklider). Where funding from these new sources (or, in the case of joint research efforts with other M.I.T. groups, other sources) is involved, note will be made to that effect in the reports that follow.

A final item: Project MAC contributed significantly to the planning and establishment of a new research project, the Cambridge Project, for computer analysis and modeling in the behavioral sciences. The Cambridge Project has been funded by the Behavioral Sciences Branch of ARPA through the Defense Supply Service of the Army. The Cambridge Project will improve and specialize interactive computer methods for use in behavioral-science research.

## COMPUTATION STRUCTURES

Prof. J. B. Dennis

R. A. Carpenter

H. M. Deitel

J. L. Gertz

I. G. Greif

P. G. Hebalkar

E. Klang

Prof. F. L. Luconi

M. J. Marcus

S. S. Patil

L. Rotenberg

L. Seligman

D. H. Vanderbilt

BLANK PAGE

# 1. COMPUTATION STRUCTURES

Objectives -- Jack B. Dennis

The Computation Structures Group seeks to develop and to study advanced concepts in the design and organization of general-purpose computer systems. We wish to understand the influence that technology, economics, programming languages, and the nature of computer use should have on the architecture of computer systems. Projects within the group often involve the invention and study of a mathematical model from which one can deduce conclusions having direct implications for the structure of programs and digital systems.

Current research by the Computation Structures Group falls into four main areas:

1) Representation of parallelism in programs and study of corresponding semantic theories of computation;

2) Structured information -- development of a formal model; basic primitive operations and their properties; the implications of security, privacy, and controlled sharing of procedures and data;

3) Architecture of computer processing and memory hardware;

4) Design and specification of digital systems, particularly in the form of interconnected asynchronous modules.

The eventual goal of building a general-purpose computer of radical, highly parallel architecture inspires and guides much of the group's research. The design of this machine would emphasize the independence of programs from the context of their application so that they might arbitrarily be combined without concern for their internal design. We have argued (Dennis, App. C) that this ability can be realized effectively only in machines that exploit the potential parallelism in algorithms.

Most of the individual research topics outlined below are starting formulations of problems for doctoral research. Although the topics have grown from the background of aims and ideas within the group, each study reflects the interest and judgment of its author.

Controlled Information Sharing in a Computer Utility -- Dean H. Vanderbilt

The development of multi-access computer systems through the principle of time-sharing has greatly enhanced the services that computer systems can offer their users. Large numbers of users are now able to have simultaneous on-line, interactive use of the system facilities, thus achieving orders-of-magnitude shorter response times than is possible with batch systems. These systems also provide facilities that enable the user to store his

programs and data within the system, rather than requiring him to submit them to the system for each use.

This ability to store information within the system has created the possibility for each user to make his work easily available to others. This is accomplished by allowing users to share information stored within the system. A user could specify with whom and in what manner information is to be shared, and other users could then make use of that information, within the specified restrictions, as easily as they could use their own.

These developments have led to a broadened concept of a computer utility: a multi-access system with facilities for general use by large numbers of people. In addition to making increased amounts of "computing power" available, the computer utility would be a vehicle through which users could make their work available to others in the user community. Therefore, provision of convenient means by which users may share, in a controlled way, information stored within the system is a primary objective in the development of the computer utility.

The present research is an attempt to understand more thoroughly the requirements that information-sharing places on the design of a computer utility. The approach is to specify a model of those parts of a computer utility that retain and access programs and data structures on behalf of users, and that control access to and sharing of these objects. Emphasis in the model is placed on the necessary logical organization of the stored information and the means provided to users for accessing and allowing other users to access the information. The following discussion illustrates the methods being employed to develop the model.

We accept the following principle as basic: Access to information is granted to a recipient user only as necessary to enable the accomplishing of sharing that has been explicitly allowed by a donor user. One obvious consequence of this minimum-access principle is that information that has not been specifically shared will be accessible only to the user who created it. There are other consequences which we will discuss.

Let us examine the types of information that it should be possible to share. First, it should be possible to share the use of programs. This is implicit in the concept of building on the work of others. There are two consequences of this idea. First, because of our minimum-access principle, it must be possible to permit the execution of a program without forfeiting knowledge of its structure. It seems clear that, unless this is possible, the idea of leasing software is not feasible. If it is necessary for the program user to be informed of the program's structure, this will be in effect purchasing the program rather than paying for its use.

The second consequence of sharing the use of a program is that it must be possible for the borrower to execute the program. Generally, the execution of a program will require the use of various data and the execution of other programs (subprograms). Some of this information will be supplied to the program by the caller. But the remainder will have been specified by the

creator of the program, and the system must ensure that it is made available during the program's execution. A common example of this second type of information is the subroutine library. In this case, the program creator specifies the subroutine by utilizing a special name for it. The system then recognizes this name, and makes the specified subroutine available whenever the program is executed. Usually, this information can be supplied by the system, or specially constructed by the program creator, or borrowed by him from some other user.

In each case, this additional information must be made available to the borrower of a program when the program is executed. However, in accordance with our guiding principle, this information should be associated with the main program so that the use of this ability is directly associated with execution of the main program.

The need to form these associations leads us to the concept of a procedure. For our purpose, a procedure is an entity of structured information consisting of first, a program, and second, a set of abilities to access other programs and data. These programs and data constitute the information, in addition to information supplied by the caller of the program, that may be required for the execution of the program.

Hierarchical Associative Memories -- Jeffrey L. Gertz

Recent work by Prof. Jack B. Dennis (Dennis, App. C) has indicated that two current trends in computing -- first, the increasing importance of parallelism in computer operations to improve hardware utilization, and second, the concept of programming generality which allows programs to be written that are independent of the hardware environment -- require a radical change in thinking about computer system architecture. In particular, the requirement that programs be allowed to transmit arbitrarily complex information structures as parameters to procedures with unknown storage requirements implies that one must use location-independent addressing. One possible manner of accomplishing this objective is to employ an associative memory.

In this research project, we are concerned with the study, analysis and design of a multi-level associative memory for a highly parallel computer system. With such a memory, when a processor needs a memory word (data or instruction), it presents a <u>key</u> to the memory system (rather than an address, as is presently the case) which uniquely specifies the word. The memory system then locates the word and brings it to the sphere of the processor.

Before one can propose detailed operation of the memory system, he must know two fundamental facts. First is the nature and representation of the information to be stored in the memory; second, the physical structure and organization of the units that comprise the memory system.

A program will consist of two parts: one or more <u>pure procedures</u>, and, for each of these, an associated <u>data area</u>. Each of these items will be organized and stored as an

information structure. In particular, each pure procedure will be represented as a precedence graph in the manner of Martin and Estrin (see Ref. 1 at the end of this section).

Each information structure will be represented in the memory by a collection of pointers. Traditionally, in addressable memories, downward pointers are used for the type of operations we shall consider. However, in this research, we shall present an implementation using upward pointers (from son to father). We shall show that, for an associative memory, this method is superior to those considered before. The physical memory will consist of several levels. Because of cost and size limitations, probably only the highest level will be truly associative -- the others will be traditional types. Therefore, we must consider the problem of making core and drum memories look associative. Possible methods are hash-coding (core memory) and associative queues (drum memory). We shall also examine the usefulness and effectiveness of using modular memories in this work.

Because the memory system will consist of several physical levels, the question of transfer of information between them arises. Others, especially Dr. Peter J. Denning (Denning, TR-50, App. B), studied this problem. We consider the possibility of using dynamic "pages" instead of the usual fixed pages, since this concept appears to fit in well with the use of information structures to represent programs. In this type of implementation, each page to be brought into a higher-level memory will be formed when it is demanded and it will not have a fixed size.

Finally, after the memory system is specified, we shall study it analytically. We shall examine the queueing characteristics of modular memory systems and inter-level memory queues. We shall simulate several parts of the proposed system to assess their performance. We shall compare the system with more conventional ones according to several performance criteria.

Modular Associative Memories -- Jeffrey L. Gertz

We have begun a study to analyze the behavior of a modular associative memory in a parallel processing system. Characteristics of the assumed system are: N statistically identical processes active at all times; sufficient processing capacity that no process need wait for instruction execution; and a common associative memory consisting of M ($\leq$ N) identical modules.

Each process is assigned one of the modules as its primary store, in the following circumstances:

1) When the process needs a word from memory, it looks first in its primary module;

2) When a word belonging to the process is brought into memory, it is placed in its primary module (at any time, every word in memory is assumed to be uniquely owned).

Statement 2, of course, is the reason for statement 1. With these conditions, we assume that, when a process makes a memory request, it finds the word in its primary store with probability p, in another module with probability q (due to inter-process sharing or use of common information), or in a lower-level memory with probability 1 - p - q. The values of p and q will, of course, be a function of the number of modules (larger M means smaller p), but remain fixed if the over-all size of the memory is also fixed. Finally, we assume there is a number $q_o$ that gives the fraction of memory requests by a process that represents sharing (and hence $q \leq q_o$).

We assume sequence of actions by any process is:

1) Generate a memory request.

2) Search its primary store for the word.

3) If unsuccessful, continue searching module-by-module for the word.

4) If still unsuccessful, locate the word in a lower-level memory.

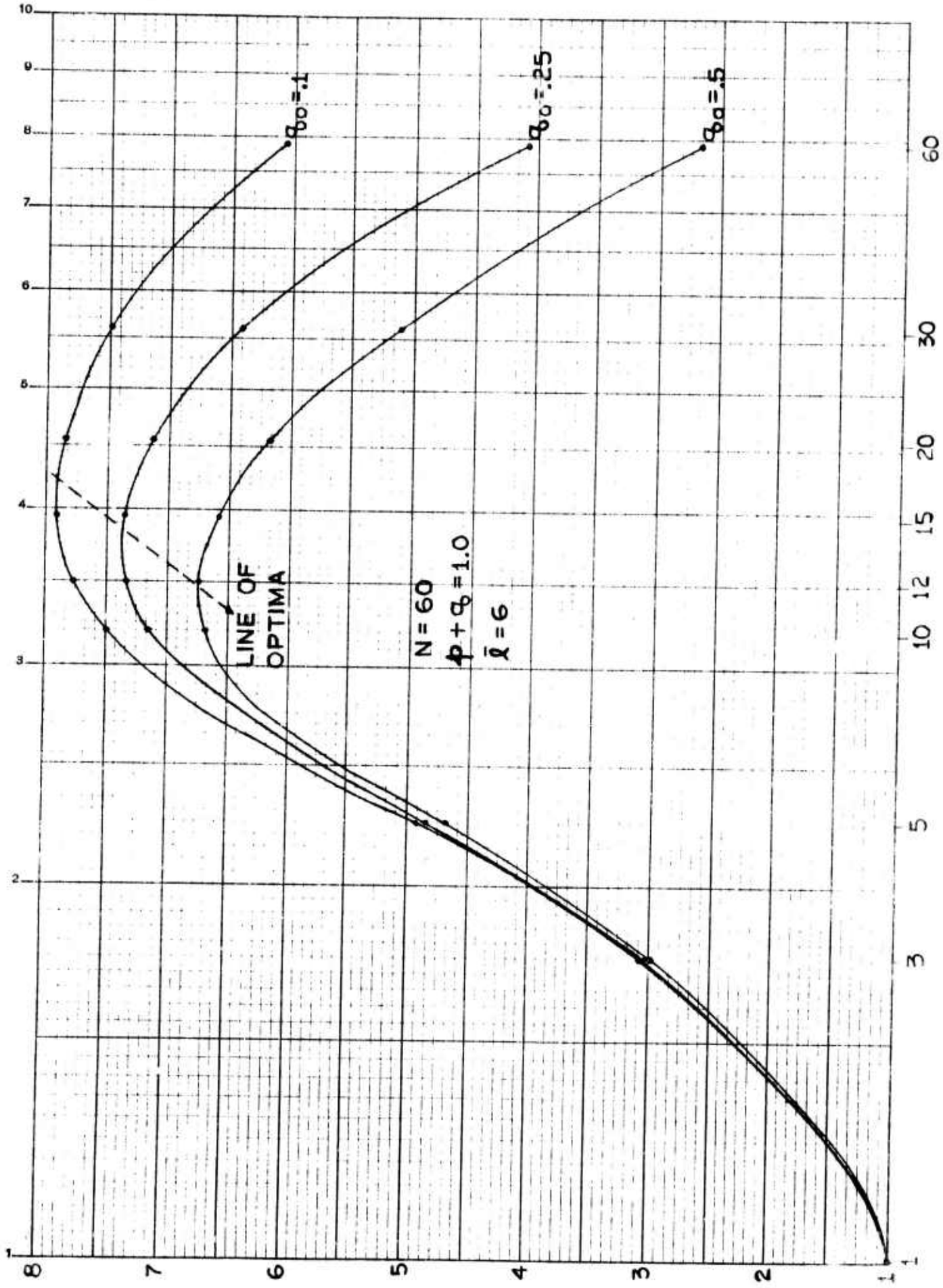
5) Enter a processing period, the length of which is a geometrically distributed random variable with mean $\frac{1}{\lambda}$.

6) Return to step 1.

Steps 2, 3, and 4 may involve waiting in queues.

We have developed an analytical solution of the above system -- giving the number of requests handled per memory cycle (R), average number of processes being processed at any time, and other parameters -- under the assumption that circulating requests are handled before new requests at any module. We have studied both methods of priority assignment with a simulation model.

We have found two preliminary results. First, several test runs with the simulation model appear to indicate that both priority assignments give the same rate of request-handling capacity. The circulating first algorithm results in fewer queues because there are no inter-module queues; it appears to be the superior method of operation. Second, an optimal number of memory modules, whose value is a function of various system parameters, is generally well below N. As one might expect, both this optimal value and the system through-put increase as inter-process sharing decreases. The figure gives a sample of this behavior for the values indicated. We hope to derive formulas that will allow determination of the optimum number of memory modules to use for determining the corresponding system through-put in terms of a few important system parameters.


$q_0 = .1$
$q_0 = .25$
$q_0 = .5$
LINE OF OPTIMA
N = 60
$p + q_0 = 1.0$
$\bar{\ell} = 6$
10
9
8
7
6
5
4
3
2
1
8
7
6
5
4
3
2
1
60
30
20
15
12
10
5
3
1

Digital Computer Organization -- Lawrence Seligman

We are studying organization of very-high-performance digital computers in the form of time-independent interconnections of asynchronous modules. Two goals are paramount: to demonstrate the suitability of asynchronous structure, and to develop corresponding analytic tools capable of characterizing system performance.

The existing very-high-speed processors are essentially synchronous designs. While some of these systems appear to employ asynchronous communication among the major components -- memory modules, processors and I/O channels -- each of these components is implemented as a synchronous system. The suitability of asynchronous logic -- for example, Prof. Fred L. Luconi's computational structures (Luconi, TR-49, App. B) -- seemed an open question at the beginning of this study. We have developed an asynchronous model for a very-high-speed processor which includes provision for representing multiple independent functional units and dynamic allocation of processor hardware for the storage of intermediate results. We are extending the model to include micro-programming facilities and hierarchical memory organizations.

The study of such models is fruitful because they are amenable to analytic techniques. Dr. Denning's recent work (Denning, TR-50, App. 3) on resource allocation has been adapted to study the performance of a hierarchical main memory system, eliminating the need for evaluation by extensive simulation.

Modular Design of Asynchronous Arbiters -- Suhas S. Patil and Jack B. Dennis

In computer systems, it is advantageous to arrange similar resources into pools from which units may be allocated as needed to serve on current request. For example, the processors of a multi-processing system are often pooled for use by many processes. An arbiter is a device that resolves potential conflicts in the allocation of resource units to meet nearly simultaneous requests. Synchronous arbiters are used in large processors such as the CDC 6600 (Control Data Corporation), but they are complex, intertwined with other mechanisms of the machine, and are therefore very difficult to design and debug. Asynchronous arbiters are frequently used to control access to multi-port memory modules. These are single-server arbiters, are not usually time-independent, and are not modular in their internal construction.

We are investigating the design of general arbiters that are time-independent in their operation and are of modular construction. We have completed a design for an n-server, m-user arbiter as the interconnection of a few types of basic asynchronous modules. Two of these module types are specific to the functional requirements of arbiters and may be viewed as additional basic modules for the design of generalized arbiters for situations in which an individual request may be for a combination of resource units, possibly of different species, rather than just a single unit of one type. The performance of

the arbiter will also be investigated.

The work of A. W. Holt and his colleagues (Ref. 2 below) has been very influential on this research: the use of "Petri nets" to represent asynchronous systems has been particularly convenient for the study of arbiters.

## References

1. D. F. Martin and G. Estrin, "Experiments on Models of Computations and Systems", IEEE Transactions on Electronic Computers, EC-16, 1, February 1967.

2. A. W. Holt, R. M. Shapiro, H. Saint and S. Warshall, "Final Report for the Information System Theory Project", Applied Data Research, Inc., February 1968.

## COMPUTER SYSTEM RESEARCH

Prof. F. J. Corbato

S. M. Adams
M. C. Burnham
R. H. Campbell
I. R. Campbell-Grant
J. H. Cecil
G. F. Clancy
D. D. Clark
R. C. Daley
C. P. Doyle
S. D. Dunten
M. N. Fateman
R. J. Feiertag
R. L. Gardner
C. Garman
Prof. R. M. Graham
J. M. Grochow
K. H. Hill
D. L. Jones
K. J. Martin
E. W. Meyer
S. Montgomery
S. C. Morr
N. I. Morris
S. Ohayon
Prof. E. I. Organick
M. A. Padlipshy
D. H. Randall
R. L. Rappaport
S. L. Rosenbaum
Prof. J. H. Saltzer
R. R. Schell
M. D. Schroeder
A. Sekino
T. Seymour
T. P. Skinner
W. Southworth
J. W. Spall
M. J. Spier
M. A. Stallings
R. H. Thomas
M. R. Thompson
T. H. VanVleck
C. A. Vogt
V. L. Voydock
M. E. Wantman
M. B. Weaver
M. W. Webber
S. H. Webber

Guests

N. Adelman
A. Sasaki

BLANK PAGE

## 2. COMPUTER SYSTEM RESEARCH

CTSS and Multics System Development -- Fernando J. Corbato

### Introduction

The research and early development plans of the Multics project were carried out jointly by members of the Bell Telephone Laboratories, the General Electric Company, and Project MAC. As Multics work has shifted into the final stages of implementation, BTL participation has diminished and, during the last six months of this period, the project has been joint between members of GE and Project MAC.

July 1968 through June 1969 was the key year for the Multics (Multiplexed Information and Computing Service) System. In June 1968, Multics supported fewer than eight users on a test-session basis. By July 1969, the system was supporting at least 18 users, doing heavy system programming work, on an around-the-clock basis. The group has established 1 October 1969 as the date for opening the system to use by the general M.I.T. user community. Again this year, the primary efforts of the group were aimed at increased functional capabilities for the system and for improved performance, in terms of both the number of users the system can accommodate and the responsive service each user gets.

The next main section constitutes a brief technical review of the steps involved this last year in bringing Multics to operational status.

### Functional Capabilities

DIM Benchmark. The group achieved what has been called the Demonstrable Initial Multics benchmark in October 1968. The DIM system was defined by its ability to support at least eight users for several hours at a time. Under DIM, group members could use the system for productive work (i.e., instead of merely allowing system testing, DIM allowed creation of new modules and debugging on the system itself). However, users still did compilation and assembly under the GECOS (batch) system. Additions made to the system in reaching the DIM benchmark included:

1) Error recovery from the "hardcore supervisor", so that one need not invariably "crash" the system when hardcore routines encounter difficulty;

2) An interim version of "multilevel storage", which allowed files to be on disc as well as in drum memory;

3) The system-shutdown facility, which allowed the running Multics system to be stopped and re-started in an orderly way;

4) An interim back-up facility, which dumps disc and drum (and permits subsequent *in toto* restoration).

LIM Benchmark. By January 1968, the group had achieved the Limited Initial Multics benchmark. LIM went beyond the DIM system in two basic ways: in performance, LIM supported 12 users; in functional capabilities, LIM permitted Multics development work to be (in principle) independent of CTSS (Compatible Time-Sharing System). Source files were created and edited under Multics, compiled under GECOS, and returned to Multics for testing and debugging. Although CTSS continued to be used for such work on a production basis, new Multics System Tapes (containing self-initializing new versions of the system) were generated from within Multics. Features in the LIM system include:

1) A basic User Control module, which performs login and logout functions and allows one to "quit" executing a process;

2) Performance improvements in the Command System, including a faster Shell (the command language interpreter) and Listener (which manages communication between a user's console and the rest of the Command System);

3) Performance improvements in the Traffic Controller, including a more efficient Interprocess Communication facility.

At the LIM benchmark, system testing and maintenance had been organized so that weekly new systems incorporated new development work and the fixes for recently analyzed bugs.

New File System. With the accomplishment of LIM, the system reached a point where formal definition of further benchmarks was inappropriate; "packages" of changes comprising discrete benchmark systems were no longer needed; independent functional areas came under intense scrutiny. As it has always been in Multics, the most important functional area was the File System. The group began in January to work for a complete re-evaluation and re-working of the extremely tightened coding and program flow modules which comprise the "New File System". Major goals were to speed up missing segment fault-handling, linkage fault-handling, new process creation, and general directory manipulation. The New File System effort adhered closely to schedule. The design was stabilized in January, firm schedules were set by early February, and by March a test system based on the New File System had reached command level (i.e., successfully completed system initialization). By May, the New File System had been integrated with the standard system. Its pre-installation check-out work was sufficiently thorough to eliminate the need to "back up" to a previous standard system because of post-installation difficulties.

Mini-GIM, Ring 0 Typewriter. The next major functional area is input/output. In early February, introduction of the Mini-GIM, a streamlined interface module, into the General I/O Controller (GIOC), considerably speeded up the typical I/O processing path. The mini-GIM, less elaborate than the full GIM (GIOC Interface Module), permits high-efficiency performance of basic I/O

operations. More elaborate operations can still be performed through the full GIM. With the mini-GIM, the "typewriter DIM" (Device Interface Module) -- the module that is the major link between the users' consoles and the system -- was moved into "ring 0" (the supervisor's protection ring). This further speeded up I/O because ring-0 modules tend to be paged out less frequently and because "wall-crossings" (i.e., changes of protection ring) are minimized. An even greater advantage is that the change requires only one input buffer for all typewriters, instead of one input buffer per typewriter.

CTSS Independence. Self-sufficiency is an important design capability of Multics. To achieve it, and thereby to become independent of CTSS, two requirements had to be met: First, Multics System Tapes (MST's) had to be generable within Multics itself. This implies working Multics facilities for magnetic tape I/O, for segment-binding, and for editing of MST's. Second, the source files for the entire system had to be conveyed from CTSS to Multics, an involved administrative and operational task. By the end of June, both were accomplished, and the only Multics work still on CTSS was in language development. It was left on CTSS simply to avoid overloading the Multics machine.

Backup and Salvager. The group developed and installed several important tools in the area of system reliability during the year. The crude back-up procedure of dumping the entire contents of drum and disc memory for in toto reloading was supplanted by an "incremental dumping" procedure. Within a few minutes after new files are created, they are written out on magnetic tape. If it is necessary to retrieve particular files, they are reloaded selectively. Incremental dumping greatly reduces work losses from system crashes.

The second major aid to reliability is a program called the "Salvager". After a system crash, most of the contents of the storage hierarchy are usually still intact and usable. Following an "emergency shutdown" program, the Salvager can inspect the hierarchy, effect repairs to files which were being read or written, and make it unnecessary to perform a complete reload. For those cases when it is necessary to perform a complete reload, a variant of the incremental dumper is available to do periodic complete dumping of the storage hierarchy. Thus reloading takes place on a per-segment basis instead of the per-storage-location basis implied by the entire-contents-dump approach. Reloading is therefore not dependent on the availability of the same hardware configuration that was used during dumping.

User Control, New IPC. User Control, with the associated Interprocess Communication (IPC) facility, is another major area in which there was considerable revision and functional expansion. User Control includes: 1) the Answering Service, which automatically responds to system dial-ups; 2) the login and logout modules; and 3) an automatic logout capability. Automatic logout is a particularly valuable feature; it allows the system load to be adjusted selectively and console sessions to be ended gracefully. IPC is important to numerous other areas of the system in addition to User Control, for it is the prescribed

means of allowing processes to communicate with one another (see also "Daemons", below). IPC was thoroughly re-designed and re-coded. The new package is much more efficient than the previous version.

Other Segments. Overhead in a Multics process can be considerably reduced if the number of segments it contains can be reduced (see Project MAC Progress Report V, 1967-1968, p. 39, regarding the desirability of binding object-code segments together). File directories would occupy less storage space if they contained fewer segments. Therefore, a plan to combine the previously independent object-code, linkage and symbol segments resulting from the translation of a given source-code segment into a single "object segment" was put into limited practice toward the end of June. The necessary changes were made to the system for it to produce and manipulate object segments and, although no massive change-over has been scheduled, object segments are propagating through the system as various modules are revised and recompiled.

"Daemons". A "daemon" process is a system (not a user) process, the operation of which is automatic. Several daemons have been developed during the year. Particularly helpful daemons installed were the "Output Driver Daemon" and the "EPL Daemon". The former accepts requests (via commands which employ the IPC facility) to perform card and printer I/O. Such requests are then stacked on to queues so that the demand for peripherals is smoothed out over time. The latter makes possible queued on-line execution of the EPL (Early PL/1) compiler and the EPLBSA assembler, an important advance because these are the current system implementation languages. (A full version of PL/1 will replace EPL; see "Other Efforts", below.) Execution is queued because of the drain on system resources which the current, interim languages represent. Thus, with only one compilation/assembly in progress on the system at a given time, other users are able to perform other useful work without degradation of system response.

Performance and Reliability

Multics supported fewer than eight users for a few hours in September and more than 18 in 24-hour-per-day operation by June. The "more than 18" users involves two considerations:

1) Because all the users are systems programmers who use the machine very heavily, the somewhat arbitrary limitation of 18 users was established to keep system response from becoming too sluggish. (The system programming tools -- in particular the EPL compiler -- are not so efficient as the programs that typical users would use.)

2) The hardware "data switch", which allows consoles to dial in to the system, imposes a temporary physical limitation on the number of users. Thus, although precise figures of system load under typical usage conditions are not yet

available, the over-all state of system performance is quite encouraging.

Performance improvements were effected in two ways. First was the application of programming techniques of re-design and enlightened re-coding (for discussion, see Project MAC Progress Report V, p. 40). All the major functional areas discussed above (as well as the numerous minor areas) were given the benefit of these programming techniques. The most basic area was the File System. The pay-off was quite high in this area, with the time required to process all functions improved by factors ranging from two to four over the final version of the "old" File System. Preparation for the New File System effort included definition of a subset of EPL for system programming. This subset specified those constructs in the language that should be employed to obtain the most efficient object. This "Restricted EPL" subset was adopted for all system programming except for those user interfaces where adjustable-length character-string manipulation was unavoidable. Recourse to hand-coding to further improve performance remains necessary only in a very few modules.

"Tuning" was the second major system improvement. Tuning is primarily alteration of various system parameters to achieve better usage of resources. Among the system parameters tuned were the value of the "quantum" or time-slice allotted to each process, the strategy of paging, the rules of eligibility to compete for the available processor(s), and the scheduling algorithm.

Members of the group developed and employed three tools to monitor performance as a basis for planning the tuning. Work on a Master's thesis research project involved initial development of a system on the PDP-8 display computer which allows on-line performance monitoring. A "Certifier", developed as a Multics command, creates a set number of processes, each of which executes commands from a file or "script" of typical commands. The Certifier allows comparison of the performance of successive systems, in both individual and over-all time consumption. Finally, a page-fault tracing tool allows deep analysis of paging behavior on a per-process or on a per-system basis.

Reliability. Multics went into operation three hours per day in September 1968. The user community was small because of poor system reliability and generally limited access. For the next three months, very little real work was accomplished solely on Multics.

By February, "mean time between failures" (hardware and software) had risen to two hours and the system was in operation about six hours per day. Users were beginning to exercise the system in a way no automatic testing could. Software bugs occurred less often and were more difficult to find and correct.

"Multics in Operation" rapidly increased to 24 hours per day during the week and over 12 hours per day on weekends. During the first week in April, Multics was in continuous operation without failure for over 30 hours before it was successfully

shut down. Mean time between failure varied widely, with an average of about four hours.

Since the beginning of May, Multics has been operating all the time the system could be kept "up", 24 hours per day, seven days per week. Installation of the New File System increased performance to the point where 12 system programmers could use the system with reasonable response -- and as many as 19 users were logged in at one time in June. (The limit was set by the availability of Dataphone equipment.) During June, the average number of users was between 10 and 12 for the hours 8:00 A.M. to midnight and between five and six for the hours midnight to 8:00 A.M. Mean time between failures still varied considerably but average was somewhere over eight hours. System usage per week is summarized in the adjacent figure.

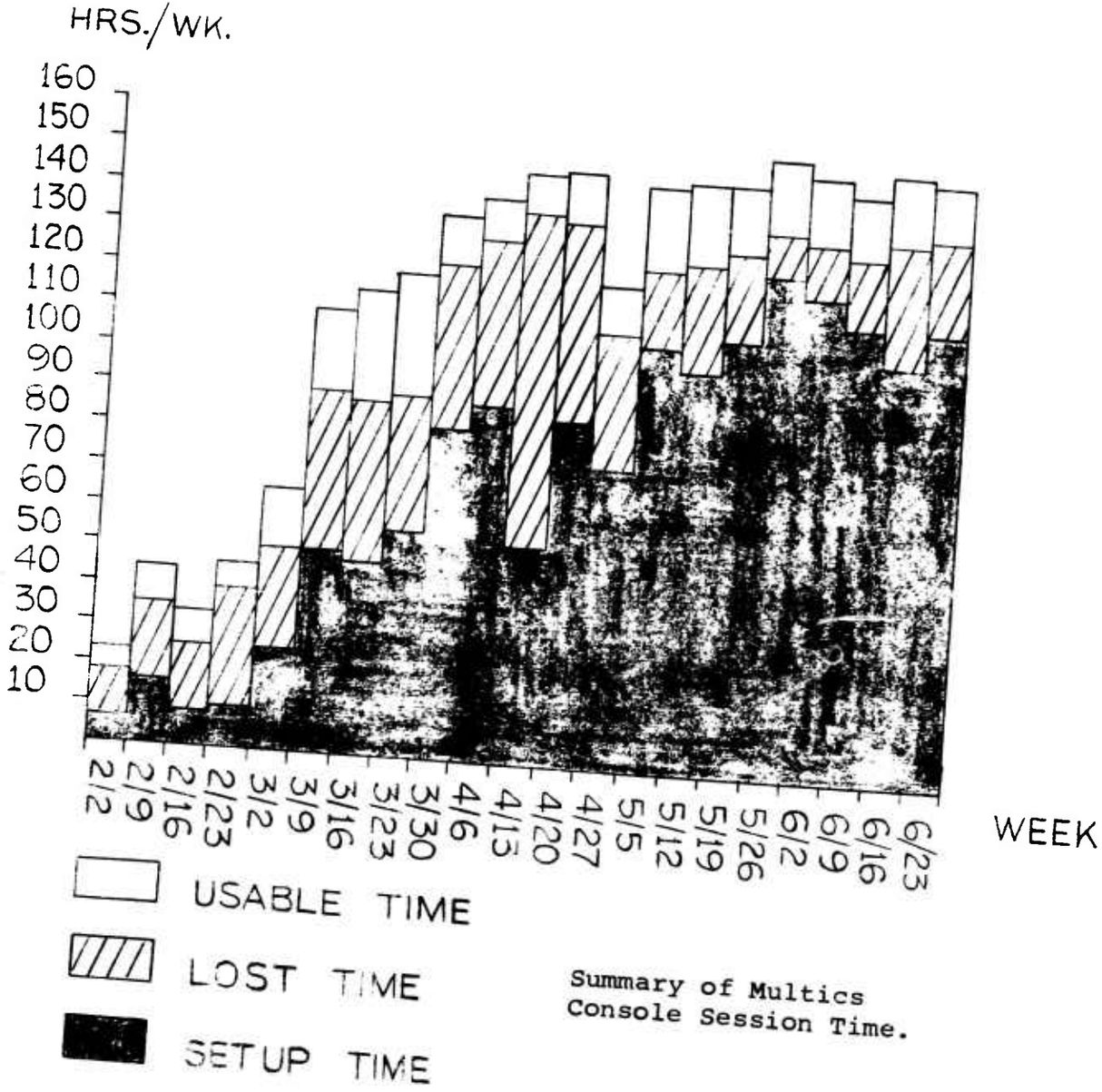
Other Efforts

Language development was the primary area of work not directly related to the main stream of development. Members of the Cambridge Information Systems Laboratory of General Electric worked on PL/1 and Fortran compilers. The PL/1 effort is particularly interesting because, when it becomes available, it will be able to recompile system programs written in EPL. Indications are that PL/1 recompilation can lead to as much as 50 per cent improvement in object-code length and to further performance improvement because of more efficient mechanisms for such things as character-string manipulation and structure-accessing. The compiler was also showing up well in terms of compilation time. Whether PL/1 will be available when the system goes public is uncertain; at worst it should be ready shortly thereafter. Fortran is expected initially to be the primary user language; it will be available in the initial public version of the system. Fortran has shown up well in compilation time and in object-code efficiency.

Another available language, BCPL, is a result of work at Bell Telephone Laboratories. BCPL, implemented on CTSS by its originator, Martin Richards, during his stay at Project MAC, is a language well-suited for the implementation of compilers. One compiler which has already been implemented in BCPL on Multics is PAL, used in an M.I.T. undergraduate course in programming linguistics.

Operations. During the initial stages of regular Multics operation, it was necessary to have a programmer available throughout the session. The frequency of crashes or of other trouble, the lack of firm procedures, and the necessity of catching every possible bug made it almost impossible to delegate the operation of Multics to the Computer Operations Staff.

For several reasons, training operators in both the operational and recovery procedures for Multics was not simple. First, as is to be expected in a research project, the "Operator Interface" was not given great attention during the early stages of Multics design. This area is now receiving programming support. Second, operational procedures changed frequently as a consequence of system improvements. As new modules replaced old, and as more




Summary of Multics
Console Session Time.

modules were added, operational procedures were changed to keep pace, and problems of communication resulted. Third, operational procedures remain numerous. However, this situation is eased because particular emphasis is being placed on increased feedback and communication among operators, programmers and users.

Successful operation of Multics requires that operators be aware of the condition of many variables and be current in their understanding of procedures. Several classes and many pages of written material have been provided to aid the Operations Staff. Nevertheless, this area will continue to receive attention since it critically affects the performance of the system as seen from the users' points of view.

Hardware

Introduction of production (as opposed to prototype) equipment was the major hardware development during the year. The necessary software changes went smoothly; no major problems were encountered during the change-over. There were, however, a few processor bugs; the drum required more frequent cleaning than had been anticipated; and unexplained hardware transients caused trouble for approximately a two-week period before they disappeared. Generally, however, the production hardware has performed satisfactorily.

Plans for the Future

To open Multics for use to the general M.I.T. user community by October 1969 is the main goal. Plans have evolved to streamline those aspects of the system of most concern to the general user, with the aim of making simple features of the system very inexpensive in execution time while retaining the generality and flexibility for which the system has been designed. A "Limited Service System" (a subset of the present service system) has been defined to accomplish this. The command language and the I/O System are the areas in which this system will differ most from the current system as used by the system programmers. The group has designed a "mini-Shell" that will process simple command lines very rapidly; the features of the full command language (i.e., the current Shell) will still be available to users who wish them: the mini-Shell will recognize a particular character in the first position of a command line as an indication to invoke the full Shell. In I/O, the streamlining is a redesign of the I/O switching complex to treat the currently designated default input and output streams as the only available streams unless the user takes action to define others, thus bypassing the switching machinery in the "limited" case. The final major enterprise regarding the Limited Service System is the definition and intensive streamlining and "polishing" of that subset of commands that constitutes the most frequently used group.

With the Complete and Limited versions, Multics will meet the needs of both large and small users.

The concern of the Computer System Research Group with Multics will not end with the completion of the Limited Service System. Efforts will still be required to refine the hardcore system and

to provide additional system features. Important areas of performance improvement include further tuning of the Traffic Controller, reduction of the number of segments (per process) required by the supervisor, and selective pre-paging when giving control of a processor to a process. As the system goes into use by the community, new commands and subsystems will be developed by users and made available for general employment.

# INTERACTIVE MANAGEMENT SYSTEMS

Prof. M. M. Jones

J. W. Alsop

Prof. J. J. Donovan

R. C. Goldstein

Prof. G. A. Gorry, Jr.

R. S. Green

Prof. D. N. Ness

L. K. Platzman

N. L. Ross

Prof. M. S. Scott-Morton

R. C. Thurber

H. M. Toong

D. M. Wells

Guest

H. Hegna

## 3. INTERACTIVE MANAGEMENT SYSTEMS

### MACAIMS -- An Interactive Management System

The MACAIMS project, initiated in June 1968, has two major goals: investigation of the value of a multiple-access computer, employing sophisticated interaction procedures, as an aid to management; and the specific application of such procedures to the management of Project MAC. The work is supported in part by ARPA funds through ONR and in part by ONR funds.

Business applications occupy the great majority of computers installed in the United States. However, there has thus far been surprisingly little use of interactive computing in management. The inability of most existing interactive computing facilities to handle economically the volume of data required for business purposes has been one reason for this. Another has been the requirement of most current systems that the user expend a significant amount of time and effort to learn to use the system.

In the near future, some time-sharing systems will have greatly enhanced data-storage capabilities. Thus it will be technically feasible to use these systems in interactive management. The question is whether systems can be devised that will permit the professional manager to interact with the computer usefully and conveniently. This is a question the MACAIMS project is asking.

We spent the Summer months of 1968 studying the existing Project MAC management procedures. Robert C. Goldstein surveyed available system-building tools. These studies suggested four applications areas for initial implementation: budgeting, purchasing, personnel, facilities. The last includes furniture and equipment as well as buildings and rooms. We decided to carry out the initial development work on the CTSS system because of its immediate availability. However, we anticipate that other computers will be used in later phases of the project. All programming for the MACAIMS system is being done in the AED language (see Electronic Systems Laboratory section) which offers both extremely sophisticated capabilities and a fairly high degree of machine independence.

After we studied the existing CTSS facilities for manipulating large data bases, we decided to create a system in which all data would at least appear to be resident in core at all times. This approach offers the maximum degree of flexibility for organizing the data, and it greatly facilitates updating the data. Because the various application areas have data requirements that vary considerably in size and structural complexity, we employ a variety of organizing strategies, from simple sequential files to partially inverted list structures.

The MACAIMS project intends to develop systems that can be used by people engaged in management activities who have no special computer training. This implies that the system should possess a rudimentary capability for natural language conversation and, in particular, that it should recognize a wide range of business-related terminology. Therefore, before we could begin

programming of applications, we had to develop a number of additional tools, primarily in the areas of highly interactive input-output, character-string manipulation and interpretation, and information retrieval. By late June 1969, Goldstein, Loren K. Platzman, Neil L. Ross, and Douglas M. Wells had essentially completed the development of these tools and had made a start on substantive MACAIMS programs.

We anticipate that initial working versions of some of the applications programs will be demonstrable in the Fall of 1969. Our further work will emphasize four objectives: completion of the initial system; extension to other management areas; improvement in the method of interaction with the user; and improvement in the efficiency of the programs.

Interactive Budgeting System

The Interactive Budgeting System (IBS) was the focus of our recent research (David N. Ness, App. A). We added a display capability that substantially improved the original system. In demonstrations, we used the ARDS (Advanced Remote Display Station) terminal at the Sloan School. Participants in the Senior Executive Program at that School used the system in an experimental "game", developed with Professors Wallace B. S. Crowston and Michael S. Scott-Morton.

IBS involves a business model. We have continued to expand the model to cover wider areas of managerial interst. Currently, we are adding a costing facility which incorporates Scott-Morton's work (Scott-Morton and Andrew J. McCosh, App. C).

During the year, we also made a preliminary study of the organization of binary trees grown with a sorting algorithm (William A. Martin and Ness, App. C).

Economies of Scale in Computer Use: Initial Tests and Implications for the Computer Utility -- Lee L. Selwyn

This study concerned economies of scale in the production of data-processing and other computing services, and the possible regulatory and public-policy implications of these economies.

The rapid development of the technology of computation since World War II has raised many questions about the supervision by public authorities of the use and progress of this technology. The Federal Communications Commission initiated a study in 1966 to consider the Commission's role in the production and distribution of computing services that involve communications facilities, supplied by regulated carriers. The present investigation concerns the production of computing services per se. The direction that public policy takes will be greatly dependent on the nature of the production of computing services, and perhaps secondarily on the interdependence between computer systems and the communications suppliers.

The relative economies of the use of large computing systems have been known for some time, in terms of the relationship between some measure of the quantity of output of a machine and its cost.

Indeed, this study demonstrated that when one considers, in addition to the cost of the computer hardware itself, the various categories of operating expenses associated with a computer installation, the relative advantages of large facilities become even more significant.

Yet the evidence would seem to indicate that, despite these apparent efficiencies of large systems, most installed computers are fairly small. In an attempt to determine whether, in actual experience, there are no true economies of large size, we made an analysis of data on nearly 10,000 computers installed at firms in manufacturing industries. Using the survival technique, which considers market experience as a basis for studying levels of optimum plant size, the analysis suggested that users did operate computers as if there were significant economies of scale in their use.

None of the evidence suggested that even the largest-size system available today is the most efficient possible size of "plant", hence the key implication for the formulation of regulatory policy toward the computer is that such policy should encourage, to the greatest possible extent, the shared use of large systems by those who require computing services. Those barriers that mitigate such shared use should be reduced or eliminated. Public-utility status would be indicated only if the costs associated with shared-computer use distribution -- software development, system overhead and administration -- are less than the potential direct savings resulting from use of large systems. This is at least as much a technological problem as it is regulatory one. The future of the computer-utility concept will thus be dependent upon the degree to which technology can reduce costs in these categories.

SIMULA

From September 1968 until June 1969, Havard Hegna, A Norwegian visitor, was associated with the group. Hegna had been involved with the implementation of the SIMULA simulation system developed at the Norwegian Computing Center in Oslo. As an experiment, he implemented a version of SIMULA on CTSS, using the AED Macro system.

## PROGRAMMING LINGUISTICS

Prof. R. M. Graham

A. Bagchi
W. D. Bilofsky
M. C. Bogue
R. L. Bushkoff
C. A. Dancy
F. L. DeRemer
Prof. J. J. Donovan
H. R. Drab, Jr.
R. S. Eanes
Prof. A. Evans
J. P. Haggerty
D. A. Henderson, Jr.
M. G. Hinchey
P. Hirsohn
J. W. Johnson
L. K. Lipman
R. F. Mabee
S. E. Madnick
P. S. Malek
R. Mandl
R. C. Moore
J. H. Morris, Jr.
J. E. Pinella
C. Ramchandani
H. A. Rideout
J. E. Sussman
R. C. Thurber
H. M. Toong
C. A. Vogt
Prof. J. M. Wozencraft
S. N. Zilles

# 4. PROGRAMMING LINGUISTICS

## Introduction -- Robert M. Graham

Programming linguistics is concerned with formal definition of programming languages and with translation between programming languages. Our definition of programming languages is a broad one. It includes languages ranging from machine language to pure declarative or descriptive languages (i.e., languages used to describe a set of functions and constraints, such as a set of differential equations and associated boundary conditions, rather than an algorithm for the solution of a problem).

The basic problem in defining a programming language is the precise, complete specification of the language's syntax and semantics. Several formal systems exist for defining programming languages. Comparison of these definitional systems exposes basic questions regarding their equivalence and hierarchical ordering in the classes of languages that they are capable of describing. Another basic question concerns the equivalence of programs that are written in the same language.

Discovery of a translation algorithm that uses directly the formal definition of either or both of the two languages involved is a basic problem in the construction of programming language translations. A "table driven" compiler is such a translator. Present translators of this type allow the implementer to specify his translation algorithm, using a special-purpose language (or languages). At present, only syntax analysis can be done automatically. Parsing algorithms can be generated automatically, given a formal description of the syntax of a language. A general translation algorithm would accept the definitions of two languages and synthesize a translator between the two languages.

Professor John J. Donovan and his students investigated problems in the formal definition of programming languages, using canonic systems, models of file systems, and comprehensive operating systems. Professor Arthur Evans, Jr. and his students studied (with Prof. John M. Wozencraft), problems of defining programming languages, using lambda-calculus, extensible languages, compiler theory and construction, and the PAL language. Professor Robert M. Graham and his students investigated the problems of a compiler specification system and of a software system design language.

## Educational Activities -- Robert M. Graham

Members of the Programming Linguistics Group continue to take an active role in course development at M.I.T. Although we shall not do so in connection with every group report, we may digress briefly to say a word about that work.

Evans and Wozencraft are in charge of 6.231 (Electrical Engineering, Programming Linguistics) which is the first of a three-course core sequence in Computer Science. Approximately 70 students took 6.231 last year.

Graham and Donovan are in charge of 6.251 (Electrical Engineering, Digital Computer Programming Systems), the basic course in computer software (covering assemblers, loaders, compilers, and time-sharing systems). This course, although it is not required in the Computer Science curriculum, has an enrollment of over 300 students each year. The course is continually being updated to reflect the latest software developments, such as the Multics system.

In the Spring term, Donovan initiated a new course, 6.686 (Electrical Engineering, Software Projects Laboratory). The intent of this course is to give students an opportunity to participate in the design, implementation and management of software projects that are significantly more complex than any project that can be assigned in the standard courses. The course was a great success, enthusiastically received by the students. The central theme of the course was the design and implementation of a comprehensive operating system for the 1130 computer. Approximately 35 students participated in this effort.

Graham completed the initial planning for a new course, 6.531 (Electrical Engineering, Principles of Programming Language Processors), which will be offered for the first time in the Fall of 1969. This course will cover advanced topics in the formal specification of programming languages, construction of programming language processors (principally compilers), and the application of programming linguistic theory to the construction of compilers. Students will be given the opportunity to build language processors, using a compiler specification system currently under development.

Software System Design Language -- Robert M. Graham

The inherent complexity of many current software systems is greatly magnified by the unavailability of any high-level language matched to the problem of the specification and description of software systems. The language should be such that the description of a system contains enough information that analysis and simulation of the system performance are straightforward and effective. We began in the Spring to study this problem. Our initial attack is the attempt to discover what primitives are really basic and common to all types of software systems (e.g., compiler-building systems as well as time-sharing systems). For example, table management seems to appear in every system and even in single software components such as compilers, assemblers and loaders. Should one then include, in a system specification language, facilities for describing tables and their management (i.e., specification of the contents of an entry, definition of the keys and search rules, and definition of the operations such as modification and deletion, desired for each key)? Perhaps such facilities are too high-level and specific and the solution instead is more powerful abstract structure manipulations.

David D. Clark, in his Doctoral thesis, is currently examining those aspects of a system that can be expressed as transformations between different address spaces. A large part of the Multics file system is concerned with mapping symbolic

segment (file) names into segment addresses usable by the hardware. This address mapping was the subject in a sequence of lectures Graham gave in Summer courses at the University of Michigan and at the Hebrew University.

Graham also discussed use of high-level language for system programming and specification at the NATO Working Conference on Software Engineering and at the Honeywell Second Annual Software Symposium.

Students also studied problems related to this project. Carla A. Vogt investigated the description and on-line maintenance of a computer system memory. Robert L. Bushkoff, in his Bachelor's thesis work, used a compiler specification system, currently being implemented, which specified the parsing, intermediate language, and symbol tables for a CIMPL compiler. CIMPL is a descendant of PL/1 designed by Graham and Prof. Jerome H. Saltzer; it is used in 6.233 (Electrical Engineering, Programming Linguistics) for system programming. Jeffrey R. Spirn's Bachelor's thesis was "A Simulation Model of a Large Multi-Processing Operating System". Michael J. Greata, for his Master's thesis, designed and implemented, on the PDP-8, a debugging aid which dynamically displays the sequence of subroutine calls made during the execution of a program. In addition, the system permits the user to stop the action and display the values of selected variables in the program.

A Compiler Specification System -- Robert M. Graham

In June, we began a project to implement a new compiler specification system. The system is being implemented in Multics with the aid of BCPL (Basic Combined Programming Language). Use of BCPL makes the system easily transferable to other computers. (BCPL compilers exist for at least seven other computers, including the 360.) However, we feel that a satisfactory solution to the problem of specification of code generation tables -- especially code optimization -- requires an on-line interactive environment.

The initial system design consists of processors for three specification languages: regular expressions for the specification of lexical analysis; reductions for the specification of the parsing and interpretation of the parse; and a macro language for the specification of code generation. The reductions language is based on earlier work by David D. Clark (App. A). A unique feature of the reductions language processor is that it compiles the reductions rather than interpreting them. The reductions are compiled into regular BCPL programs which are then compiled by the standard compiler. This permits a system user to write action routines (for interpreting the parse) in BCPL and to merge them easily with the reductions.

Canonic Systems -- John J Donovan

A canonic system is a simultaneous recursive definition of sets of strings on a finite alphabet. Canonic systems can be used as a formalism for specifying both the syntax of a programming language and its translation. Canonic systems are significantly

more powerful than the traditional Backus-Naur Form (BNF) formalism, because the latter is unable to specify context-sensitive features. By restricting the canons of the systems, a hierarchy of less powerful canonic systems can be defined which include correspondents for some of Chomsky's types of formal grammars.

In a Master's thesis research, Robert Mandl has developed an alternate method for presenting the theory of canonic systems. In addition, he has developed a new hierarchy of canonic systems which relates general canonic systems to all four types of formal grammars defined by Chomsky. He has also shown that all attempts to define a mathematical system that exactly corresponds to the recursive sets must fail.

Given a canonic system, C, it is possible to generate another canonic system, $C_m$, which is a proof measure function that is an indication of the complexity of the language defined by C. One can also generate a companion system which characterizes the recognition of strings generated by C. Joseph P. Haggerty, in a Master's thesis (App. A), was able to show that algebraic bounds on $C_m$ can be derived from the structure of the system C. He also established a relationship between the complexity of the recognition procedure and the complexity of the language description.

Hoo-Min D. Toong has developed a means for formalization of discrete-event simulation languages using the lambda-calculus and graph theory (App. A). Through application of lambda-operators to the graph theoretic representation of data structures, one can derive basic properties of systems modeled by these data structures.

File Systems -- John J. Donovan

During his thesis research, Stuart E. Madnick developed a model for file systems which isolates the basic function of file systems (App. A). This model is useful for comparing various implementations of file systems and in judging various design proposals.

1130 System -- John J. Donovan

In order to give students the opportunity to deal with the problems one encounters in the design and implementation of a comprehensive operating system, we began a project to design and implement a comprehensive operating system for the 1130. The system also provides a test bed for new ideas in compiler implementation. The basis of this project is student participation; the design and implementation are done entirely by students. Several Master's theses have grown out of this project.

Clifford R. Hollander's Master's thesis (App. A) was on the design of a multi-tasking feature which not only provides for a mixture of batch-processing and interactive tasks but also accommodates devices such as graphic displays and communications facilities. Charles A. Dancy is working on a COBOL compiler for

his Master's thesis, and Leon E. Travis on a COBOL interpreter for his Master's thesis.

Programming Linguistics -- Arthur Evans, Jr.

Completion of the notes for 6.231 (Electrical Engineering, Programming Linguistics) gave closure to a considerable amount of past conceptual research. Our understanding of the imperative constructs in PAL had finally reached the point where we could write the chapters on assignment statements and on transfer control. We expect to give a two-week M.I.T. special Summer Session in July which will cover all the programming linguistics material. This will be the first major presentation of this material to a non-M.I.T. audience.

In connection with the Summer course, we have implemented the language PAL (which plays an important role in the teaching of this material) on Multics. The students will use PAL to do exercises on the computer.

Extensible Languages -- Arthur Evans, Jr.

Support for our research in extensible languages comes in part from NASA's Electronic Research Center and in part from ARPA through ONR. Robert M. Thomas is working on a Ph.D. thesis in the area of syntactic extensions to programming languages. J. Dix Fulton did an imbedding of SLIP (Symmetric LISP Processing) in BCPL for his Master's thesis. This imbedding is unique; it was done entirely in the higher language, with no assembly code required at all. To illustrate this point, the same implementation was demonstrated as workable on both CTSS and the 360, with changes only to the data-describing declarations. Because the machines have different word lengths, this is a significant accomplishment.

Compiler Theory and Construction -- Arthur Evans, Jr.

Franklin L. DeRemer, in the abstract to his Doctoral dissertation "Practical Translators for LR(k) Languages" (App. A), states:

> A context-free syntactical translator (CFST) is a machine which defines a translation from one context-free language to another. A transduction grammar is a formal system based on a context-free grammar and it specifies a context-free syntactical translation. A simple suffix transduction grammar based on a context-free grammar which is LR(k) specifies a translation which can be defined by a deterministic push-down automation (DPDA).
>
> I present a method for automatically constructing CFSTs (DPDAs) from those simple suffix tranduction grammars which are based on the LR(k) grammars. The method is developed by first considering grammatical analysis from the string-manipulation viewpoint, then converting

the resulting string-manipulating algorithms to DPDAs, and finally considering translation from the automata-theoretic viewpoint.

The results are relevant to the automatic construction of compilers from formal specifications of programming languages. If the specifications are, at least in part, based on LR(k) grammars, then corresponding compilers can be constructed which are, in part, based on CFSTs.

Joseph W. Slater completed a Master's thesis on GENRAP, a system for generating automatically a parsing algorithm from a BNF description of a language. The method used was copied from a scheme of Cheatham's.

The BCPL Language -- Arthur Evans, Jr.

We brought the implementation of BCPL on the IBM 360 at the Information Processing Center to a consistent state so that the language could be exported to other installations. We expect to send out tapes during the Summer.

We are assuming responsibility for maintenance of the BCPL language on Multics. We plan to improve the interface between BCPL and the rest of the Multics environment to make it easier to write and run BCPL programs on Multics. We also hope to make significant improvements in the efficiency of the code compiled by the compiler. Because the compiler is written in BCPL, this will also improve the efficiency of the compiler.

The PAL Language -- Arthur Evans, Jr.

PAL has been implemented on Multics. Because PAL is written in BCPL and the Multics version of BCPL is compatible with that on CTSS, the entire effort took only about three man-months of work.

Marcus C. Bogue, in his Master's thesis research, completed and fully documented a student operating system on the 360. The system runs as a subsystem under OS/360, batching many student PAL jobs and making a significant saving of computer time. Although we have neither made nor contemplated any major changes in the PAL language, the Master's thesis of Stephen N. Zilles is a significant effort toward extension of PAL's data-description abilities. Results of this work may ultimately influence PAL.

## THEORY OF AUTOMATA

Prof. F. C. Hennie

M. E. Baker

V. Berardinelli

M. Edelberg

M. M. Hammer

D. J. Kfoury

Prof. Z. Kohavi

Prof. C. L. Liu

K. D. Venezia

C. Ying

## 5. THEORY OF AUTOMATA

Professor Chung L. Liu and his students studied several problems in the general area of abstract machine theory during the past year.

One of these was the algebraic structural theory of finite state machines. A finite state machine, viewed as an information transducer, can be characterized by lattice functions describing the relationship between the supply and demand of information at its input and output terminals. Using the lattice function characterization, they were able (generalizing the definition in the literature) to define the notion of pair algebras.

The second problem was the structural properties of a class of probabilistic finite state machines, known as definite machines. Liu's group established necessary and/or sufficient conditions for testing the definiteness of probabilistic finite state machines.

Finally, Liu's group was able to establish a synthesis procedure for a minimal nonlinear shift register which generates a given output sequence. (This problem is a generalization of the problem of synthesizing a minimal linear shift register.)

Professor Zvi S. Kohavi and his students investigated the problem of detecting and diagnosing failures in combinatorial logic circuits. They developed a procedure to determine "nearly minimal" sets of tests which detect single failures and locate them to within an equivalence class without resorting to a fault table.

## ELECTRONIC SYSTEMS LABORATORY

E. C. Anderson
R. Ascott
H. G. Baker
A. K. Bhushan
R. J. Bigelow
M. F. Brescia
L. M. Chui
F. Ciaramaglia
R. W. Cornew
Prof. M. L. Dertouzos
R. S. Eanes
C. G. Feldmann
J. Fiasconaro
H. L. Graham
K. Hatch
R. Hill
W. Hutchinson
D. Huy
D. Isaman
G. P. Jessel
P. Johansen
M. Kalisk
H. D. Levin
M. Lam
C. Lynn
R. P. Parkins
R. B. Polansky
R. A. Rausch
C. L. Reeve
D. J. Ross
J. R. Ross
T. Smith
R. Stinger
C. W. Therrien
D. E. Thornhill
K. VanBree
D. Vedder
A. Vezza
J. F. Walsh
J. E. Ward
T. G. Weston
R. B. Zara

### Guests

D. T. Cameron
J. T. Doherty
R. B. Gluckstern
R. J. McDowell

BLANK PAGE

# 6. ELECTRONIC SYSTEMS LABORATORY

## Introduction

The participation of the Electronic Systems Laboratory in Project MAC, begun in 1963, continued this year. Graphics Research (reported at the end of this section) and two other ESL projects had close working ties with, and received partial support from, Project MAC. One of these, the Computer-Aided Design Project (CADP) for the U.S. Air Force Materials Laboratory, Wright-Patterson Air Force Base, has developed the AED language and systems. The other, Prof. Michael L. Dertouzos's project for National Aeronautics and Space Administration (Electronics Research Center), worked with on-line simulation of networks and systems. Both projects used the Project MAC computer facilities and received direct Project MAC support of facilities and/or of graduate students.

## The AED Language

The M.I.T. Computer-Aided Design Project has been engaged since 1960 in research on the application of the concepts and techniques of modern data processing to the design of mechanical parts, as an extension of automatic programming APT (Automatic Programming Tool) systems for numerically controlled machine tools. Whereas part-programming is a relatively bounded domain which permits a single, standard APT program and language, the problem of designing large systems such as an aircraft is so complex that no one design program or language can be constructed that will serve all the varied needs. A very large number of design languages and programs is required, each tailored to a specific aspect of the over-all design process. Since, if traditional methods were used, the time and effort needed to construct each specific language and program and to make it available on computers of various types could equal that of the entire APT development, the project's major effort for the past several years has been the development of techniques for automating as much as possible of the process of constructing specialized languages and programs, and development of the process of moving programs from one computer to another. The result is the AED (Automated Engineering Design) family of programming systems including: first, the AED-1 system, whose domain is general programming, compiling and operating of programs on essentially any large-scale computer; second, the RWORD System, which builds a lexical processor; third, the AEDJR System, which builds a parsing processor; and fourth, the CADET (Computer-Aided Design Experimental Translator) System, aimed at a generalized approach to computer-aided design applications.

Major emphasis last year (Project MAC Progress Report V, p. 49) was on the subject of machine independence and the process required to convert the AED system programs to new and basically different computers through "bootstrapping". The bootstrapping procedure (described in detail in Project MAC Progress Report V) resulted in preliminary releases of AED for the IBM 360 and Univac 1108 computers in the Spring of 1968.

These systems were incomplete and differed not only from each other, but also from the M.I.T. 7094 system from which they were derived. The group's major effort during the past year has been a complete reworking of the AED system packages and bootstrapping procedures to produce a new, fully releasable Version 3 AED-1 system for the IBM 360 in both OS (Operating System -- batch) and CP/CMS (Control Program/Cambridge Monitor System -- time-sharing) versions. The Version 3 release is described in the next section. At the same time, the system was re-bootstrapped back to the M.I.T. CTSS on the IBM 7094, on which it was developed, so that all compiler versions on both machines would be identical. (The 7094 re-bootstrap is completed, but it has not yet been made operational. It, like other Version 3 releases, produces assembly language output, which requires a further assembly pass. Since the older binary output (Version 2) AED on the 7094 is more efficient, it is still being used. Version 3 bootstraps compatible to two other machines -- the Univac 1108 and the GE 645 (Multics) -- were partially completed, and are described in the second section below. The major effort made to improve AED documentation is described in the third section below.

With the release of the AED 360 system, both the Air Force funding for the Computer-Aided Design Project in ESL and the ARPA funding (through ONR) of AED work in Project MAC terminate. Douglas T. Ross and his key staff members leave M.I.T. in July to form a private company.

The AED/360 Release

This section is a summary, mainly in AED terms, of what constitutes the AED/360 system as released. Version 3 of AED-1 for IBM 360 computers consists of the AED-1 Compiler for the AED-0 Language, the AEDJR System for language definition, and the AED Library of system-building packages. (The RWORD System for setting up lexical processors, which was available in earlier 7094 and Univac 1108 releases, was not included in the July 1969 Version 3 release.) The release is on four reels of magnetic computer tape. The four tapes are divided into the following logical categories, with one tape for OS/360 users, two tapes for CP/CMS users, and one tape for both OS and CP/CMS users:

| Tape No. | Contents | Operating System |
|---|---|---|
| 1 | Source Programs | OS/360 and CP/CMS |
| 2 | Partitioned Data Sets for Running AED | OS/360 |
| 3 | Modules and Libraries for Running AED | CP/CMS |
| 4 | Text Files | CP/CMS |

Tape 1 contains the 540 source programs for the AED-1 Compiler, the AEDJR System, and the AED Subroutine Libraries. The tape also includes two simple test cases, one for the compiler and one for AEDJR, to check that no malfunctions (e.g., bad tapes) have occurred in the copy process. The total tape contains

approximately 64,000 card equivalents of EBCDIC (Extended Binary-Coded Decimal Interchange Code) source data written primarily in the AED-0 Language, with a few 360 Assembly Language, AEDJR, and RWORD System input-language programs.

Tape 2 contains the OS modules and libraries required to run AED on the 360 under OS. All modules are designed to work under PCP (Primary Control Program), MFT (Multiprogramming -- Fixed number of Tasks), and MVT (Multiprogramming -- Variable number of Tasks), but not under DOS (Disk Operating System). The eight files include: AED-1 Compiler, Basic AED Text Library, "Fast Free" Text Library, AEDJR System, AEDJR Text Library, AED-1 Macro Pass, TRACE Debugging Text Library and AED-1 Text Library.

Tape 3 contains the MODULE, TXTLIB, and MACLIB files of the CP/CMS version of AED. The tape is divided into four files: the AED-1 Compiler and its run-time support libraries; the AEDJR System; the AED-1 Macro Preprocessor; and the OSTOCMS (OS-to-CMS) utility, which permits the OS Source Tape contents to be read selectively into the user's file directory.

Tape 4 contains all text (object) files for the AED system; it is designed for CP/CMS only. Files include the AEDJR System and its libraries; the Macro Pass; all AED-1 Compiler programs; the "Fast Free" Package; and the TRACE Debugging Package. The tape also includes the SHOWIT AEDJR Example, described in the fourth section below.

Documentation of AED

Ross and other staff members are preparing, for publication in the Fall of 1969, an Introduction to Software Engineering with the AED-0 Language, a book based on a lecture series they gave in the M.I.T. Electrical Engineering Department's course 6.687, Software Engineering.

At the same time, Clarence G. Feldmann has completely revised and reworked the AED Programmer's Guide. This has been a loose-leaf collection of memoranda and system "flashes", but with this reworking it now conforms to the standards of the new Version 3 AED release.

By the end of June, both books were about 80 per cent completed, and both were scheduled to be finished by early September.

The SHOWIT System: An Example of the Use of the AED Approach

From the beginning, the CADP group has viewed "design" as only a special term for a type of problem-solving. However, the field of man-machine problem-solving is too broad to permit a single system to be used for all applications. Many systems are needed, and each must:

1) Use the specialized jargon of its particular field of application;

2) Require little or no knowledge of computer programming to be used effectively;

3) Be "evolutionary" to adapt to the changing needs of its users; and

4) Be created and maintained by the users themselves or by skilled staff in intimate contact with the users.

For these reasons, the project's efforts have been directed toward a system for making systems rather than a single computer-aided design system. What has evolved is a "system of systems for making systems" and an orderly method for applying it. The CADP group refers to this collection of concepts and working tools as the "AED approach".

The SHOWIT System, which originated as an unscheduled demonstration of AED capabilities at the Second AED Technical Meeting in January 1967, is a tutorial example of the application of the AED approach, with particular emphasis on the facilities of the AEDJR parsing for implementing new languages. The SHOWIT language is a subset of the Iverson language, chosen during the January 1967 AED Meeting.

By using AEDJR as the framework for the new system, the programmer can implement quickly all the procedures and grammar rules. In particular, the initial plateau of SHOWIT acts as a subsystem of AEDJR. By means of three successive commands to AEDJR, one can cause the following:

1) Reading in and making active the grammar rules for the entire SHOWIT language;

2) Invoking a special set-up procedure, written by the programmer; and

3) Executing any specific program, written in the SHOWIT language.

The lexical processor for the SHOWIT system is a specially constructed RWORD machine. Its item-building rules conform to the designer's specifications for the SHOWIT language. In particular, the lexical phase identifies and discards comments written by a user in his input message. The RWORD machine is invoked -- that is, a request is made for a new system to be extracted from the input stream -- by a special procedure the programmer writes to replace the standard one provided in AEDJR. The programmer's procedure is called by the First-Pass Algorithm each time a new input item is needed to continue the parse. A second RWORD machine is used in the SHOWIT system to read in data values which a user types on-line. This machine accepts numeric items written in integer, decimal, or E-type format.

The SHOWIT system has been documented by John R. Ross and Douglas T. Ross, in "The SHOWIT System: An Example of the Use of the AED Approach", ESL-TM-394, June 1969.

## Syntax Definition Facility

Robert S. Eanes, in his Master's thesis research (App. A), studied the language-definition problem. His work resulted in a system called the Syntax Definition Facility (SDF). This is an interactive system that allows the designer of a computer programming language to define the syntax of his language in a relatively simple natural meta-language. A set of tables for deriving a general parsing algorithm is produced from this syntax definition. If the system detects possible ambiguities or inconsistencies in the definition supplied by the language designer, it will report them and try to indicate the source of the problem. The system includes test and debugging facilities to aid the language designer.

The SDF meta-language allows the language designer to specify the syntax of his language by writing a series of sample statements which are marked to indicate how they should be parsed. Each sample statement specifies the "kind of value" or semantic type of a construction in the language. By the underlying principle of phase substitution, which allows any construction to be substituted for any other construction of the same semantic type, the small number of sample statements induces a complete language definition, allowing statements of arbitrary size and complexity. A syntax definition in the SDF meta-language is somewhat similar to a Backus-Naur Form of context-free grammar definition, but it is more readable and easier to produce. The algorithm used by the system to produce a parser from the meta-language description is a synthesis of the precedence techniques and the AED Language definition systems. The thesis will be issued as an ESL report.

## On-Line Simulation of Networks and Systems

The effective use of on-line computer utilities in the design of electrical networks and systems is the main objective of this research. This includes studies in the mathematical foundations of computer-oriented network and system analysis and in the interactive features essential for the design of networks and systems.

Work in these areas has proceeded along several directions. Charles W. Therrien and Huber L. Graham completed their doctoral dissertations: the former on network tearing, the latter on a new recursive approach for computer analysis of non-linear networks (App. A). The group's work on CIRCAL-II, a general-purpose network-analysis program, continued with the development of: a "pseudo user" (a user-defined program for the automatic optimization of networks); a method for dynamically loading and unloading program sections; implementation of non-linear transient, symbolic-frequency, linear-time and recursive-analysis techniques for CIRCAL-II; and development of a general-function and functional capability for CIRCAL-II. In computer-aided system design, the group completed the initial version of LOTUS, a block-diagram analyzer. It also initiated implicit computation and continuous-discrete systems, with the objective of developing effective computing techniques for the solution of linear and non-linear systems of equations.

### Tearing of Networks

Therrien has continued work on the use of "tearing" for reducing computation and for expediting the computer solution of electrical networks. His thesis discusses the question, "When and how should a network be torn in order to minimize the computation necessary for solution of that network?" He has provided several answers to this question. They involve use of a model for tearing, several deterministic tearing algorithms, and one statistical algorithm which achieves statistically predictable computational savings over a class of networks.

### A Recursive Approach to Network Analysis

Graham describes in his thesis a new technique for the direct analysis of non-linear networks through recursive decomposition of their network graphs. This approach differs from conventional approaches that use iteration techniques. Using the new technique, one constructs a non-linear function relating the given excitation to the required response variables. The construction of this function is recursive, and it is in one-to-one correspondence with the steps leading to recursive decomposition of the network graph.

### CIRCAL-II System Developments

The CIRCAL-II program is a general-purpose, on-line, circuit-design program. (For detailed discussion, see Project MAC Progress Report V, p. 60.) The main program, which became operational last year, performs all the necessary "overhead" tasks in computer-aided network analysis; it has convenient "plugs" into which various analysis techniques can be connected to enlarge its capabilities. There have been four developments in the main system of CIRCAL-II during the year:

1) A preliminary investigation and implementation of the pseudo-user or Defined-Command Feature. This feature substitutes for the real user a user-specified program which "observes" analysis results and makes appropriate modifications to the network so as to optimize performance. This system feature has been used, for example, to design an oscillator circuit by automatically adjusting circuit parameters so that the oscillation will be at a pre-specified frequency.

2) A method for controlling the dynamic loading and unloading of program sections in order to maximize available storage.

3) Extension of the definitional capabilities of CIRCAL-II, including the results of (1) above. This includes a generalized capability for defining functions and functionals. Functions are used in CIRCAL-II for non-linear element characteristics, for source waveforms, and for "post-processing" of computed network variables. Functionals are used in handling hysteresis, thermal effects, and other phenomena that exhibit memory.

4) Preparation of CIRCAL-II users' and programmers' manuals.

New CIRCAL-II Analysis Techniques

During the past year, the group began development of several new analysis techniques. The corresponding tasks involve implementation of: first, the recursive-analysis technique, the theory of which has been developed, as discussed above; second, a symbolic frequency-analysis technique for large, linear, sparse networks; third, linear-time techniques; fourth, non-linear transient-analysis techniques. The symbolic-frequency analysis is of the so-called compiler type, which requires little execution time and is especially suited to large, sparse networks.

Computer-Aided System Design

The group's work in computer-aided system design concentrated on three main areas: 1) completion of the development of LOTUS, a program for the on-line simulation of block-diagram systems; 2) synthesis of continuous systems; and 3) implicit computation.

The latter involves research on synthesis tools similar to those used in the synthesis of discrete finite-state machines. It entails a new approach to the organization and structure of computing hardware, capable of solving implicit algebraic and differential equations.

Graphics Research

The ESL Display Group continued its research on hardware and software techniques for graphic interaction in a time-sharing environment. In past years, with support from the Air Force-sponsored Computer-Aided Design Project and Project MAC, the group developed the ESL Display, a graphics display with special hardware capabilities for picture rotation and scaling, and the ARDS (Advanced Remote Display Station), a storage-tube graphics terminal for remote operation via a telephone-line connection. The group's work this year dealt with remote operation of the ESL Display Console, computer interfaces to support additional ARDS units, a hard-copy facility for ARDS, and graphics-support software.

ESL Console

During the year, the Project MAC 7094 computer was# moved to the M.I.T. Information Processing Center, and the former IPC 7094 was retired. Since the present implementation of the ESL Display and its PDP-7 buffer computer requires direct connection to a 7094 data channel, they also were moved from Project MAC and made operational in the new location. At the same time, the second ESL Display and PDP-9 buffer computer, which were to have been used remotely in ESL, were moved to Project MAC to maintain graphics capability there. In late May, the 50-kilobit telephone connection between the PDP-7 and PDP-9 was installed, and check-out of the link and associated communications software began. By the end of June, communications between the PDP-9 and

PDP-7 were established, and the remainder of the program modules for remote-display operation were being checked out. Study continues on means for automatic start-up from the remote location. This remote start-up requires that both the PDP-9 and PDP-7 be running and that they contain at least the bootstrap communications modules for program loading from the 7094.

During the Spring, a Sylvania Data Tablet was incorporated into one of the two consoles of the ESL Display at Project MAC. An interface was constructed to permit reading the tablet output directly into the PDP-9 via input-output transfer instructions. A character-recognition program, which will be the first use of the tablet, is being written.

Eight ARDS units are now used, primarily with the CTSS 7094 computer. That number posed a problem, a shortage of computer ports. The 7750 communications interface of the 7094 will support a total of four "high-speed" (1200 bps) lines; only two such lines were operational for ARDS use at the start of the year. An additional port, implemented during the year, brought the total to three; and there was progress in converting the present dedicated use of the remaining port to ARDS compatibility.

Looking beyond this four-port capability is a study of an information concentrator for the 7094 which would provide up to 15 ARDS ports, plus 50-kilobit connections to the two ESL Displays and perhaps to the Interface Message Processor of the ARPA computer network. The concentrator would be a small computer interfaced to the 7094 Direct Data Device (DDD), providing 36-bit word transfers at up to 170,000 words per second. The PDP-7 buffer computer for the ESL Display, which presently occupies the DDD interface, would then be connected to a 50-kilobit serial port of the concentrator. Several small computers have been evaluated for use as concentrators. The general system configuration and buffer memory requirements have been determined.

The System/360 data adapters for USASCII (USA Standard Code for Information Interchange) interfaces operate only at Teletype speed (110 bps). Thus, at M.I.T. those ARDSs used with the 360/67 computer must operate at this low speed. The group has assisted the IPC to plan for alleviation of this problem by acquiring a special data-adapter unit to connect to a 360 multiplexer and to provide a number of 1200-bps (or higher) ARDS ports.

The successful operation of an ARDS for two weeks, 22 August to 3 September 1968, at the M.I.T./Technical University of Berlin Conference, was a highlight of the year. International cooperation established a 1200-bps circuit via a dial-up connection from Berlin to Frankfurt with an ITT Datatel connection to New York, and a dedicated AT&T line from New York to the Project MAC 7094. There were several operators to deal with, not all of whom spoke English. Except for problems in re-establishing the circuit each time it was to be used, however, operation was trouble-free over many hours of use.

ARDS Hard-Copy Device

The experimental hard-copy station discussed in Project MAC Progress Report V was assembled and tested. This station was designed by Albert Vezza. It consists of: a Tektronix 611 storage monitor driven by an ARDS display; a modified 3M dry-silver paper print unit; a lens to image the display screen at 1:1 on the recording paper; and a shutter mechanism. Modifications made to the print unit increased the temperature of the heated development cylinder to process photographically a new Kodak dry-silver-process paper that has a photographic speed eight to twelve times faster than the paper for which the print unit was designed. With that paper, heating constitutes the entire development process. The new Kodak paper is first stabilized by heating to a temperature of 240 degrees; it is then exposed to ultraviolet light to bring out a visible image. Total development time is 10 seconds.

With the increased speed available in the Kodak paper, it is possible to make good-density prints with five-second exposures from the storage-tube screen, compared to the 30-50 seconds required for the old paper. The wide-angle-to-wide-angle lens used to obtain 1:1 imaging of the 6-1/2" x 8-1/2" display screen consists of two aerial camera lenses back-to-back. It has an equivalent f-rating of 2.25. Lens cost in quantities of 10 or more would be about $300.

In June, several problems remained concerning operational use of the hard-copy terminal. One problem is the heat generated by the development cylinder if the unit is left on continuously, ready to print at any time. Another is devising a satisfactory method for integrating the hard-copy terminal into the time-sharing system. One method for this would be to slave the hard-copy terminal to an ARDS when prints were to be made, but this would require either providing a local operator or having the hard-copy terminal continuously dialed-in through one of the already-scarce ports. Another method would be to provide a capability in the time-sharing system for saving output pictures on disc for batch-printing at scheduled times. The group is studying these problems and hopes to have at least limited operational use of the hard-copy terminal in the next few months. The hard-copy terminal should provide a remote hard-copy capability at a capital cost of about 50 per cent greater than a basic ARDS. Paper cost compares with that of office copiers.

Graphics Software

The GRAPHSYS software support system provides a convenient, high-level and nearly display-independent interface between user programs and the ESL Display Console or the ARDS. During the year, Daniel E. Thornhill and Christopher Reeve made a number of changes to improve compatibility between the ESL console procedures and the ARDS procedures. Reeve's Master's thesis (App. A) described procedures for interactive graphics on the ARDS. In December, ESL and Project MAC issued a technical report (MAC TR-56, App. B) describing GRAPHSYS. Several student projects contributed to the augmentation of GRAPHSYS capabilities. Some are discussed below. To make graphics more

easily available to the general community of time-sharing users, the group formulated plans for a coherent collection of graphics programs which would be available to all users. It began collecting existing programs and set up an on-line documentation system, using the TIP information retrieval subsystem of CTSS. Members started on the following procedures for inclusion in the common file:

1) A data-plotting package for plotting data on rectilinear, semi-log, log-log, and polar coordinates, and a facility to plot bar graphs;

2) A contour-plotting package;

3) A generalized light-button package;

4) A graphical display and simulation package for digital logic;

5) Procedures for dumping data structures on to a disc file and for retrieving them from a disc file.

Other Hardware Developments

Robert J. Ascott, in his thesis project (App. A), modified a low-cost resistance-paper (Teledeltos) data tablet (which F. F. Blount had developed earlier) so that it operates with a-c rather than d-c excitation on the tablet. Although the original tablet (see Progress Report IV, p. 92) worked well, with extremely simple electronics, it suffered from the requirement for electrical contact between the stylus and the tablet surface, and that prevented tracing an already-drawn figure. With a-c excitation, the stylus coupling is capacitive, and tracking can be accomplished through a sheet of paper.

In another thesis project, James G. Fiasconaro is adapting ARDS circuit techniques to add character and vector-generation capabilities to an existing point-plotting display for the PDP-9 computer of the Speech Synthesis Group at the Research Laboratory of Electronics. This will provide a refreshed graphic display of moderate performance at a cost considerably lower than comparable commercially available units.

In connection with the high-speed digital link between the PDP-7 and PDP-9, Thomas L. Smith, carried out a thesis study (App. A) of: trade-offs between transmission speed and CPU time required for on-the-fly error checking; error recovery procedures; and optimal lengths for message blocks. The results of this study indicated that the present 50-kilobit speed is adequate, that the 230.4-kilobit speed initially specified for the PDP-7/PDP-9 link is unnecessarily high.

## TECHNICAL INFORMATION PROGRAM (TIP)

T. F. Dempsey

D. M. Jordan

M. M. Kessler

W. D. Mathews

L. H. Morton

W. I. Nissen, Jr.

BLANK PAGE

## 7. TECHNICAL INFORMATION PROGRAM

Work on the Technical Information Project (TIP) has been carried out in the M. I. T. Libraries, with support mainly from the National Science Foundation in the early years, now with support from many sources within and outside M. I. T. Currently, several specific tasks are funded by Project MAC. Since this year marked the termination of the development effort related to CTSS, it seems appropriate to us at this time to give some perspective on the advances made during the five years with CTSS, along with our review of current progress. Our report deals mainly with program development, and with construction of software systems or subsystems. It deals also, briefly, with education, and with training of students and representatives of industry. We shall not describe, here, our work in system evaluation, data representation, and applications.

### Program Development

The first set of programs we developed at TIP, in mid-1964, was a loose set of on-line retrieval programs, each tailored to a specific point-to-point information-dissemination need. We created separate programs for retrieving information on titles, authors and citations in bibliographic data pertaining to the physics literature. At this early stage in our development, the deeper implications of an on-line operating system were not apparent to us. Instead, we dealt with CTSS as though it had merely made traditional batch-processing machine capabilities more available. The types of questions and the modes of interaction had been thought about in advance. New avenues were closed to the user. Our mistake became apparent on CTSS, where users were trained to be creative and divergent in their thinking patterns.

SHARE -- William D. Mathews:-- In late 1964, we implemented software, using the phenomenon of bibliographic coupling as a retrieval criterion. Physics articles that exhibited similarity in their pattern of citations to other articles could be retrieved efficiently by the computer. Although programmers had tried this earlier in an off-line mode, the ability to enlarge the scope of the search and to alter the search criterion brought new retrieval possibilities to light.

COUNTB and INDEXB -- William D. Mathews: -- We then, in mid-1965, developed programs to count and index bibliographic citations from one physics journal to another. This attempt to make available an on-line citation index brought to our attention a mismatch between technically feasible and operationally useful processes. While the idea of a citation index is attractive to the professional indexer and the librarian as a research tool, it is a very unattractive product to the scientific researcher. Retrieval questions are seldom posed in terms of citation characteristics, even though the growth of the scientific literature depends strongly on knowledge of previous research.

TIP/1 -- William D. Mathews: -- In the Autumn of 1965, we developed a unified retrieval program for the physics literature. In this one package, it was possible logically to combine

requests for papers pertaining to specific titles, authors and citations. We made this retrieval capability public to all CTSS users. Any CTSS user could interrogate a body of literature of approximately 30,000 recent articles in physics.

TIP/II -- William D. Mathews:-- It soon became apparent that we needed a more generalized information-retrieval capability to serve the scientist adequately. Unlike the physics literature, much scientific information is not neatly packaged and published. One cannot expect to get information from a library-maintained data base. Invariably, the answer must be organized and structured into different forms for a final intelligent presentation. Early in 1966, we began work on the TIP/II system. This involved considering retrieval as a general text-management process in which the user, with items of information containing ASCII text streams, approaches the computer and requests the partition of these items into groups satisfying very complex selection criteria. Further, a substantial part of the interactive vocabulary, especially the naming of fields of information in the user's data, should be under his control. From our experience, it was also evident that control over the detailed format of presentation of selected items should similarly be under user control. Thus design for the TIP/II retrieval system evolved. This system has since been used on fiscal, personnel and inventory information, as well as on bibliographic data.

TIP/II Components -- William D. Mathews:-- We had to develop a range of component packages to support the TIP information system on CTSS. We produced a powerful string package to allow quick manipulation and operations on ASCII character strings, and we introduced a list package for maintaining lists and dictionaries of attributes. We also developed an efficient disk I/O package to handle ASCII files. Other important component packages supplied functions for recursion, free-storage manipulation, and data conversion. These components formed the building blocks for nearly 30 file-manipulating subsystems that became known collectively as the TIP System.

SORT and MERGE -- William D. Mathews:-- The TIP/II retrieval subsystem allowed the user to divert information into a file for later processing. We developed a battery of subsystems in June 1967 to permit organization, deletion, extraction and updating of these files on an extensive basis. We introduced other subsystems to tally or count the number of similar fields in a data collection and to produce a new file reflecting this summarized information.

EDIT/I and QEDIT -- Lewis H. Morton: --Insuring reliability and accuracy was one of the most pressing problems facing us as the size of our data base increased. In mid-1967, we developed EDIT/I as a programmable editor with a large set of text-handling requests. It can be used for validation of data or transformation of information from one format to another where some syntactical analysis is required. In 1968, we introduced QEDIT. This is a quick ASCII editor that takes its requests from a file and writes a report of all editing changes into another file. The saving in machine time over the conventional CTSS file

editors ranges from a factor of two to a factor of five. The design of an EDIT/II subsystem is now well under way. EDIT/II will compile code if the user wishes to run an editing procedure repetitively. Other features include the ability to slave one pointer to another in such a way that operations performed on one pointer affect the contents of another.

FIBCHK -- Timothy F. Dempsey, Jr.: -- In the Autumn of 1968, we developed a subsystem which, with certain protocols in construction of RUNCOM's, allows for the convenient running of foreground-initiated background (FIB) jobs. Some of the problems that had to be approached were:

1) The chaining of several FIB jobs into one controlled job;
2) Prevention of FIB job-looping in jobs which establish other jobs;
3) Prevention of destructive interactions between chains of FIB jobs working in the same directories.

ZOT -- Lewis H. Morton: -- We developed a general-purpose environmental subsystem for use within command chains. Used in RUNCOM's, the ZOT subsystem can test to see, for example, if there is enough space left to write a file of specified length, or enough time left on a shift to complete a certain operation. Special action may be taken if the specified condition is not fulfilled. ZOT may also be used on-line as a desk calculator.

TAP and ASEMBL -- Walter I. Nissen, Jr.: -- We have made two advances toward a flexible and powerful text-oriented assembler for a modular programming system. The first is the development of the TIP Assembly Program (TAP), which is a derivative language of the IBM 7094 FAP macro language. TAP includes a number of standard functions, implemented as macro definitions, to provide, among other features, automatic subroutine linkage, table and list manipulation functions, symbolic indexing, and automatic recursion. Other pertinent features of the language are additional machine operations for pointer manipulation and logical operations, and the ability to treat standard subroutines and their calling sequences as machine operations. Initial experience with this assembly program has produced an exclusively favorable reaction. One can write significant programs with a very small investment of programmer time. This capability is largely a result of the naturalness of expression in the new language.

A parallel development has eliminated some of the major difficulties in the assembly of text functions. ASEMBL is a text-processing language which functionally can be considered as the front end for a standard assembler, either FAP or TAP. ASEMBL accepts as input any TIP text and produces as output, according to an accompanying field table, an FAP source file containing ASCII coded text strings with whatever hierarchical text-pointer structure one desires. There is provision for the utilization of all standard TAP machine operations and pseudo-operations, and, generally, for complete program structure. It is possible to provide a single source file for numerous distinct programs or for various versions of the same

program, and to allow the assembly of any subset of the total source by use of the appropriate field table.

SHARE -- Timothy F. Dempsey, Jr.:-- In the first half of 1969, we extended the concept of citation sharing, which had been used as a retrieval criterion, to cover sharing of other attributes. Further generalization permits the specification of a coupling strength and the number of items in a file with which selected items must share. The linkage information is incorporated into an output file for further iteration of the process. Now, for example, one may select items in a personnel file using similarity of attributes to some sample item.

The RUN System -- Walter I. Nissen, Jr.: -- The RUN system has passed through several evolutionary stages in the past year. The program modules have been completely recoded to increase their efficiency and to broaden their power considerably. The number of programs available through RUN has increased enormously. As a result of its expanded usefulness, it has been installed as a CTSS command.

By reducing the coding in RUN SAVED down to a dispatcher and an internal table, we have made RUN SAVED small and efficient. At the same time, we added the capability to dispatch to programs in independent SAVED files, not just in the special RUN files. And if RUN cannot locate a desired program in a RUN SAVED file or in the TIP Program library, it will search the public file, the user's directory and, finally, the CTSS command directory to find the desired program. Therefore the RUN system enables the inexperienced user to have access to virtually all commands or programs that he might wish to use; the familiar user can avoid the necessity of noting each trivial change in the organization of SAVED files that might previously have affected his method of access. We have increased the efficiency of core-storage management by RUN by developing an algorithm to identify those subroutines that are loaded for a particular group of "main" programs and to return to free storage any such subroutines when any other "main" program in the same SAVED file is referenced. One "main" program may call another, but any "main" program not called will also be returned to free storage just before execution begins. An unexpected offshoot of the development of the RUN system has been the capability to organize the TIP library of programs so that all programs are easily loaded and updated at any time.

PAPER TAPE INPUT AND OUTPUT -- David M. Jordan: -- We have extended TIP subsystems to include both reading and punching of paper tape on the PDP-7. With the TIP subsystems for reading and writing magnetic-tape directories and the CVFILE subsystem for code conversion, we have provided a flexible information-storage facility. It is possible to load massive amounts of paper tape on to the disk, convert the information to ASCII, and store it for archival purposes on magnetic tape. When one desires it, he can retrieve this information and format it for typesetting machines, and can reconvert and punch it in the special code configuration needed to run such varied off-line devices as the Photon or the IBM Selectric Composer.

FORMAT -- Walter I. Nissen, Jr.: -- We have developed a new subsystem, FORMAT, that permits high-speed bulk output under the control of a format table like that available from the TIP subsystem. This new subsystem permits easy testing of format tables during the formative stages of an applications system. Also available are features for truncation or trimming of fields of text before output, and tabular formats for columnar data.

Picture Retrieval -- Walter I. Nissen Jr.: -- We have made alterations to the TIP retrieval subsystem to allow for the storage and retrieval of pictures for display on the ARDS scope. Special format printers control erasure of the screen and delimit the text to be plotted. It is also possible to stop the output stream under format control. This is important on a scope display since the user should be able to indicate when the text has been read and, therefore, when this display may be erased and rewritten.

CALC/I -- David M. Jordan: -- Handling of numerical text within the TIP system in ways other than retrieval and formatting had long been an objective. At the beginning of the Summer of 1969, the CALC subsystem was nearing completion. Standard mathematical functions available in this subsystem will allow users to add, subtract, multiply or divide the contents of any field of ASCII text by the contents of any other fields or by constants. Execution of statements in a CALC specification may be conditional, permitting a computation to take place only if certain fields are present or permitting results to be reported only at some logical break in the data or at the end of a file. This subsystem will be useful in billing and inventory systems.

Minor Subsystems -- William D. Mathews: -- To round out the capabilities of the TIP system, we introduced a number of minor file-handling subsystems. ADDON allows the user to add constant fields on to every item in a file; RENUMB allows for the renumbering of fields in each item; and SEQUEN sequentially numbers the items in a file. One may accomplish some unusual results with such simple subsystems. SEQUEN, for example, may be used to add rank-order information to items in a particular field that have been sorted.

## Freshman Seminars

In alternate years, we have held freshman seminars on methods of handling scientific information. Students are given on-line experience and are encouraged to do independent study in organization and manipulation of information files.

## Summer Sessions

For the past two years, the Project TIP staff has given a two-week M.I.T. Summer Course in Information Technology. Professionals from industry, government and other universities

have discussed topics ranging from data collection and file organization to analysis of the user population.

TIP/III Planning Conferences

In June 1969, we held a series of conferences on an over-view of the planned TIP/III system on Multics. Project MAC participants and other interested personnel from the M.I.T. community discussed and criticized of the proposed system and suggested directions for this future work.

## 8. MATHEMATICAL ASSISTANCE PROGRAM (MAP2)

Work on MAP2 has been carried out in the School of Engineering, mainly with support from the National Science Foundation. One of the most difficult aspects of implementing an on-line user-oriented system is to ensure that the interface is sensitive to the likelihood that users will make errors. In particular, we have found that the programming necessary to ensure that all, or nearly all, possible erroneous usages of an operator are detected and the user informed (preferably before damage is done) is often more extensive than that required for the operation itself. The complications of such precautions can inhibit the ordinary user from himself making significant and useful additions to the system. Perhaps for this reason, almost none of the existing systems makes provisions for user-added operators. This problem, which we have been considering in some detail, is the subject of Terence M. Colligan's Bachelor's thesis (App. A). Colligan has designed and partially implemented a generalized and automatic procedure for checking for errors in the specification of operands for operators during the process of breaking down the input statements into an executable structure. The procedure utilizes simple bit-checking of "allowed" masks against coded specification of data-operands. The masks allow checking of the operands individually, as well as of more complicated effects involving logically permissible combinations of operands. It is intended that they can be specified to the system with simple declarations by "non-programming" users. In effect, the error-checking routines constitute the heart of an "add-an-operator" operator.

Prof. R. Kaplow

J. W. Brackett

T. Colligan

ADMINS (BUREAU OF SOCIAL SCIENCE INFORMATION)

I. deS. Pool

D. Griffel

S. McIntosh

## 9. ADMINS

Dr. Stuart D. McIntosh and Dr. David M. Griffel, who designed, implemented and used Mark III Admins to the limits of its capability, have been developing another system called Mark V Admins for primary and secondary data-handling. This work has been carried out in the Bureau of Social Science Information, of the M. I. T. Center for International Studies, under funding mainly from the National Science Foundation.

Admins Mark III has been described by competitors as a CTSS data-reduction and cross-tabulation system. Admins then (four years ago) relied, and now (Mark V development) relies quite heavily on CTSS facilities for computer programming.

There were several data-handling problems which we attacked four years ago:

1) Computer-usable data descriptions for each element of computer-usable coded data;

2) Computer-usable operations for linking several files together according to cross-reference relations within data sets and between data sets;

3) Interactive data analysis such that an analyst could path his way down a tree, look at the cell of a table, and make a decision to recombine his data;

4) Higher-level analysis facilitated by having computer-usable output that can be input to further processing as well as being used to generate reports.

The potential users of the system were primarily interested in questions of quantity (not identity) and in numbers of items with certain characteristics (not with numbers of identified items *per se*). This meant that it was out-of-place to put design emphasis on "find the identity of the item, then tell me all its characteristics", or "tell me the identities of all the items with a certain characteristic", or "count all the items with a certain identity".

Mark I Admins was short-lived. Most of the development work was done on Mark II. We dropped certain features (e.g., an automatic tree analyzer) and settled on a simplified version called Mark III. This has been in use for three years. The on-line documentation and the hard-copy documentation have been rehashed many times to keep pace with users' cultural change. The system has not changed in essence, but we have added some conventional statistics and some output features in order to relate it to other systems.

The aspect of data-handling for which Mark III Admins is most suited is the handling, for purposes of analysis, of many environmental variables with many contingencies among them. Mark III Admins is not particularly suited for progress control over a lesser number of operating variables or over the even lesser number of budgetary variables. This is mainly because of the

lack of emphasis on up-date and on item-identity quantification.

The paging and segmentation in the procedural and non-procedural Mark III subsystem are ad hoc. This is to some extent a CTSS constraint, but mainly it is because of problems of computer-usable data description.

The data world can be divided into several parts, among which are: data concerned with physical locations; data concerned with storage locations, i.e., data structures; program data, i.e., code and storage tables. We have not been concerned with these, nor with graphics data and textual data. Dictionary data (data description) and data data -- i.e., coded data -- have been the foci of our attention.

Mark IV was an ad hoc attempt to provide identity and quantity facilities and cross-reference features not available in Mark III. The Mark V design is in essence concerned with facilitating #all aspects of primary and secondary data-handling regarding all questions of identity and quantity of item and characteristic. In computer jargon, this means development of a programming language and operating system as an environment for an organization (simulation) language that can get to the data base via a data-description dictionary as the language of representation. The user language rests on a representation-language redescription facility and a report-generation facility.

We have settled on page and segmentation techniques -- especially on page tables that index identity, and on data structures suitable for the irregular matrices that are our data descriptions. This is not intended to prejudice further developments in categorization of data structures, but primary and secondary data-handling is our current problem.

# PART II

## ARTIFICIAL INTELLIGENCE AND INTELLIGENT AUTOMATA

Prof. M. L. Minsky

G. K. Adler
M. D. Beeler
W. T. Beyer
T. O. Binford
Prof. M. Blum
H. E. Brammer
T. F. Callahan
E. Charniak
P. E. deCoriolis
D. E. Eastlake
H. Fell
J. S. Freiberg
S. L. Geffner
J. P. Golden
R. W. Gosper
R. D. Greenblatt
A. K. Griffith
Prof. A. Guzman
W. H. Henneman
A. Herskovits
C. E. Hewitt
J. T. Holloway
P. Holloway
B. K. P. Horn
K. M. Jacobs
J. L. Jaroslav
T. L. Jones
E. I. Kampits
T. F. Knight
L. J. Krakauer
Prof. W. A. Martin
G. H. H. Mitchell
Prof. J. Moses
R. Noftsker
Prof. S. A. Papert
D. N. Perkins, Jr.
J. S. Roe
P. R. Samson
R. C. Schroeppel
J. M. Shah
S. W. Smoliar
M. Speciner
W. A. Spies
N. F. Stone
G. J. Sussman
C. T. Waldrop
D. L. Waltz
J. C. Wentzell
J. L. White
P. H. Winston
T. Winograd

Guests

Prof. C. K. Chow
T. G. Evans
Prof. E. Fredkin
Prof. H. N. Mahabala
Prof. M. S. Paterson
G. Voyat

BLANK PAGE

This report should be read in conjunction with last year's. This is particularly important to obtain a rounded view of our work on vision. In fact, much of what we say this year is logically prior to much that we said last year. Thus last year we discussed the abstract and theoretical problems related to the interpretation of pictures presented in a clean form (as line drawings or as subsets of an abstract retina) while this year we say more about how to obtain such pictures from the real world. The reason of course lies in the fact that the "higher level" work did not depend as heavily on prior solutions of hard-ware and systems problems.

We have not reported new work that lies directly in the line of development of ideas discussed last year. In particular, we have deepened and generalized some of the theorems on computational geometry. But the new state of knowledge is indistinguishable from the old on the level of discussion of this kind of report. Similarly we have not reported very new work which is still at a too primitive level of of developedment to be presented intelligibly. This includes work on natural language processing, concept information, teaching and a number of mathematical topics. The time constant for project of this sort is longer than a year. These matters will be dealt with in a further report due in the fall of 1970.

Analysis of Visual Scenes: The Concept of Vertical Problem-Solving

One can use vision in many ways to find out about objects in space. Let us begin with some of the simplest ideas, and then move on to the deeper problems. If the camera can move, one could use stereoscopic correlations of two pictures. Or one might use local operations to measure motion parallax. These do not solve all problems. Stereoscopy does not work well on featureless surfaces or curved boundaries; motion effects are misleading on plain or shiny objects. Nevertheless, both methods can help find the three-dimensional locations of parts of visual objects.

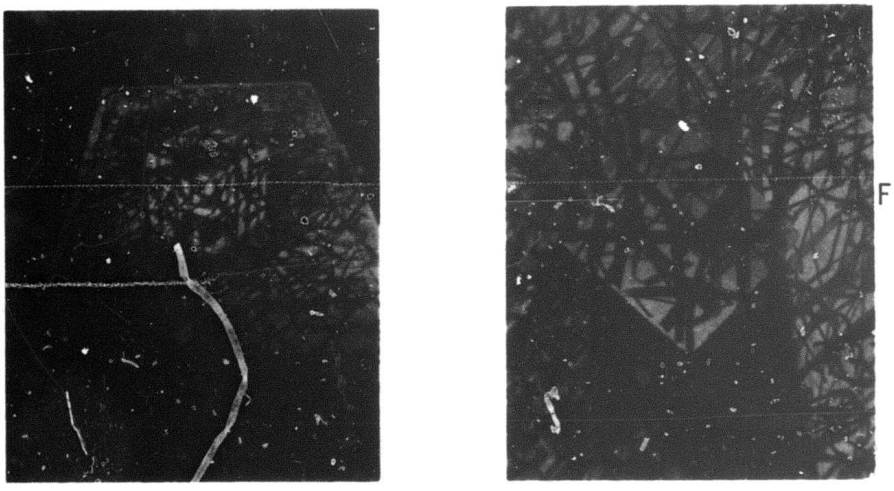


Fig.1

Focusing provides a third method of location. Its distinctive feature is that it needs only a single eye in a fixed location, provided that the eye has a wide enough optic aperture. For then the idea of measuring distance by stereoscopic parallax merges with that of measuring distance by finding the best focus setting. Figure 1 shows two views of an object lying on a plane; in the photograph it is hard to see the object because of camouflage.

(The machine sees from a little below the viewpoint of the left picture.) Berthold K.P. Horn has developed a program that can find the lens setting for best focus at each point in the visual field. The procedure, applied to a series of points along a horizontal scan through the middle of the cube, yields the profile shown in Fig. 2. (The location of the background is less definite than that of the cube because of the background's oblinquity to the camera.) Horn's program uses local Fourier transforms and compares the relative energy in the high and low spatial frequencies. It servo-controls the lens to maximize the highs, and it focuses at least as well as one can do manually.

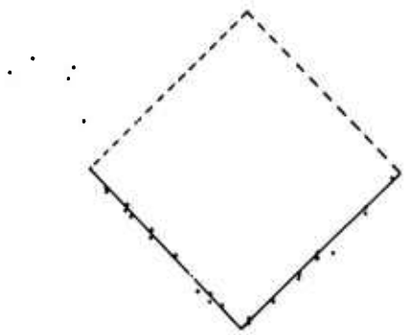


Fig.2

Horn will discuss the focusing procedure in detail in his thesis ("A Method for Finding the Shape of an Object from One View and Application to Face Recognition", in manuscript.) Two techniques are available: one uses a fast circular scan to obtain a periodic function characteristic of a large neighborhood; the other makes a faster scan of a smaller square. An operation manual (Horn, "Focusing", AI Memo 160) gives instructions for using the system with the PDP-6 computer and its optical-mechanical accessories, as well as some of the theory of the focusing procedure.

Now we have described three methods for optical range-finding. There are many other ways to attack just this one aspect of visual technology -- such as flying-spot scanning from a site off-center from the camera, holographic methods, and even optical radar. We must not, however, allow the myriad of technological possibilities to divert us from the deeper -- and incompletely understood -- problem of developing a visual system flexible enough to deal with real-world problems. The goal we have set ourselves is to find how to make a system that can approach the versatility of human vision. One outstanding feature of that system is its "passiveness" -- the great extent to which it can see without much special preparation or interaction with the objects in the scene. The secret lies in the intelligent viewer's ability to combine what he sees with what he knows about his world.

To pursue this, we have concentrated on the problem of reconstructing a three-dimensional structure using only a single monocular picture. People are quite good at understanding what is shown in a photograph; we should like to know how to make a machine do this. Now, with a very few exceptions, the many past attempts at computer analysis of scenes have been rather fruitless, and we should try to understand what went wrong. Almost all of those past attempts followed the same general plan: the picture is subjected to a sequence of transformations; each transformation is intended, in turn, to produce a successively more abstract representation until, finally, one obtains the

desired description of the scene. Typically, such a sequence might be:

1) Remove noise (by clipping, smoothing, etc.);
2) Enhance features (by boosting gradients, etc.);
3) Extract features (finding edges, vertices, etc.);
4) Group features into objects (by regions, parallelisms, etc.);
5) Identify objects (by partial matches, etc.).

Although there is a great deal of plausibility to this idea of progressing relentlessly from local to global, the concept of serial stages of pre-processing does not actually work well in practice. It is simply not suited to the real problem. Errors and assumptions made at each level are passed on to the next, and, even if each stage is quite clever at how it handles its data, the accumulation of mistakes over many stages leads to chaotic over-all results. The basic grammar of the problem is too context-dependent. The appearance of an object's features (and even their occurrence or non-occurrence) usually depends on global aspects of the arrangement and illumination of the scene. One must cope, for example, with

1) Direct line-of-sight occlusion of parts of objects,
2) Shadow occlusions that depend on the directions of lighting,
3) Highlights,

4) Reflections,

5) Textures,

6) Decorations,

7) Many other interactions between visual features and spatial forms.

Accordingly, the inevitable ambiguity problems met at each level -- "Is this an edge or not?" or "Are these two features part of the same object?" -- are often not solvable at that level. One can select a plausible interpretation only by using a wider variety of knowledge about the real world -- knowledge that ranges from principles of optics and geometry to knowledge about the particular environment and the objects likely to be in it.

In the traditional processing sequence outlined above -- we shall call it the horizontal vision system -- different kinds of knowledge are implicit at each level. The principles of optics are involved because each visual point represents a distribution function of space points in a way that depends on focus, scattering, reflection and noise. In the extraction of features, the processor must know whether the objects are likely to have straight edges, or texture boundaries, or polished surfaces. In analyzing a room, to give an extreme but real example, imagine the resulting chaos if the system did not know the significance of a picture frame!

At the level of grouping features and identifying spatial bodies,

we have already seen (Project MAC Progress Report V) how problems of projections and occlusion of three-dimensional objects lead us away from the simple template-matching schemes that work fairly well for two-dimensional problems. We found, however, that many of these difficulties could be handled by symbolic-description systems, and we shall assume that the reader is familiar with Prof. Adolfo Guzman's work, either through the survey in Progress Report V, or through his Doctoral dissertation. We now conclude that the lower-level aspects of vision, too, are best treated as problems in artificial intelligence to be handled by a mixture of general methods and special knowledge. Because the different kinds of knowledge interact at different levels, we must provide channels for such interactions so that hypotheses about high-level things like objects -- perhaps proposed by heuristics that use local evidence -- can be confirmed, rejected or revised by returning to other levels for other kinds of evidence. We use the term vertical system for this type of organization.

We have not yet enough experience with verticality to discuss it abstractly. But we now know a substantial amount about its application to visual problems; the body of this section reports what we have found so far.

Optical Anatomy of a Simple Scene

Fig.3

In Fig. 3, we see three cubes with dull white painted surfaces against a dark background, illuminated by concentrated light from a lamp above and to the right of the camera. The data and methods used here are from work by Arnold K. Griffith. Fig. 4 shows three plots of the light intensity, measured along three horizontal scans, each of 1000 points across the picture.

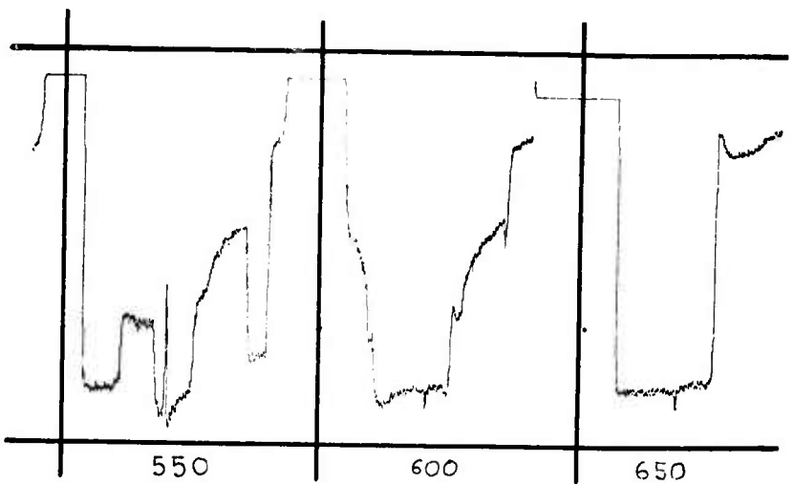



Fig.4

It is difficult to discern very much in these plots. The next illustration, Fig. 5, shows the result of applying, to a sequence of such cross sections, a kind of second-derivative operation averaged over enough adjacent sample points to give flat plots over regions that have reasonably uniform gradients. (The photograph and the measurements were taken from slightly different positions.)

We have marked a number of typical features:

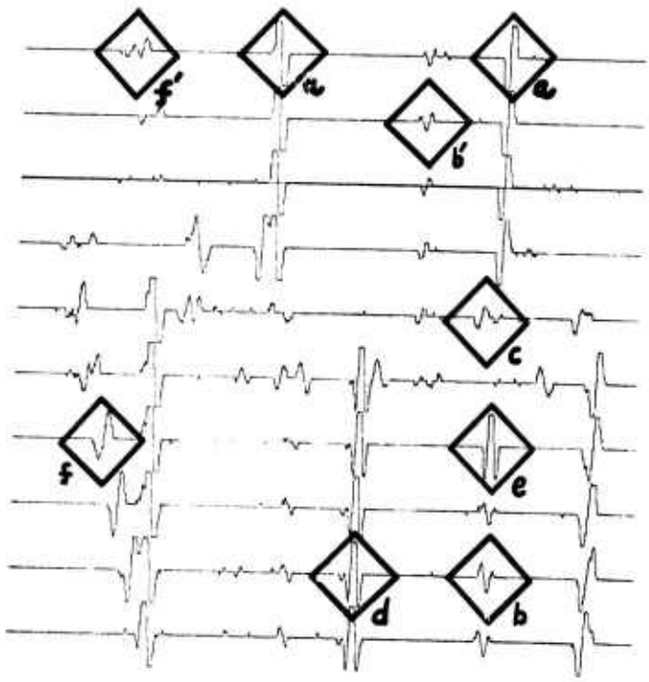



Fig.5

a is an outer edge. Against the dark background, it is a simple, sudden change in intensity, giving a typical "bipolar" pulse in the smoothed second derivative.

b is a more symmetrical feature; it is due to the "highlight" reflection on the front edge of the lower right cube.

b' is a superposition of the effects of a b and a small a.

c is the reflection of the bright surface of the upper cube in the top of the lower right cube. Although the paint there is dull, this surface is seen at a low angle, and this enhances reflections.

d is the crack between the lower cubes. The brightness measurements are all logarithmic, and truncated so that the plots won't overlap.

e is the spot of dirt on the upper front corner of the lower right cube. The edges and corners of objects often have highlights and often are dirty.

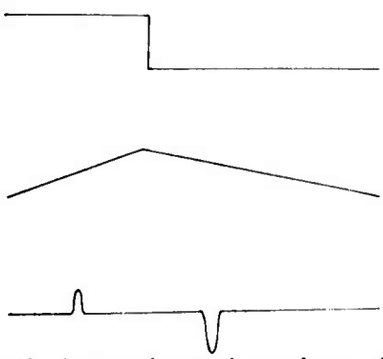
f is a shadow boundary visible on the dark background.

f' is another shadow boundary, somewhat less sharp because of penumbra and depth of focus.

The internal edges, such as at b, of uniformly colored objects often give small signals. Their width, in our plots, is an illusion due to the smoothing operation; on sharp corners, the highlight line is very narrow and easily missed by a coarse scan.

The history of attempts to write edge-finding and edge-following programs is long and inconclusive. Because the results were so obscure, Annette Herskovits (AI Memo 183) and Griffith made separate studies of the edges of geometrical objects; both concluded that the most common phenomena were superpositions of three effects (see Fig. 6):

Fig.6



(1) a simple step

(2) a slope change

(3) a highlight or a crack

They both investigated various detection filter methods for these. Griffith's thesis will include a theory of optical detection of edges under various assumptions about their intrinsic character and about the kinds of noise one might expect in a vision system. The most sensitive methods for detecting edges use two-dimensional operations, but these are very expensive with conventional hardware, because of the large amount of high-resolution information.

When a compromise must be made, it is not especially good simply to use a coarser homogeneous scan; for instead of the arrangement of Fig. 7, one can use that of Fig. 8, which has the same mean density but is better at catching thin edges.

Fig.7

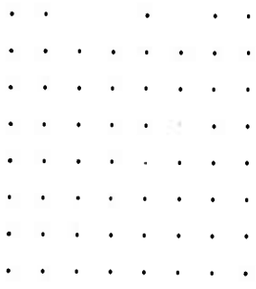
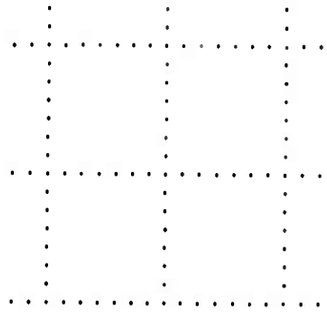
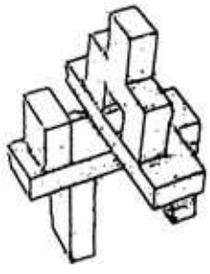
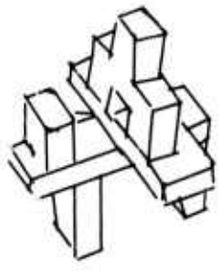


Fig.8



Now we apply this idea to some more realistic, cluttered scenes. We mark with short dashes the local maxima of the output of the edge-filter, along a mesh of fine horizontal and vertical scans. The problem remains to convert this set of local features into lines, and then into objects. Figs. 9 and 10 show the results of a process that uses a projection operation sensitive only to straight-line segments. Although it is not good for close-packed features, it is conservative and does not propose many false lines. Griffith's thesis will give details.





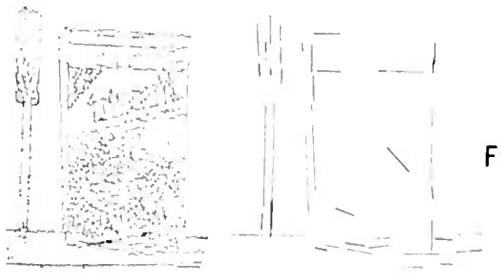
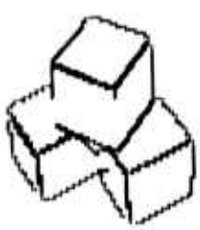


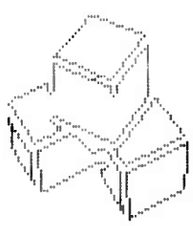
Fig.10

Another kind of process works in a complementary manner; it labels points whose neighborhoods are relatively homogeneous. Then the system finds the boundaries of the connected regions of such points. Thomas O. Binford (AI Memo 182) describes experiments on such a system. There are many problems in deciding how to reduce the region boundaries to useful line-descriptions. We see in Fig. 12 the result of such a system applied to the relatively simple scene shown in Fig. 11.

Fig.11

Figs.12

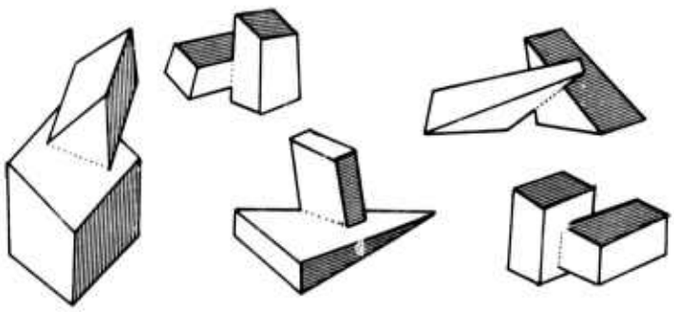




Still another approach to line-finding uses a two-dimensional local-gradient detector, followed by a scheme for assigning the locally maximal features to edges, as in the early work of L.G. Roberts. Richard D. Greenblatt is developing a system of this sort.

Problems exist at every stage of such processes. At each level, the selection of relevant features requires some a priori knowledge about the local world. Our verticality thesis holds that one cannot expect any one decision policy to work over a very wide range of situations, but that even a little feedback in this selection will help considerably. For example, each of the systems mentioned above will miss some edges of some objects, because of the problems of resolution, illumination, focus, contrast, texture or noise. If the system misses an interior edge, the SEE program (or rather, one version of it) may have to propose one object in place of two, as in Fig. 13, or two objects in place of one.

Fig.13



Assuming we are in a world of geometrical bodies, there is a variety of ways in which to use knowledge of the fact to propose corrections. (These will have to be verified, but proposing them is most of the battle.)

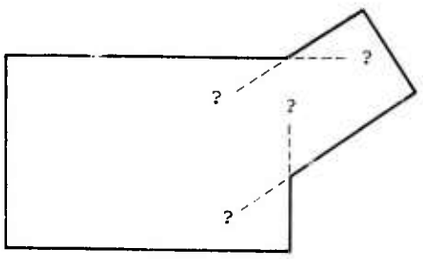



Fig.14

Missed edges, for example, are often related to concavity (see Fig. 14), and, in a rather Bayesian way, this suggests a search for missed lines radiating from the concave vertices into the figure's interior.

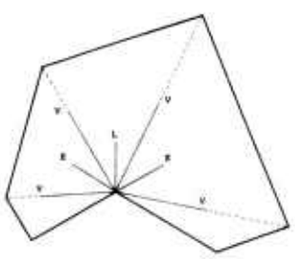
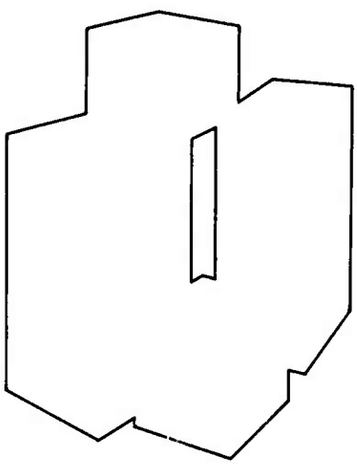



Fig.15

In Fig. 15, we indicate proposed lines of several kinds: V directly to other vertices; E interior extensions of the vertex's edges; and L an (absolute) vertical edge (very common in real interior scenes). One might further propose parallels and lines that make confocal triplets.

It is remarkable how much can be done with such a line proposer. In the case of structures made entirely of rectangular solids, Prof. Manuel Blum showed (Fig. 16) that a remarkable number of interior edges can be reconstructed just from the outer profile of the scene.



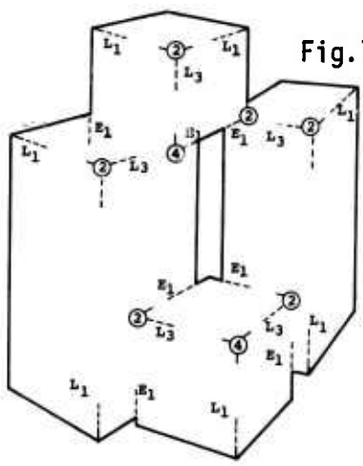


Fig.16

The numbers of lines proposed by such a scheme can be held to the order of tens, rather than of thousands. And the cost of verifying the existence of an edge with a specified location is enormously smaller than that of finding all such features independently, because -- for the same statistical confidence -- rejecting a particular null-hypothesis is always much easier than screening all of a large family of possibilities. The use of proposer-verifier system can thus reduce the total picture-processing effort by relaxing the tolerances on the

early, brute-force, feature-finding stages. In his forthcoming thesis, Griffith will give details of a complete system, already working, that does this. A first stage finds some of the edges in the scene. Then new lines are proposed on bases of parallelisms, region completion, etc., and verified by the detectors mentioned earlier.

By using a priori information, one can often get much more out of a picture than might seem to be in it. Assuming (correctly) that the sphere in Fig. 17(a) is uniformly colored and that the light comes from a compact source,

Fig.17a

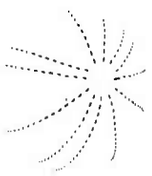


Fig.17b

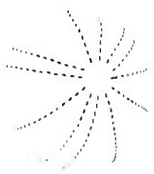
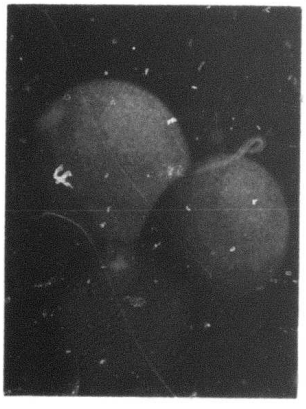


Fig.17c

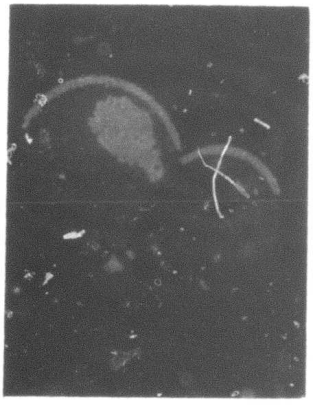
one of Horn's programs is able to reconstruct the surface by solving the appropriate differential equations. Then this program produces the stereoscopic pair of Fig. 17(b), (c). (Some readers will be able to fuse these by looking at a virtual point beyond the page.) The sharp shadow detail confirms, with the picture, the hypothesis of sharp illumination. Horn will present details of this program in his thesis. The method is quite complementary to stereoscopy and focusing need inhomogenous surface detail or discontinuities.

Even if the surface is not of uniform color and texture, there is still much more that can be done with a monocular picture. Lawrence J. Krakauer is developing a system that analyzes scenes such as a bowl of fruit. The process begins by locating and analyzing illumination maxima; it appears that, from their intensity-shape behavior one can, in many cases, distinguish dull from shiny surfaces. Local maxima connected by an illuminated band are likely to be on the same object, and Krakauer is testing some other heuristics for associating highlights with edges (Fig. 18).

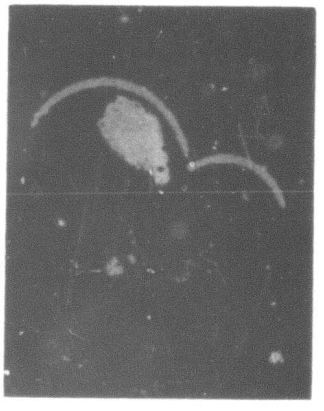
Digitized Image of Some Fruit



Edges and Maxima Enhanced



Features Alone



Krakauer's goal is to discover visual characteristics of surface properties, and to establish heuristics for grouping "non-geometric" features into natural objects, perhaps as Guzman did for geometrical objects.

The Vision System

We have discussed Guzman's SEE program (Progress Report V); this program assumes a scene description in terms of edges, vertices and regions, and produces a proposed assignment of these features to a set of three-dimensional bodies. Guzman's thesis describes in detail a more advanced version of that project. It includes additional heuristics for linking parts of objects, analysis of the system's behavior, discussion of and heuristics for correction of mistakes because of preo-processor errors, and some analysis of the system errors due to inherent ambiguities in ordinary scenes and in a variety of standard "optical illusion" scenes. Guzman's thesis also includes some observations about the problem of matching features between stereo pairs. In particular, the following simple technique, when it is embedded in a vertical system, will solve the majority of such problems. Consider two views of a geometrical scene (Fig. 19). In any stereo pair, one can dissect the two pictures into sets of matching line-pairs, defined by the planes through the two eye-points. Any physical object visible to both eyes will be sandwiched between the same highest and lowest such lines, as suggested by Fig. 20. For two different objects, it is unlikely this will be true by coincidence, especially

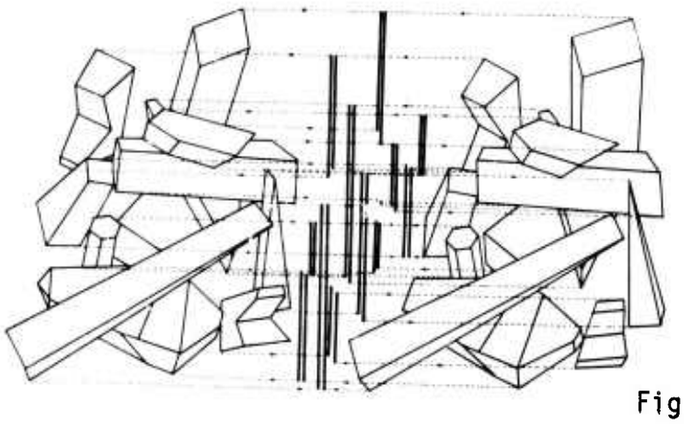


Fig.19

if the objects have more than one visible face, since the sandwich proposition is true also for each face! Thus, once enough point features are identified in the monocular pictures, one will have little difficulty in matching them between the pictures, and expensive cross-correlations should be unnecessary. This matching works well even when the two viewpoints are far apart.

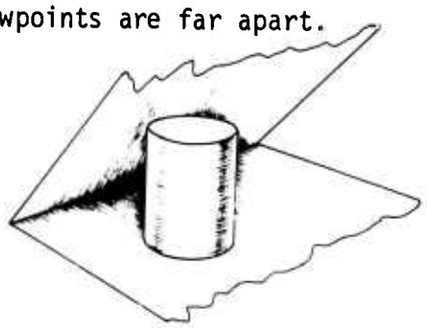
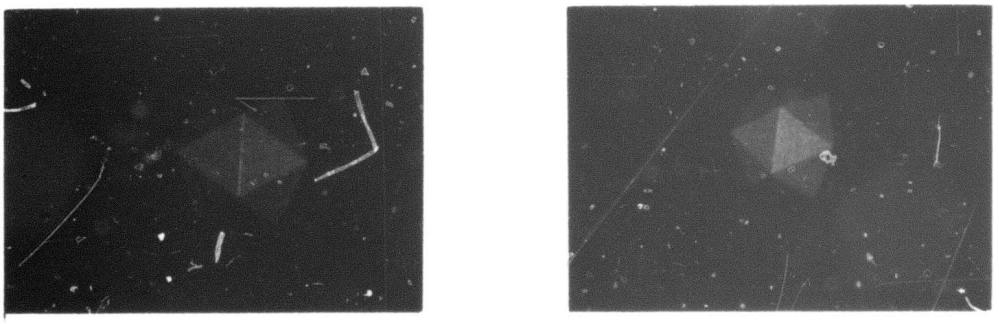


Fig.20

In another attack on stereoscopic vision, David N. Perkins, Jr. is developing a system that compares two views and produces depth information. His matching procedure is complicated by the requirement that it be tolerant of the many kinds of errors that pre-processors are likely to make. It proceeds by matching vertices and arcs topologically, with some geometric constraints, and with a back-up and search procedure for finding maximal matches of substructures when there are possible local ambiguities. Then, given the best topological match, the program proceeds to analyze the arcs that are thus proposed for matching to obtain space curves. For example, the stereo pair of Fig. 21 are converted, by Binford's TOPOLOGIST system, to the views of Figs. 22 a and b.

Fig.21



Then Perkin's system, on request, produces views as seen from the right side and from above in Figs. 22 c and 22 d (shown here without suppression of lines that might be hidden by surfaces).

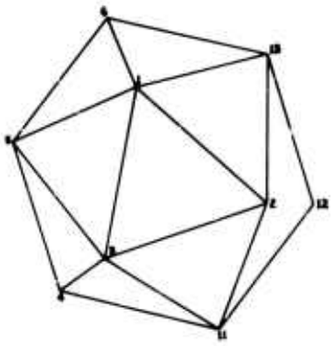


Fig.22a

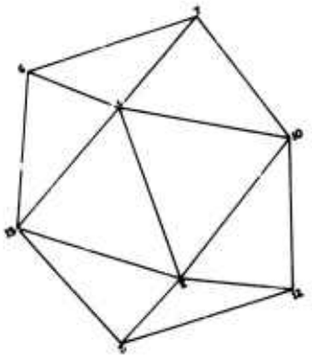


Fig.22b

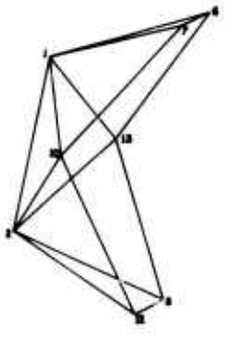


Fig.22c

Fig.22d

We now have a complete horizontal system of programs connecting Binford's topological pre-processor with Guzman's SEE program and on through to Patrick Winston's new system (described later in the report) that recognizes some particular types of objects -- e.g., wedges, pyramids, and rectangular blocks -- and learns to identify some multi-object structure such as rows, towers and bridges.

Professor Hosakere N. V. Mahabala developed the TOPOLOGIST-to-SEE interface. This program, SETUP ("Preprocessor for Programs Which Recognize Scenes", A.I. Memo 177) gobbles a list of line segments and produces a complete topological description of the graph formed by the lines. Because such features as lines and vertices found by pre-processors have some uncertainty in location, there are usually serious problems in deciding when two edges are really the same, or in connecting edges and localizing vertices. SETUP contains heuristics for plausible guesses about such matters. All lines are treated as enclosed within strips of a certain width, and all problems about closure, intersection, containment, colinearity, etc., are resolved by procedures that are based on a single predicate about the orientation of a point with respect to one of these half-line strips. Although this may not be any better in performance than other

segment-joining and line-grouping criteria, it is probably no worse, and Mahabala's system is distinctive in having greater logical clarity than any other system we have seen before. It is therefore much more likely to be improved by advances in theory!

Finding the edges themselves is still a problem. When one knows, a priori, that they are straight, the criteria Griffith and Herskovits developed are probably adequate for practical purposes, and these are further subject to substantial heuristic speed-up innovations. For the more general problem of describing an ordered set of points (such as one obtains as the boundary of a "homogeneous region") as a curve, we need a systematic way to apply various kinds of a priori knowledge. At present, we have a variety of such attempts, such as POLYSEG (Griffith, A.I. Memo 131), the Greenblatt-Holloway line-finder (A.I. Memo 101), and three others, by Binford, Greenblatt, and Jayant M. Shah. Unfortunately, none of these is well enough understood to be considered theoretically firm. A variety of other general-purpose curve-segmentation procedures has been described in the literature. But no one has really come to grips with the basic problem of incorporating the relevant a priori information, and we conclude that this is one of the aspects to be faced

in designing the vertical vision system.

R. Orban has developed a new program, ERASER, which detects and removes shadow boundaries from gemoetrical scenes. ERASER resembles SEE in that it uses the same classification of vertex features, but it also uses information about the relative brightness of regions. Its heuristics are based largely on the abundance of L, T and X types of vertices on the boundaries of shadow regions. ERASER is designed to criticize the output of SETUP, to remove shadow boundaries, and then to re-submit the result to SETUP before passing the problem on to SEE. It will not remove all shadows, but it is conservative about not removing real edges. Examples of ERASER performing well are given in Fig. 23.

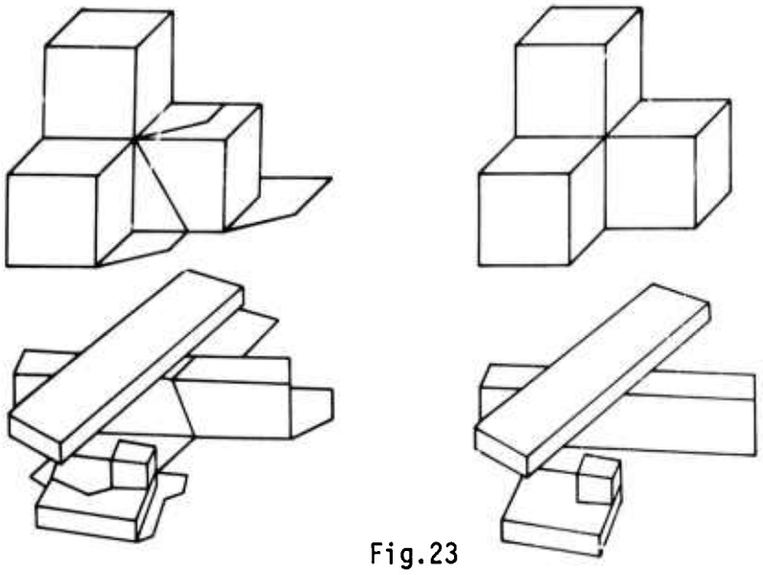


Fig.23

The ERASER system also contains some heuristics that Binford developed for guessing the direction of illumination; this is used to bias the operation of ERASER and is available to the rest of the system. In geometrical scenes, the system can measure the angles between real vertical edges and those shadow boundaries that appear to lie on horizontal surfaces in order to get a quantitative measure of illumination angle. As an application of verticality, the operation of both ERASER and SEE could be enhanced by verifying, at a higher level, that some of the shadow boundaries lie across otherwise uniform surfaces (say, by using Perkins's stereo system) or by verifying, at a lower level, that the proposed inner shadow boundaries are less than sharp, i.e., have penumbras.

Mechanical Structure Analysis of Visual Scenes

Blum and Griffith have written a program that can analyze the stability of the three-dimensional structure of rectangular blocks. The program uses this analysis in a planning scheme to propose the order in which the structure is to be built. Even in manipulating toy blocks, there are problems; one can ask which of these structures can be constructed with one hand.

Referring to Fig. 24, the one on the left cannot be built, the program asserts, because there is no stable three-block sub-part of the structure.

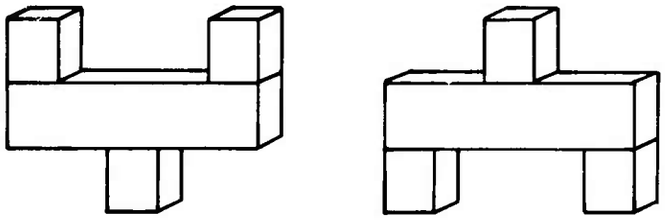


Fig.24

But, with another interpretation of the rules, one could first assemble the upper three blocks on the floor, then lift them into position. We expect our more advanced construction-planning programs to be able to use more advanced strategies in which sub-assemblies are so identified.

Winston is completing a program that learns to recognize types of structures from sequences of examples. We consider it to be a major advance in the areas generally known as concept-formation, or machine-learning. Given a scene (represented by a collection of regions, as produced at the output of the SEE program), Winston's program attempts to describe the scene in terms of elementary objects and already-known sub-structures and relations, using a descriptive language reminiscent of that Dr, Thomas G. Evans used (Ref. 2), but Winston's language is further developed.

For example, the scene of Fig. 25 leads to a description like that shown to its right. We tell the program this is a picture of an ARCH.

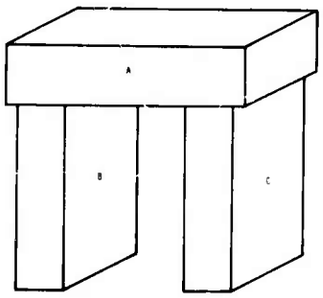
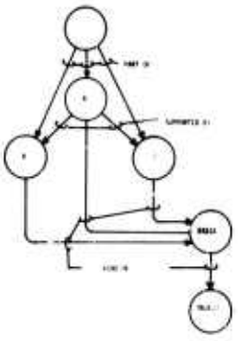


Fig.25



Next, we inform it that the picture of Fig. 26 is not-an-ARCH. The program then proceeds to compile a new description (of ARCH), as shown to the figure's right.

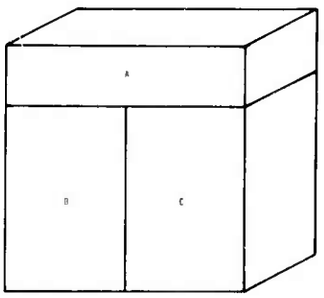
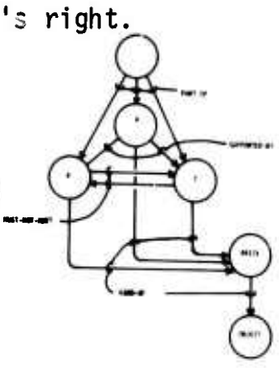
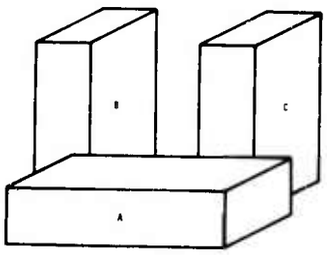


Fig.26



This new description of ARCH is obtained by comparing the descriptions of the two scenes, thus providing a second-level description. The must-not-abut relation is the outstanding difference it discovers, and it modifies the description of ARCH to require that the abutting relation not hold between the two supporting blocks. (Winston's program is equipped ab initio with heuristics for proposing support and non-supporting contact.)

Now, when we give the program the next figure (Fig. 27) as another example of not-an-ARCH, it again modifies the ARCH description, this time requiring (rather than just mentioning) the supported-by relations.



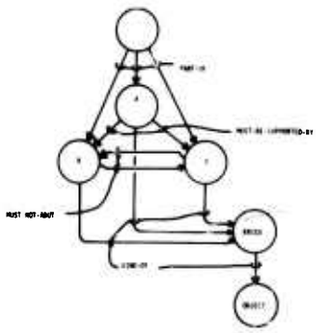
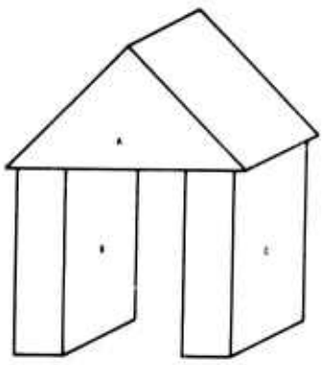



Fig.27

Again, this change occurs because the change in support is the most prominent difference the comparison program found. Finally, we show it one more example of an ARCH, and it re-compiles a description in which the requirement that the top-object be a rectangular brick has been removed (Fig. 28).



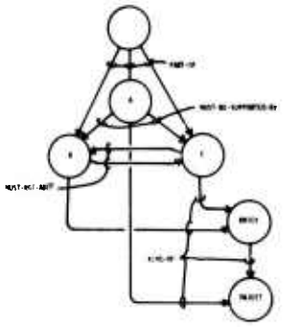



Fig.28

There are many arbitrary elements in how this program decides what to do at each step -- which differences to give highest priorities, how to match up different description networks, what explanations or excuses should be assigned to the differences it notices. These commitments are kept on back-up trees, so that it is possible for the program to recover from at least some disasters. The goal is to obtain behavior that would be plausible in, if not typical of, a child. The particular concept that the program develops from a certain sequence of examples will depend very much on those examples and on the order in which they are presented -- as well as on the connection of concepts the program has already acquired at that time. The experimenter will not always get the results he wants or expects! We cannot expect the first real concept-learning programs to be foolproof, any more than a teacher can expect his favorite instructional technique always to work; but here at last we have a chance to understand precisely how a training sequence interacts with built-in and previously acquired ingredients of the system. Winston's methods are related to earlier ideas like those in Newell and emphasize the use of explicit descriptions. While there is some similarity, in strategy, to the Feigenbaum-Simon EPAM scheme, the result is pointed toward a true structural network description of the concept. It should therefore lend itself better to direct analyses of examples,

instead of having to work through synthesis by generating and testing proposed examples.

More generally, having a description (rather than just a test) for a concept seems absolutely crucial. The advantages include:

1) The possibility of making deductions about the concept, including its consistency with a proposed detection scheme
2) Combining several descriptions in non-trivial ways
3) Comparing and contrasting descriptions, as in Evans's program
4) Using the description to generate, rather than merely to select, what next to do in a search program.

Similarly, in search processes, one could combine several descriptions to obtain summaries of the information acquired in exploring different alternatives. In most earlier heuristic search schemes, the procedures usually have to abandon almost all information acquired in the course of unsuccessful exploratory attempts because of inadequate descriptive facilities.

The A.I. Group is now committed to a braod attack on the problems of symbolic learning, through the application of heterological kinds of knowledge to the analysis of descriptions. An essay (A.I. Memo 185) gives a preliminary statement of our plans for this project.

Theorem-Proving

Gerald J. Sussman has implemented a theorem-proving program that uses the Resolution principle with an assortment of retrieval methods and other heuristics to make deductions in predicate calculus. Although there has been a number of interesting results, we nevertheless believe that the use of this technique, as an approach to artificial intelligence, is receiving much undue attention today, and we do not plan to give it a large place in our future activity unless some new and impressive demonstration of its power comes to light. Sussman has been experimenting with a variety of means for combining the deductive strength of predicate-calculus resolution with heuristic flexibility of less formally constrained problem-solving schema, but his conclusions are not encouraging. An example that puts one of the problems into a nutshell is this: suppose one is given

$A \Longrightarrow B$ and

$C \Longrightarrow D$, and

$(A \Longrightarrow B) \;\&\; (C \Longrightarrow D) \Longrightarrow E$,

and one wants to deduce the simple conclusion

E.

To be sure, the program manages eventually to obtain E, but only after many steps of converting the given statements into expanded "normal" forms that are in themselves rather meaningless.

This would not be so bad in itself, but it would become an acute problem if one were to try to give the theorem-prover additional advice about what to do in other situations, since the kinds of situations for which we can describe such advice are hard to represent in terms of the fractured internal expressions the Resolution system creates for its own use. In another experiment Sussman modified the representation of this problem so that the $\Rightarrow$ symbols were not recognized by the prover as meaning implies, and he added a separate set of axioms for using a more natural deduction method. Now the Resolution system produced a better proof in less time, even though it had to work indirectly through the new axioms! No doubt this particular problem could be ameliorated by some variant of the many combinatorial schemes that are being widely studied today, but we feel that, once such systems attempt to solve "real" problems, all such devices will fail as mere stop-gaps; they do not help one to come to grips with the construction of the kinds of cognitive models we think are needed to solve hard problems by using accumulated knowledge and experience. Sussman's system takes some steps in this direction by maintaining its statements in a structure partially ordered by the substitution-instance relation. Nor is his system restricted to first-order predicate calculus. But our gloomy expectations remain. Incidentally, we do not feel that completeness, or even consistency, is of very large importance. Logical completeness

is more or less inevitable in any system that knows a great deal, and consistency is not a notable feature in the intelligent machines that already exist. (The backers of the Resolution method achieve the wrong kind of completeness; the kind of completeness we need is for the problem-solver to be able to use any "natural" technique.)

At perhaps another extreme, Carl E. Hewitt is developing a language for constructing deductive systems with the utmost heuristic flexibility. His language, PLANNER, is designed to allow use of knowledge and types of representations of great variability. PLANNER must be one of the least procedural languages: to a large extent, one can specify what one wants done rather than how to do it. Consider, for example, a statement of the form "A implies B". As it stands, it is a simple declarative statement. But in PLANNER it can instead be interpreted as the imperative: "set up a procedure that will see if A is ever asserted, and if this happens assert B also." Or, it can be interpreted: "set up a procedure that will see if B is ever desired as a goal, and if so assert A as a new sub-goal to be deduced." This is only the skeleton of the idea: PLANNER contains machinery for easy manipulation of many different roles of declaratives, imperatives, goals and deductions. In attempting a particular deduction, for example, programs can specify suggestions for other theorems that should be used (and even in what order) to make the deduction. All

statements are expressed in a powerful new pattern-matching language, MATCHLESS, in which control of procedures is specified in terms of the forms of assertions, rather than in terms of particular assertions.

Because problem-solving often requires building up elaborate temporary structures, PLANNER has machinery for handling statements that were once true in a model which may no longer be true after actions have been performed, and for drawing conclusions that may have to be deduced from such a change. Control of such matters is specified in terms of local states, to which are bound information about changes in the data base -- erasures, assertions, new definitions, etc. -- since the data were last updated.

Hewitt describes his system qualitatively in a conference paper and in detail in A.I. Memo 168. He has now implemented the language and is programming a compiler to obtain sufficient speed to permit full-scale experiments. At present, he is using it to study deductions about the manipulations of objects by a robot, as in "pick up all pairs of cubes that are the same color, with one cube on top of a third cube that is in front of the other." Terry A. Winograd is completing a system that will translate such natural English statements into PLANNER assertions and theorems.

Natural Language Systems

Winograd is completing a system for handling problems of computer understanding of natural language. The system is designed to accept information in normal English sentences, to answer questions, and to execute commands, using semantic information context to understand pronoun references and to disambiguate gramatically complicated texts. To do this, Winograd uses a new linguistic scheme, based partly on the systemic grammar described in Halliday (1967) and in Winograd (1968) and partly on a special representation for both the grammar and the semantics. In previous attacks on such problems, some workers have used heuristic tricks -- key words or matching of phrase fragments -- or they have used non-heuristic formal grammars to do a detailed analysis of the sentence to which the semantics are to be applied. Winograd's system is based on a heuristic grammar that uses contextual information in analyzing the sentence, carrying out the semantic analysis concurrently; this is made possible by representing the grammar as a program instead of as a set of static rules.

Definitions of words, as well as the system's knowledge about things, are also stored as programs and are available to a deductive part of the system. This allows much more flexibility than one gets naturally from networks or rule-lists. The program can make long, complex deductions in answering questions or in absorbing new information.

The grammatical part of the system is operating, with a quite comprehensive English grammar (for a description, see Winograd, "PROGRAMMAR, A Language for Writing Grammars", A.I. Memo 181). The semantic programs are still in preparation, to be combined with the deductive system, which will use Hewitt's PLANNER language. The entire system could be used for a general question-answering facility for any corpus of knowledge programmed into the deductive system. The first applications will probably be concerned with instructing a robot and with analysis of children's stories at the first-grade levels.

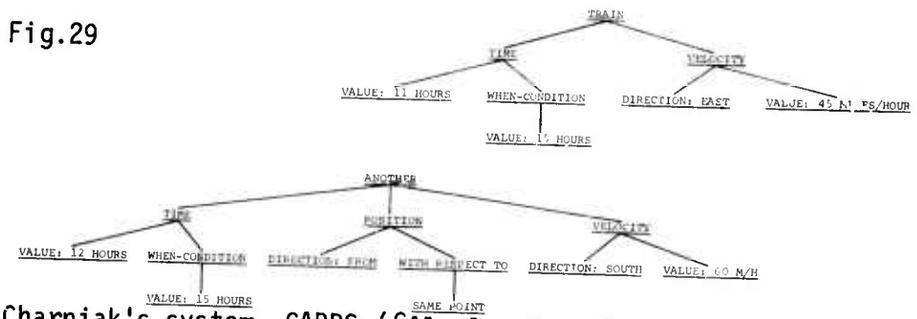
Loosely Stated Mathematical Problems

Several years ago, Daniel G. Bobrow completed a program that was able to solve some algebra problems stated in ordinary English. The mathematical material was rather sharply restricted to converting the sentences into simultaneous linear equations. Recently, Eugene Charniak has completed a new program that can solve some problems like this, stated in imprecise English:

> A train, starting at 11:00 a.m., travels east at 45 miles per hour while another, starting at noon from the same point, travels south at 60 miles per hour. How fast are they separating at 3:00 p.m.?

Solution of this problem requires a number of intellectual skills. Among these are, of course, the ability to manipulate the English text to derive a well-formulated symbolic problem, and knowledge of elementary calculus necessary to solve the symbolic problem. Perhaps less obvious, but equally important, is access to miscellaneous knowledge about the real world: for example, the knowledge that east and south are orthogonal directions, and enough knowledge about time to deduce that 3:00 p.m. is four hours later than 11:00 a.m.

Fig.29




Charniak's system, CARPS (CAlculus RAte Problem Solver) (TR-51, App. B), translates this problem into an internal representation of which a general impression can be gleaned from Fig. 29. This structure indicates two objects, TRAIN and ANOTHER. Associated with each is a velocity given as direction and magnitude, a TIME that has a starting value for each train and a WHEN-CONDITION. The latter refers to the fact that most calculus "rate problems" ask questions roughly of the form, "What is A when B?"; in our case, we obtain, "What is the speed at which the trains are separating when the time is 3:00 p.m.?"

CARPS uses this information structure to formulate an algebraic manipulation problem. HOW FAST and SEPARATING indicate that what is desired is the rate of change of the distance between the two objects. The distance is the hypotenuse of a right triangle with legs toward EAST and SOUTH. The lengths of the legs are obtained by multiplying the rate of travel by the time. The expression for the derivative is therefore

$$\frac{d}{dt} \sqrt{(45[t-11])^2 + (60\,[t-12])^2}$$

The desired value is at 3:00 p.m. (our WHEN-CONDITION). The derivative is found by algebraic manipulation, and the program then types

THE ANSWER IS 72.246 MILES/HOUR.

For its algebraic operations, CARPS uses parts of the MATHLAB programs described elsewhere in this report. CARPS has been used to solve other problems taken verbatim from calculus textbooks. These problems deal with cones, spheres and shadows as well as distances. In each case, the program was given sufficient knowledge to parse the sentences, set up an internal structure, and generate the equations.

We do not wish to imply that the program is very strong at solving calculus problems. Frequently, a problem cannot be solved because the program is unable to handle the syntax used, no method of solution is known to the program, or the problem requires facts about the real world which the program does not know or could not handle.

An example of the last difficulty arises in:

> A ladder 20 feet long leans against a house. Find the rate at which the top of the ladder is moving downward if its foot is 12 feet from the house and moving away at the rate of 2 feet per second.

Most adults have little difficulty in visualizing the situation described in this problem. CARPS is stuck because it does not know in which direction the foot of the ladder is moving. The phrase MOVING AWAY is interpreted by people to mean "moving away along the horizontal ground on which the foot of the ladder is presumed to rest." CARPS, however, does not realize that.

Clearly, CARPS's lack of such real-world knowledge cannot be circumvented or ignored. A crucial part of research towards problem-solving ability of this sort is concerned with the representation and use of such knowledge.

We have discussed (AI Memo 185) preliminary studies attempting to analyze the sorts of knowledge used by a first-grade child in understanding children's stories. Much of the calculus student's "general knowledge" is already structured at the elementary-school level, and we feel that progress in this area is essential to the future development of anything like "general intelligence". We have decided to make this area a very large part of our work in the next few years.

MATHLAB

MATHLAB is an interactive computer-program system that is being developed to facilitate creative work that involves extensive manipulation of symbolic algebraic expressions. MATHLAB is intended to be of help to research mathematicians and to appliers of mathematics engaged in "heavy manipulative work." In the MATHLAB project, therefore, serious attention has been paid to human engineering and to efficiency and speed of operation as well as to basic mathematical problems and to the problem of programming the computer to exercise initiative and some judgment in selecting and carrying out transformations of algebraic expressions.

MATHLAB is our best illustration of the idea of building "knowledge" into computer programs. The MATHLAB programs are actually quite capable in solving certain nontrivial mathematical problems. The essential idea, however, is to include provisions for interaction with that knowledge so that the user can build and administer complex but well-understood procedures. At best, the user and the programs supplement and reinforce one another and march rapidly through symbolic-manipulation problems that would take the unaided mathematician countless hours.

This last year, a new and much advanced MATHLAB was planned and partly implemented. Progress was made on faster parsing algorithms and on a representation of polynomials that speeds up addition and multiplication. At the same time, a significant advance in programmed symbolic integration was achieved by

implementing a decision procedure, due to Risch, for expressions involving rational functions, logarithms, and exponentials.

Also this last year, an extension of MATHLAB to handle special functions defined as integrals has enabled Moses to verify or correct some published tables of integrals. In 1958, W.D. Maurer of the Argonne National Laboratory compiled a table of 150 integrals involving the error function, erf(s). Maurer was unable to verify, by hand, the following integral, which he included in the compilation with a note to that effect. In fact, it was slightly incorrect as printed:

$$\int x^2 \mathrm{erf}(ax+b) e^{p\cdot x} dx = \frac{1}{p^3} \mathrm{erf}(ax+b) e^{px}(2 - 2px + p^2x^2)$$

$$+ \frac{pax - pb + p^2/2a - 2a}{a^2p^2\sqrt{\pi}} e^{-(ax+b)^2+px}$$

$$- \frac{p^4/a - 4p^3b + 4ab^2p^2 - 2ap^2 + 8a^2bp + 8a^3}{4a^3p^3} e^{-p/a(\frac{p}{4a} - b)} \mathrm{erf}(ax + b - \frac{p}{2a})$$

Moses' program was able to find the error.

Here is an example of factorization of a polynomial:

(1) x**6 - 1 = 0 (typed in)

(2) $x^6 - 1 = 0$ (MATHLAB's response)

In line 1 the user typed an equation. (The syntax is awkward because the typewriter is a one-dimensional device.) Line 2 shows the computer's response displayed in the conventional two-dimensional syntax of mathematics.

(3) 'PF('WS) (input)

(4) $(x-1)(x+1)(x^2+x+1)(x^2-x+1)$ (response)

In line 3 the user asked the system to factor the equation in line 2 by requesting the operation PF (Polynomial Factorization) to be applied to the WS (Work Space). The Work Space is always the last expression known to the system. In the response given in line 4, the quadratics are not fully factored because the program is restricted to finding the smallest factors which have integer coefficients.

Next we shall ask the system to integrate an expression, differentiate the result, and then check to see whether or not the result is identical with the original expression.

(5) 1/(x**3 + A*x**2 + x) (input)

(6) $\dfrac{1}{x^3 + Ax^2 + x}$ (response)

(7) 'integrate('WS, x) (input)

(8) IS THE EXPRESSION

$A^2 - 4$

to be considered positive, negative or zero (response)

(9) NEGATIVE (input)

The computer wants to avoid terms in the integral which have complex values, if possible. This is the reason for the question posed in line 8. Line 9 is the user's response, and line 10 is the integral. Obtaining the result involves use of a subsystem for handling partial fraction decompositions and one for integration of rational functions:

(10) $- 1/2 \log(x^2 + Ax + 1)$

$$+ \frac{-A}{\text{sqrt}(-A^2+4)} \arctan\frac{2x + A}{\text{sqrt}(-A^2+4)} + \log(x)$$

We shall now differentiate the integral.

(11) 'DERIV('WS, x) (input)

(12)
$$\frac{- 2A}{(\frac{(2x+A)^2}{-A^2+4} + 1)(-A^2+4)} + \frac{-1/2(2x+A)}{x^2+xA+1} + \frac{1}{x}$$

Well, the result is certainly not identical to our original expression (line 6). This is because the differentiation program differentiates a sum term-by-term without combining the results. (Note that the log x term in line 10 gave rise to the term in line 12.) To our rescue comes a simplification program RATSIMP (RATional SIMPlification) which will expand denominators and combine the entire result into a single fraction. This fraction is simplified by removing the greatest common divisor of the numerator and denominator.

(13) 'RATSIMP('WS,x) (input)

(14) $\frac{1}{x^3 + Ax^2 + x}$ (response)

Here is another integration problem:

(15) x**3/(1 - x**2)**(3/2) (input)

(16) $\frac{x^3}{(1-x^2)^{3/2}}$ (response)

(17) 'integrate('WS, x) (input)

(18) $\text{sqrt}(-x^2 + 1) + \frac{1}{\text{sqrt}(-x^2 + 1)}$ (response)

Note the system's preference for $-x^2 + 1$ over the more conventional $1 - x^2$. Mathematicians tend to obey the rule: If a sum of two terms contains a positive term and a negative term, write the positive term first. Graphic heuristics of this sort are now included in Martin's more sophisticated display routines.

(19) (2*x**6+5*x**4+x**3+4*x**2+1)/(x**2+1)**2*e**x**2 (input)

(20) $\frac{2x^6 + 5x^4 + x^3 + 4x^2 + 1}{(x^2 + 1)^2} e^{x^2}$ (response)

(21) 'integrate('WS, x) (input)

(22) $\frac{2x + 2x^3 + 1}{2x^2 + 2} e^{x^2}$ (response)

These results depend on a powerful pattern-matching program for algebraic expressions.

This example points clearly the need for a better way to enter mathematical expressions into the system. Methods which allow users to hand-write algebraic expressions, using a pen-like device, are now close to the step of useful application.

The following example shows the system solving a linear differential equation with constant coefficients, a problem-type of great interest to electrical engineers.

(23) DERIV(y,x,3) + A*DERIV(y,x) = sin(2*x) (input)

(24) $\frac{D^3y}{Dx^3} + A\frac{Dy}{Dx} = \sin(2x)$ (response)

(25) 'LDESOLVE('WS, y, x) (input)

(26) NEED INITIAL CONDITIONS (response)

The program allows one either to enter specific initial values for Y(0), Y'(0), and Y"(0) or to leave them as indeterminates (as is done below).

(27) ALLFORMAL (input)

(28) IS THE EXPRESSION

A

TO BE CONSIDERED POSITIVE, NEGATIVE OR ZERO

(response)

As before, the answer to this question will be used to generate an answer which contains no complex terms.

(29) POSITIVE (input)

(30) $$\frac{2Y(0) + 2Y''(0) + 1}{2A}$$

$$+ \frac{-Y''(0)A + 4Y''(0) + 2}{A^2 - 4A}\cos(\mathrm{sqrt}(A)x)$$

$$+ Y'(0)\frac{\sin(\mathrm{sqrt}(A)x)}{\mathrm{sqrt}(A)} + \frac{-1}{2A - 8}\cos(2x)$$

(response)

The solution is obtained with the help of a subsystem that takes Laplace transforms of both sides and obtains the inverse Laplace transform of their ratio, using the package for integrating

rational functions.

Hierarchical organizations such as MATHLAB's will become increasingly popular as programs and systems get more complex and sophisticated. While this system is ostensibly working at the higher levels of the hierarchy, it is actually spending most of its time at the lower supporting levels. Probably, over half the routines of its more than 60,000 words of memory are being utilized in solving a differential equation. This should serve as a warning to designers of time-sharing systems who would prefer that programs limit their use of memory.

The field of computer-aided instruction (CAI) has not progressed to the extent that many have wished. Teaching programs are unable to answer questions other than those which the designer had foreseen -- not only because these programs do not know enough English to understand the question, but basically because they do not at all understand the field about which they are supposed to teach. The designer of the usual kind of CAI course writes a script which is controlled by a context-independent interpreter. The script designer must provide for many possibilities at each step because he cannot rely on the interpreter of the script to know anything about the field with which the script deals. If a student is allowed, as he rarely is, a reasonable flexibility in replying to the program, then either the script designer is physically exhausted from considering all the

plausible replies, or he misses considering many such replies. We contend that a primary step in writing a sophisticated teaching program should be the education of the program in the area about which it must teach.

Consider, for example, the problem of writing a program for the teaching of freshman calculus techniques for differentiation and integration. Using the knowledge of these techniques that is imbedded in the MATHLAB system, it would be possible to write teaching programs that check the steps of a student's calculation, advising him of errors as he progressed. A preliminary program with these facilities has been prepared. Such a program could also be capable of performing differentiation or integration problems suggested by the students, explaining its steps as it proceeded. To achieve such capabilities through the writing of mindless scripts is unthinkable. To be sure, the experiments with the Calculus Rate Problem Solver (CARPS) have shown that a great deal more must be known before such programs can deal effectively with human beings in a wide area of knowledge.

The task of finding out what human users know is clearly related to the tasks of workers in the field of linguistics, psychology, and philosophy. A major failing, however, in the education given in those traditional disciplines is the lack of understanding of the concept of an algorithm or process. We believe that advances in our understanding of human knowledge as algorithmic processes are likely to be of supreme importance.

During the latter part of the proposal period, MATHLAB will interact strongly with the planned work in computer graphics. Although the most essential part of MATHLAB is conceptual and the next most essential is the programming of symbolic transformations, the interface with the user is very important, and we look forward to giving MATHLAB every advantage of effective man-computer-interaction techniques.

Chess

MACHAC-VI, a chess program developed by Richard D. Greenblatt for the PDP-6 computer, was begun in November 1966. It entered its first tournament the following January. Since then, it has participated in four more official U.S. Chess Federation tournaments and has achieved an official rating of 1528. In its last tournament, it had a performance rating of 1720 and drew an 1880 player. These statistics put it a little more than one standard deviation below the mean strength of all U.S. tournament players (which is about 1880). The machine has played perhaps 2000 games against chess players of every strength. It wins about 86 per cent of its games against tournament players. Greenblatt estimates that it represents a programming effort of about six man-months.

Once the basic program was completed, the first step in refining its play was to give it a model of the flow of power on the chess board. Next, it was given explicit

techniques for pins and discoveries; then it was given positional and pawn structure considerations. These and other features are largely completed now.

The next major step, only partially implemented now, will be to incorporate symbolic reasoning ability into the program. The opponent's threats are specifically identified, and each reply is evaluated for its ability to answer some or all of the threats. This does not mean that the opponent's threats were ignored before; but now the program proceeds with a more explicit sense of what it is doing, whereas in the classical minimax strategy the threats are implicit and therefore cannot be subject to symbolic statements of what to do. The explicit system should be much more efficient.

Another new feature, already implemented, adapts the plausible-move system to recognized states of the game: if one side appears to be ahead in the look-ahead analysis, it will prefer holding, trading and simplifying moves; the other side will require positive, attacking moves that might yield tangible gain. These and other features are operational, but the other programs have not been modified to take as much advantage of them as they probably could.

Another class of improvements is planned which will include real symbolic learning rather than mere tuning-up. We expect that these, with improvements in end-game strategy, will result in a substantial increase in strength of play.

## Eye-Tracking

The much-used term "man-machine interaction" usually refers to situations in which the communication relationship is exceedingly lopsided. Man can see, hear and touch the computer in many ways and places, but the machine is restricted to receiving information through a narrow bottleneck -- usually a Teletype. We have been interested for a long time in redressing this imbalance. This year we were able to achieve an old goal of enabling the machine to look at a person. More precisely: the machine looks at the man's eyes to determine his point of fixation.

Recording eye movements is a well-established technique in the study of perception, tracking skills, and even problem-solving behavior. But traditional methods give the pattern of eye movement only after analysis. We believe our system, which incorporates an eye-tracking device developed by J. Merchant of Honeywell under NASA contract, is the first that enables a computer to use the information in real time. As an example of its uses, Samuel L. Geffner has two programs that display text for a subject (such as a small child) to read and take action related to the word currently fixated. One of the programs causes the word to be pronounced by the computer; the other causes the word to be replace, in the display, by its translation into another language. These programs are mere demonstrations -- but they illustrate rich applications to teaching and other interactive situations. We are pursuing such applications, partly with support from NASA.

The A.I. Time-Sharing System

The experimental work of the Artificial Intelligence Group requies a high level of computer service for a limited number of users. Our system is based on time-sharing a two-processor machine (PDP-6 and PDP-10) with a core memory of $2^{18}$ words of 36 bits. Normally, the programs of the active users remain in fast memory; this limits the number of users now, but a paging device to be completed early in 1970 should allow some expansion required by the maturation of certain projects, notably MATHLAB.

The special requirements of the project led to a time-sharing system with enough novel features to merit discussion. Donald E. Eastlake describes the system in detail ("ITS 1.5 Reference Manual", A.I. Memo 161a). Besides the usual kinds of input-output and system calls, there are a number of special calls to reduce overhead and to facilitate real-time control. These include programs for operating the mechanical hands and computer eyes and other special remote-control devices. All ordinary time-shared programs run in one of the processors, and critical real-time processes are assigned, by user calls, to the other. The real-time processor is normally assigned to a single user at a time.

Because all user programs ordinarily reside in core, it is possible to switch between programs with great rapidity. Quanta of user time are short enough to allow program response to single

typed characters without noticeable delays. When swapping is introduced, we expect it to affect only semi-dormant users.

A user may have many jobs running "simultaneously". Each user commands a tree of procedures; each can create and control subordinate procedures; all have equal access to external devices and files. The top procedure, loaded automatically when the user declares his existence, contains a version of the well-known DDT debugging system. A special character allows transfer to the top procedure from any level, so that the user is automatically in position to use DDT's interrogation, breakpoint, and other debugging features.

Input-output devices are referenced symbolically and data can be transferred on character, word or block bases; an entire video image can be acquired by a system call while the calling procedure continues without waiting. User programs need no buffers in their own core images. Line-printer requests, for example, are buffered until the device is free. User programs communicate by "software interrupts" that are treated the same as hardware interrupts; superior procedures can store and retrieve words in inferior procedures as though they were I/O devices; buffered communication is provided so that pairs of procedures can treat each other as input devices. The file system resembles the CTSS file system.

The schedule algorithm tries first to equalize time between users, and second to equalize times between the procedures of a single user tree. A special procedure always runs which performs various jobs and can check constant portions of the system against a copy in an attempt to detect some forms of hardware or system failure.

The secondary storage uses IBM 2311 discs and DEC microtape drives which are file-structured in the same way. Users can establish symbolic links between file directories so that files can be shared by many programs and many users; these links can be chained.

The system has several operating stations. These include four text-display devices, several Teletypes and external telephone lines, a main console with DEC 340 display and several slave monitors around the laboratory, and a special console with controls for operating the eye-hand system. The system interfaces with a radio transmitter for inexpensive long-distance remote experiments.

A multiplexed digital-analog system operates either in word or block mode, on call or automatically, as requested. Any number of users may simultaneously have analog access on different channels in different modes. These are some of the real-time facilities:

1) Iris, focus, stereo mirror, for high-resolution image dissector
2) Pan, tilt, zoom, focus, iris, for small, low-resolution image dissector
3) Extend, tilt, rotate, grasp, finger curl, for hand MA-3
4) Extend, tilt, rotate, grasp, for hand MA-2
5) Roll, yaw, horizontal, vertical, swing, for arm MA-2
6) Position for tactile-sensor device
7) X, Y, Z, rotate for arm MA-3.

For moving the arms, special system calls provide acceleration- and velocity-limiting, software limit stops and other performance limits. To prevent disaster, certain motion commands are uninterruptable for limited times, and an arm-moving call is illegal if the device is under control of another user. Motion calls are not automatically buffered like those to the line printer! Although one can read any input channel at any time, most mechanical devices can be operated at leisurely speeds, with the input multiplexer channels read automatically. The system normally does this at a minimum of 50 times per second.

On the input side are sensors for all mechanical degrees of freedom, including hands and arms and optical parameters. There is a test stand for calibrating the positional control of the arms, and there is a variety of portable remote-control boxes with switches and continuous-adjustment knobs. A special system call gives the user great flexibility in real-time control

of program parameters. It connects potentiometers (through the input multiplexer) to arbitrary program variables -- i.e., memory registers -- and these can be specified to be fixed or floating, or arbitrary bytes. There are options for assigning absolute or incremental meanings to knob positions; in the incremental mode, there is a programmed velocity-dependent gain and a side-to-side hysteresis so that it is easy to make fine and coarse adjustments with the same control, and it tends to remain centered in its motion range.

The system contains good facilities for graphic display, which are presently limited to one user. Growing needs dictate acquisition of multiple-display hardware.

LISP (MACLISP)

LISP is the high-level language the Artificial Intelligence Group uses. Our LISP version is different in many ways from other LISPs, and some of these differences represent progress. The details are of interest only to specialists, who may consult the memo by Jon L. White ("Time-Sharing LISP for the PDP-6", A.I. Memo 157). The system does not use an A-list for ordinary variable bindings; it uses a value property; a special stack is used to unbind after lambda conversions. There are many small improvements in human engineering such as default values of non-specified function arguments. The garbage collector manages

space for arrays. The READ program allows new character-handling facilities and macro definitions of certain kinds. There is a variety of powerful editing facilities and display packages, and some strong debugging systems. The machine-language feature, LAP, has been strengthened. Whitfield Diffie has restructured the compiler to make the most of its decisions by dispatching on tables so the users can introduce special forms by giving the compiler instructions through the table entries. It would be more efficient if the compiler were able to make a deeper and more systematic analysis of the uses of expressions being compiled, separating executions for value from those for effect on flow of control, for side effects, etc., and for recognition of unnecessary recursion and similar simplifications. This itself is a problem in artificial intelligence.

Motivated by MATHLAB and other projects, we are now making an effort to introduce facilities for very fast arithmetic into LISP. We expect that the resulting system will be almost as efficient at number-crunching computation as is any conventional algorithmic language.

Mechanical Hands and Arms

The project has developed several mechanical effector devices for computer-controlled manipulation. This field is still substantially unexplored, and our work can be considered only to clarify some of the problems, not to solve many. There are

problems in several different areas.

There is a need for much deeper analysis of what is needed in designing a hand-arm system for various kinds of jobs. In his thesis, David L. Waltz studies motions the human hand uses to perform several basic tool-handling and similar tasks, to see which of the degrees of freedom of the hand were most essential, and how they were sequenced. There is some information about this in the orthopedic literature, but it is not sufficiently structured to serve as a base for programs. One needs to describe actions in terms of interactions of position, force, velocity and sensory responses -- in short, one needs something like a programming language for it. We also examined some choreographic languages but, although they contain some good ideas about path-of-motion description, they do not have sensory-interaction predicates. Waltz developed some notation that seems helpful and Ernst's discussion is still relevant. Waltz's thesis shows how a few degrees of freedom can accomplish much, but one wishes there were much more known about this subject.

In most of our experimental work, we have employed a modified AMF Versatran industrial manipulator arm (which operates in cylindrical-polar coordinates). Although the hands we attached to this device have some extra motions, the large-scale motion of that type of arm has no such redundancy; thus there is basically only one way to reach each point in space. Calculating this is straightforward, solving a not-too-complicated

equation that has only one solution.

Because this is a very clumsy device -- such an arm is unable to reach around obstacles or even to support an unobstructed object from an arbitrary direction -- we developed a much more mobile arm with many extra degrees of freedom. The new arm behaves more like a tentacle than like a coordinate system. Hydraulically operated, it is relatively slim and has five articulations, each with two degrees of freedom, a shoulder, three elbows and a wrist. An articulated prototype shows clearly that this geometry is adequate for everything a human arm can do and more. In the actual hardware mechanism, each of the eight interior hinge joints has approximately 110 degrees of flexion. We conclude that this is inadequate; it must be more nearly 180. The lesson we learned is that in a minimal system a small angular restriction means only that the work space is somewhat reduced. But, in a linkage whose purpose is a great variety of ways to reach points in the work space, all restrictions interact in messy ways to break up the higher-dimensional mobility space into hard-to-understand fragments, so that two slightly different three-dimensional positions may have to be reached (if at all) by grossly different global arm configurations. For such a mechanism with 10 degrees of freedom for reaching points in only three dimensions, there is a mathematical problem of inverting the position equations. One can sometimes get by

with simple hill-climbing by successive approximations, but this is really unsatisfactory because it is hard to adjoin global information, and it does not give an analytic picture of the alternative solutions to the problem. Jacques-Yves Gresser, in his thesis discusses a method that gives a solution specific to our arm-mechanism, using a rather general method of dividing the position problem into a series of almost-independent factor sub-problems. What Gresser does is to divide the arm in half, and describe the mobility space of each half, using simplifying approximations for each side. When the half-solutions are put together, there needs to be an accuracy-increasing iteration anyway, so there is hardly any real loss through approximating.

Now, in our arm-mechanism, it happens that the mobility zones of the half-arms can be described in terms understandable to humans -- for this particular arm, each zone resembles a portion of a torus. To find the ways to get the whole arm to a certain space position, one describes the intersection of the two tori, one centered at the shoulder position and the other the goal position. Each intersection point yields an approximate solution to the problem (that is, a possible location of the center of the arm), and a higher-level program could be asked which solution is most desirable.

There is a larger problem here: how should one represent a

machine's body image? For the problem of a single, not-too-complicated arm, one can doubtless get by with cleverely coded, sparse, three-dimensional arrays, but one would like something more symbolic. And one wonders what happens in the nervous system; we have not seen anything that might be considered to be a serious theory. Consider that a normal human can place an object on a table, turn about and make a gross change in his position and posture, and then reach out and grasp within one or two inches of the object, all with his eyes closed! It seems unlikely that his cerebellum could perform the appropriate vector calculations to do this; it is not a mere matrix inversion (and, even if it were, one would still need a theory). We would presume that this complex motor activity is made up, somehow, of a large library of stereotypical programs, with some heuristic interpolation scheme that fits the required action to some collection of reasonably similar stored actions. But we have found nowhere any serious proposal about neurological mechanisms for this, and one can only hope that some plausible ideas will come out of robotics research itself. Unfortunately, at present this area is somewhat dormant.

Another conclusion from our experiments is that delicate manipulations must be controlled by application of controlled forces (rather than by direct position control), with attention to matching velocities with inertias. For measuring the forces on a mechanical hand, we have taken two approaches. First, it is

quite feasible to engineer a grasping surface with good pressure sensitivity at a great many points by wrapping a coaxial cable connected to a Time-Domain Reflectometer. (A TDR is a sort of radar system designed to measure reflected radio waves that are produced in a soft-sheathed tube, by any deformations of the wall. The instrument permits hundreds of thousands of points, using a single electrical connection to the hand. (Waltz describes this system in his thesis.) Second, we have developed a strain-guage telemetered wrist that provides force information in the necessary six degrees of freedom. This little instrument is able to tell where and in what direction the hand is being pushed, assuming that the force is being applied at a single point -- a condition that usually holds in a first contact with an object.

Solution of the appropriate moment equations for the set of six forces at the wrist (Fig. 30) yields a certain line in three-dimensional space, along which a force of a certain magnitude is applied. If the machine knows the geometry of its hand, it can presume that the force is applied where this line intersects the hand.

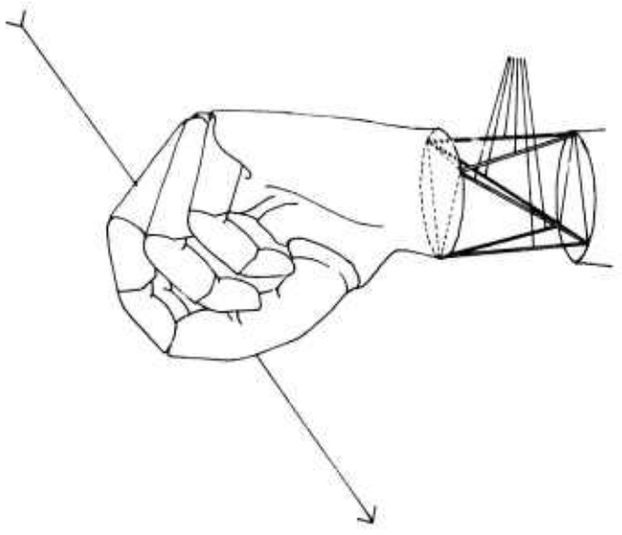


Fig.30

We have built this hand and are testing it. We shall describe it in a memo by Callahan, Shah and Minsky.

Computer Eyes

A computer must have eyes if it is to see. When the project started, no device was avaiable for interfacing a computer with a real-time camera, although there were film-reading devices on the market. We first used a Vidicon television camera, but we had difficulties with the limited signal-to-noise quality of the Vidicon. Besides, Vidicons are not suitably adaptable to random-access scanning, they have motion persistence and other problems. We decided that this added up to too many obstacles. The image-dissector camera offered much cleaner images (at the expense of sensitivity, which, for laboratory purposes, was unimportant) so, in collaboration with Information International, Inc., we designed and constructed our present camera system. An image dissector is essentially an inverted cathode-ray tube: an image projected on a photo-cathode produces electrons which are focused through a deflection system so that the current from a selected image point falls through a small hole and is then measured.

In such a system, the signal-to-noise ratio depends essentially on the number of electrons, and one can obtain more precision at the price of longer time-exposures. We decided to put this signal-to-noise ratio under direct program control and to make

it independent of the brightness of the point being measured. This is built into the camera's video processor. Horn gives the details ("The Image Dissector 'Eyes'", A.I. Memo 178). If the constant signal-to-noise constraint were enforced even on very dim points, the exposure times would become excessive, so the system has also a dark cut-off parameter that terminates the measurement very early if an initial estimate of the brightness falls below a specified value. This is also built in the video hardware. The resulting system can measure intensity over a 4096:1 dynamic range, with a brightness-discrimination sensitivity of one part in 64 throughout this range. The output appears as a floating point number or as a logarithm of intensity. The eye has also a reference-brightness input that can be used to make the measurements almost independent of over-all illumination fluctuations, and there are various protective devices to prevent damage to the image tube by excessively bright light sources. Horn's report discusses many advantages and pitfalls in using this system; anyone contemplating using image dissectors for computer vision should correspond with us about the results of some modifications now under construction.

It is frequently suggested that one ought to build various forms of parallel-processing artificial retinas, perhaps along the lines of biological systems. But the operations of the vertebrate retina and the subsequent image processing is not nearly so well

understood as is generally believed, and we do not think the time is quite ripe for taking such a step in hardware. In our present operations, the computation time consumed in the higher levels of scene-analysis is comparable to that spent in the lower-level pre-processing, so at present there would be no large time factor to be gained from fast parallel hardware -- even if we were able to decide how it should work.

## References


Lawrence G. Roberts, Machine Perception of Three-Dimensional Solids, Department of Electrical Engineering, M.I.T., Ph.D. Thesis, May 1963.

Thomas G. Evans, "A Program for the Solution of Geometric-Analogy Intelligence Test Questions", Semantic Information Processing, M.I.T. Press (Cambridge) 1968, pp. 271-353.

A. Newell, "Learning, Generality and Problem Solving", Information Processing 1962, IFIP Congress 1962, North Holland Publishing Co. (Amsterdam) 1963, pp. 407-412.

Edward A. Feigenbaum, "The Simulation of Verbal Learning Behavior", Computers and Thought, McGraw-Hill Book Co., (New York) 1963, pp. 297-309.

Carl E. Hewitt, "PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot", Proc. Int'l. Joint Conf. on Artificial Intelligence, ACM, May 1969.

M. A. K. Halliday, "Some Notes on 'Deep' Grammar", J. Linguistics, 3, 1967.

Terry A. Winograd, "Linguistics and the Computer Analysis of Tonal Harmony", J. Music Theory. 12, 1968.

Joel Moses, C. Engelman, and William A. Martin, "The Hierarchical Organization of an Algebraic Manipulation System", Proc. Conf. on Language and Society, Olivetti, Milan, 1968.

R. Anderson, "Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics", Ph.D. Thesis, Division of Engineering and Applied Physics, Applied Math, Harvard University, Cambridge, Mass., Jan. 1968.

Eugene Charniak, Proc. International Joint Conference on Artificial Intelligence, ACM, pp.303-316, May 1969.

Marvin Minsky, "Limitations of Language", Proc. Conf. on Language and Society, Olivetti, Milan, 1968.

H.A. Ernst, MH-1, A Computer-Operated Mechanical Hand, Department of Electrical Engineering, M.I.T., Ph.D. Thesis, December 1961.

Some Relevant

Internal Memos

of the

Artificial Intelligence Group

101 SIDES 21, Richard Greenblatt, Jack Holloway; described by Donald Sordillo, August 1966, MAC-M-320.

123 Computer Tracking of Eye Motions, Marvin Minsky, Seymour Papert, March 1967 (Vision).

131 POLYSEG, Arnold Griffith, April 1967 (Vision).

140 Perceptrons and Pattern Recognition, Marvin Minsky, Seymour Papert, Sept. 1967. Superseded by the book: Perceptrons, Minsky & Papert, M.I.T. Press, 1968.

145 A Fast-Parsing Scheme for Hand-Printed Mathematical Expressions, William Martin, Oct. 1967 MAC-M-360.

160 Focusing, Berthold Horn, May 1968.

161A ITS 1.5 Reference Manual, Donald Eastlake III, MAC-M-377. Revised July 1969 (ITS 1.4 Ref. Manual June 1968).

163 Holes, Patrick Winston, Aug. 1968. Revised April 1970.

164 Producing Memos using TJ6, TECO and the Type 37 Teletype, Larry Krakauer, Sept. 1968.

165 Description and Control of Manipulation by Computer-Controlled Arm, Jean-Yves Gresser, Sept. 1968.

166 Recognition of Topological Invariants by Modular Arrays, Terry Beyer, Sept. 1968.

167 Linear Separation and Learning, Marvin Minsky and Seymour Papert, Oct. 1968. This is a reprint of page proofs of Chapter 12 of Perceptrons, Minsky and Papert, M.I.T. Press, 1968. It replaces Memo 156.

168 PLANNER, Carl Hewitt, MAC-M-386. Oct. 1968; revised June 1969.

169 PEEK and LOCK, Donald Eastlake III, MAC-M-387, Nov. 1968. Replaced by revised Memo 161A, July 1969.

171 Decomposition of a Visual Scene into Three-Dimensional Bodies, Adolfo Guzman, Jan. 1961 MAC-M-391.

172 Robot Utility Functions, Stewart Nelson, Michael Levitt, Feb. 1969 replaced by Revised Memo 161A, July 1969.

173 A Heuristic Program that Constructs Decision Trees, March 1969, Patrick Winston.

174 The Greenblatt Chess Program, Richard Greenblatt, Donald Eastlake III, Stephen Crocker, April 1969.

175 On Optimum Recognition Error and Reject Tradeoff, C.K. Chow, April 1969.

176 Discovering Good Regions for Teitelman's Character Recognition Scheme, Patrick Winston, May 1969.

177 Preprocessor for Programs which Recognize Scenes, H. N. Mahabala, Aug. 1969.

178 The Image Dissector 'eyes', B.K.P. Horn, Aug. 1969.

180 The Integration of a Class of Special Functions with the Risch Algorithm, Joel Moses, MAC-M-421.

181 PROGRAMMAR: A Language for Writing Grammars, Terry Winograd, Nov. 1969.

184 Parsing Key Word Grammars, William Martin, MAC-M-395.

188 A Stability Test for Configurations of Blocks, Manuel Blum, Arnold Griffith, Bernard Neuman, Feb. 1970.

Books Published:

Perceptrons, Marvin Minsky, Seymour Papert, M.I.T. Press, 1968.

Semantic Information Processing, Marvin Minsky (Ed.), M.I.T. Press, 1968.

MAC Technical Reports:

MAC-TR-1 Bobrow, Daniel G., Natural Language Input for a Computer Problem Solving Language, June 1964. AD-604-730. (in Semantic Info. Proc.).

MAC-TR-36 (Thesis) Martin, William A. Symbolic Mathematical Laboratory, Jan. 1967, AD-657-283.

MAC-TR-37 (Thesis) Guzman-Arenas, Adolfo, Some Aspects of Pattern Recognition by Computer, Feb. 1967, AD-656-041.

MAC-TR-47 (Thesis) Moses, Joel, Symbolic Integration, Dec. 1967, AD-662-666.

MAC-TR-51 (Thesis) Charniak, Eugene, CARPS, A Program Which Solves Calculus Word Problems, July 1968, AD-673-670.

MAC-TR-59 (Thesis) Guzman-Arenas, Adolfo, Computer Recognition of Three-Dimensional Objects in a Visual Scene, Dec. 1968, AD-692-200.

MAC-TR-66 (Thesis) Beyer, Wendell Terry, Recognition of Topological Invariants by Iterative Arrays, Oct. 1969, AD-699-502.

Theses in progress:

| | |
|---|---|
| E. Charniak | Research on Natural Language |
| A. Griffith | Research on Edge-Finding |
| W. Henneman | Research on Complexity Theory |
| A. Herskovitz | Research on Edge-Finding |
| C. Hewitt | Research on Theorem Proving |
| N. Horn | Research on Curved Surfaces |
| L.J. Krakauer | Research on Curved Surfaces |
| R. Orban | Research on Shadows |
| D. Perkins | Research on Stereo Vision |
| D. Waltz | Research on Natural Language |
| T. Winograd | Research on Natural Language |
| P. Winston | Research on Concept-Formation |

BLANK PAGE

# APPENDIX A

## MAC-SUPPORTED M.I.T. THESES

Ascott, R. J., A Low Cost Capacitatively Coupled Tablet for Graphical Input, Department of Electrical Engineering, M.S., June 1969.

Berdell, J. R., Heuristic Approach to Routing and Sequencing of Multiple Component Jobs, Sloan School of Management, M.S., June 1969.

Betaque, N. E., Utility Theory Applied to Medical Diagnosis and Treatment, Department of Electrical Engineering, M.S., June 1969.

Beyer, W. T., Recognition of Topological Invariants by Iterative Arrays, Department of Mathematics, Ph.D., June 1969.

Clark, D. D., A Reductions Analysis System for Parsing PL/1, Department of Electrical Engineering, M.S., E.E., September 1968.

Colligan, T., Automatic Error Checking for On-Line Interpretive Systems, Department of Electrical Engineering, B.S., June 1969.

Eanes, R. S., Interaction Syntax Definition Facility, Department of Electrical Engineering, M.S., January 1969.

Graham, H. L., Recursive Graph Decomposition; An Approach to Computer Analysis of Networks, Department of Electrical Engineering, Ph.D., June 1969.

Greenbaum, H. J., A Simulation of Multiple Interactive Users to Drive a Time-Shared Computer System, Department of Electrical Engineering, M.S., September 1968.

Gresser, J. -Y., Description and Control of Manipulation by Computer-Controlled Arm, Department of Electrical Engineering, M.S., September 1968.

Guzman, A., Computer Recognition of Three-Dimensional Objects in a Visual Scene, Department of Electrical Engineering, Ph.D., December 1968.

Haggerty, J. P., Complexity Measures for Language Recognition by Canonic Systems, Department of Electrical Engineering, M.S., E.E., January 1969.

Hollander, C.R., Design of Multitasking Monitor for the IBM 1130, Department of Electrical Engineering, B.S., June 1969.

Ledgard, H. F., A Formal System for Defining the Syntax and Semantics of Computer Languages, Department of Electrical Engineering, Ph.D., February 1969.

Madnick, S. E., File System Design Strategy, Department of Electrical Engineering, M.S., Sloan School of Management, M.S., January 1969.

McMorran, P. D., On the Simulation of Nonlinear Dynamic Systems, Department of Mechanical Engineering, M.S., September 1968.

Morris, J. H., Jr., Lambda-Calculus Models of Programming Languages, Sloan School of Management, M.S., December 1968.

Reeve, C. L., A Text and Interactive Graph Software Interface for a Storage-Tube Display, Department of Electrical Engineering, M.S., September 1968.

Schroeder, M. D., Classroom Model of an Information and Computing System, Department of Electrical Engineering, M.S., E. E., February 1969.

Selwyn, L. L., Economies of Scale in Computer Use: Initial Tests and Implications for the Computer Utility, Sloan School of Management, Ph.D., June 1969.

Slater, J. W., Implementation of an Interactive Syntax Processor, Department of Electrical Engineering, M.S., B.S., June 1969.

Spirn, J. R., A Simulation Model of a Large Multi-Processing Operating System, Department of Electrical Engineering, B.S., June 1969.

Therrien, C. W., Tearing of Networks, Department of Electrical Engineering, Ph.D., June 1969.

Toong, H-M., The Ability of Lambda Calculus to Specify a Class of Simulation Language Algorithms, Department of Electrical Engineering, M.S., E.E., June 1969.

Waltz, D. L., A Versatile Electromechanical Hand, Department of Electrical Engineering, M.S., E.E., January 1968.

Wieselmann, P. A., Computer Implementation of the Polynomial Mapping Method of Plane Elasticity with Application to the Study of Crack Tip Stress Intensity Factors, Department of Mechanical Engineering, Ph.D., January 1969.

Zwick, M., New Computer Methods for Protein Crystallography, Department of Biology, Ph.D., September 1968.

# APPENDIX B

## PROJECT MAC TECHNICAL REPORTS

TR-1 BOBROW, Daniel G.
Natural Language Input for a Computer Problem Solving System
September 1964 AD-604-730

TR-2 RAPHAEL, Bertram
SIR: A Computer Program for Semantic Information Retrieval
June 1964 AD-608-499

TR-3 CORBATO, Fernando J.
System Requirements for Multiple Access, Time-Shared Computers
May 1964 AD-608-501

TR-4 ROSS, Douglas T. and Clarence G. Feldmann
Verbal and Graphical Language for the AED System: A Progress Report
May 6, 1964 AD-604-678

TR-6 BIGGS, John M. and Robert D. Logcher
STRESS: A Problem-Oriented Language for Structural Engineering
May 6, 1964 AD-604-679

TR-7 WEIZENBAUM, Joseph
OPL-1: An Open Ended Programming System Within CTSS
April 30, 1964 AD-604-680

TR-8 GREENBERGER, Martin
The OPS-1 Manual
May 1964 AD-604-681

TR-11 DENNIS, Jack B.
Program Structure in a Multi-Access Computer
May 1964 AD-608-500

TR-12 FANO, Robert M.
The MAC System: A Progress Report
October 9, 1964 AD-609-290

TR-13 GREENBERGER, Martin
A New Methodology for Computer Simulation
October 19, 1964 AD-609-288

TR-14 ROOS, Daniel
Use of CTSS in a Teaching Environment
November 1964 AD-661-807

TR-16 SALTZER, Jerome H.
CTSS Technical Notes
March 1965 AD-612-702

TR-17 SAMUEL, Arthur L.
Time-Sharing on a Multiconsole Computer
March 1965 AD-462-158

TR-18 SCHERR, Allan Lee (Thesis)
An Analysis of Time-Shared Computer Systems
June 1965 AD-470-715

TR-19 RUSSO, Francis John (Thesis)
A Heuristic Approach to Alternate Routing in a Job Shop
June 1965 AD-474-018

TR-20 WANTMAN, Mayer Elihu (Thesis)
CALCULAID: An On-Line System for Algebraic Computation and Analysis
September 15, 1965 AD-474-019

TR-21 DENNING, Peter James (Thesis)
Queueing Models for File Memory Operation
October 1965 AD-624-943

TR-22 GREENBERGER, Martin
The Priority Problem
November 1965 AD-625-728

TR-23 DENNIS, Jack B. and Earl C. Van Horn
Programming Semantics for Multiprogrammed Computations
December 1965 AD-627-537

TR-24 KAPLOW, Roy, Stephen Strong and John Brackett
MAP: A System for On-Line Mathematical Analysis
January 1966 AD-476-443

TR-25 STRATTON, William David (Thesis)
Investigation of an Analog Technique to Decrease Pen-Tracking Time in Computer Displays
March 7, 1966 AD-631-386

TR-26 CHEEK, Thomas Burrell (Thesis)
Design of a Low-Cost Character Generator for Remote Computer Displays
March 8, 1966 AD-631-269

TR-27 EDWARDS, Daniel James
OCAS - On-Line Cryptanalytic Aid System
May 1966 AD-633-678

TR-28 SMITH, Arthur Anshel (Thesis)
Input/Output in Time-Shared, Segmented, Multiprocessor Systems
June 1966 AD-637-215

TR-29 IVIE, Evan Leon (Thesis)
Search Procedures Based on Measures of Relatedness Between Documents
June 1966 AD-636-275

TR-30 SALTZER, Jerome Howard (Thesis)
Traffic Control in a Multiplexed Computer System
July 1966 AD-635-966

TR-31 SMITH, Donald L. (Thesis)
Models and Data Structures for Digital Logic Simulation
August 1966 AD-637-192

TR-32 TEITELMAN, Warren (Thesis)
PILOT: A Step Toward Man-Computer Symbiosis
September 1966 AD-638-446

TR-33 NORTON, Lewis M. (Thesis)
ADEPT - A Heuristic Program for Proving Theorems of Group Theory
October 1966 AD-645-660

TR-34 VAN HORN, Earl C. (Thesis)
Computer Design for Asynchronously Reproducible Multiprocessing
November 1966 AD-650-407

TR-35 FENICHEL, Robert R. (Thesis)
An On-Line System for Algebraic Manipulation
December 1966 AD-657-282

TR-36 MARTIN, William A. (Thesis)
Symbolic Mathematical Laboratory
January 1967 AD-657-283

TR-37 GUZMAN-ARENAS, Adolfo (Thesis)
Some Aspects of Pattern Recognition by Computer
February 1967 AD-656-041

TR-38 ROSENBERG, Ronald C., Daniel W. Kennedy and Roger A. Humphrey
A Low-Cost Output Terminal for Time-Shared Computers
March 1967 AD-662-027

TR-39 FORTE, Allen
Syntax-Based Analytic Reading of Musical Scores
April 1967 AD-661-806

TR-40 MILLER, James R.
On-Line Analysis for Social Scientists
May 1967 AD-668-009

TR-41 COONS, Steven A.
Surfaces for Computer-Aided Design of Space Forms
June 1967 AD-663-504

TR-42 LIU, Chung L., Gabriel D. Chang and Richard E. Marks
Design and Implementation of a Table-Driven Compiler System
July 1967 AD-668-960

TR-43 WILDE, Daniel U. (Thesis)
Program Analysis by Digital Computer
August 1967 AD-662-224

TR-44 GORRY, G. Anthony (Thesis)
A System for Computer-Aided Diagnosis
September 1967 AD-662-665

TR-45 LEAL-CANTU, Nestor (Thesis)
On the Simulation of Dynamic Systems with Lumped Parameters and Time Displays
October 1967 AD-663-502

TR-46 ALSOP, Joseph W. (Thesis)
A Canonic Translator
November 1967 AD-663-503

TR-47 MOSES, Joel (Thesis)
Symbolic Integration
December 1967 AD-662-666

TR-48 JONES, Malcolm M. (Thesis)
Incremental Simulation on a Time-Shared Computer
January 1968 AD-662-225

TR-49 LUCONI, Fred L. (Thesis)
Asynchronous Computational Structures
February 1968 AD-677-602

TR-50 DENNING, Peter J. (Thesis)
Resource Allocation in Multiprocess Computer Systems
May 1968 AD-675-554

TR-51 CHARNIAK, Eugene (Thesis)
CARPS, a Program which Solves Calculus Word Problems
July 1968 AD-673-670

TR-52 DEITEL, Harvey M. (Thesis)
Absentee Computations in a Multiple-Access Computer System
August 1968 AD-684-738

TR-53 SLUTZ, Donald R. (Thesis)
The Flow Graph Schemata Model of Parallel Computation
September 1968 AD-683-393

TR-54 GROCHOW, Jerrold M. (Thesis)
The Graphic Display as an Aid in the Monitoring of A Time-Shared Computer System
October 1968 AD-689-468

TR-55 RAPPAPORT, Robert L. (Thesis)
Implementing Multi-Process Primitives in a Multiplexed Computer System
November 1968 AD-689-469

TR-56 THORNHILL, D.E., R.H. Stotz, D.T. Ross and J.E. Ward (ESL-R-356)
An Integrated Hardware-Software System for Computer Graphics in Time-Sharing
December 1968 AD-685-202

TR-57 MORRIS, James H. (Thesis)
Lambda-Calculus Models of Programming Languages
December 1968 AD-683-394

TR-58 GREENBAUM, Howard J. (Thesis)
A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System
January 1969 AD-686-988

TR-59 GUZMAN, Adolfo (Thesis)
Computer Recognition of Three-Dimensional Objects in a Visual Scene
December 1968 AD-692-200

TR-60 LEDGARD, Henry F. (Thesis)
A Formal System for Defining the Syntax and Semantics of Computer Languages
April 1969 AD-689-305

TR-61 BAECKER, Ronald M. (Thesis)
Interactive Computer-Mediated Animation
June 1969 AD-690-887

TR-62 TILLMAN, Coyt C. (ESL-R-395)
EPS: An Interactive System for Solving Elliptic Boundary-Value Problems with Facilities for Data Manipulation and General-Purpose Computation
June 1969 AD-692-462

Project MAC Progress Report I
to July 1964 AD-465-088

Project MAC Progress Report II
July 1964-July 1965 AD-629-494

Project MAC Progress Report III
July 1965-July 1966 AD-648-346

Project MAC Progress Report IV
July 1966-July 1967 AD-681-342

Project MAC Progress Report V
July 1967-July 1968 AD-687-770

# APPENDIX C

## PROJECT MAC EXTERNAL PUBLICATIONS

Baecker, Ronald M., "Picture-Driven Animation", AFIPS Conference Proceedings 34, 1969.

Bhushan, Abhay K. and Robert H. Stotz, "Procedures and Standards for Inter-Computer Communications", Proceedings of the Spring Joint Computer Conference 32, 1968.

Corbato, Fernando J., "A Paging Experiment with the Multics System", Philip M. Morse Festschrift, 1968.

Corbato, Fernando J., "PL/1 as a Tool for System Programming", Proceedings of the PL/1 Seminar, U. S. Air Force Electronic Systems Command, 1968.

Corbato, Fernando J., "Sensitive Issues in Multi-Use Systems", Honeywell EDP Software Symposium Proceedings, 1968.

Corbato, Fernando J. and Jerome H. Saltzer, "Some Considerations of Supervisor Program Design for Multiplexed Computer Systems" (Invited Paper) IFIP Congress Proceedings, 1968.

Corbato, Fernando J., "PL/1 as a Tool for Systems Programming", Datamation 15, 5, May 1969.

Dennis, Jack B., "Program Generality, Parallelism and Computer Architecture", IFIP Congress Proceedings, 1968.

Dertouzos, Michael L., "Man-Machine Interaction in Circuit Design", Proceedings of the Hawaiian International Conference on Systems Sciences, January 1968.

Dertouzos, Michael L., "Computer-Aided Analysis for Integrated Circuits", International Conference on Circuit Theory, Miami, December 1968.

Dertouzos, Michael L., "On-Line Simulation of Block Diagram Systems", IEEE Transactions on Computers C-18, 4, April 1969.

Donovan, John J., Malcolm M. Jones, and Joseph W. Alsop, "A Graphical Facility for an International Simulation System", IFIP Congress Proceedings, 1968.

Donovan, John J., "Data Structures", Proceedings of the Hawaiian International Conference on Systems Sciences, January 1969.

Donovan, John J., "A Program for the Underprivileged and Overprivileged of the Boston Community", Proceedings of the Spring Joint Computer Conference, May 1969.

Evans, Arthur Jr., "PAL -- A Language Designed for Teaching Programming Linguistics", Proceedings of the ACM National Conference, 1968.

Fano, Robert M., "'Time-Sharing' uno Squardo al Futuro", Estratto dal Quaderno N. 110, Atti del Confegno sul tema: "L'automazione Electronica e le sue implicationi scientiriche, tecriche e sociali", Rome 1968.

Feldmann, Clare G., "Subsets and Modular Feature of Standard APT", Proceedings of the Fall Joint Computer Conference, December 1968.

Gorry, G. Anthony, and G. Octo Barnett, "Sequential Diagnosis by Computer", Journal of the American Medical Association 205, 12, September 1968.

Gorry, G. Anthony, "Strategies for Computer-Aided Diagnosis", Mathematical Biosciences 2, 1968.

Gorry, G. Anthony, "Modeling the Diagnostic Process", Sloan School of Management Working Paper No. 370-69, February 1969.

Guzman, Adolfo A., "Analysis of Scenes by Computer: Recognition and Identification of Objects", Proceedings of the Conference of Automatic Interpretation and Classification of Images, Italy, August-September 1968 (in press, Academic Press).

Guzman, Adolfo A., "Decomposition of a Visual Scene into Three-Dimensional Bodies", Proceedings of the Fall Joint Computer Conference 33, Part One, December 1968.

Guzman, Adolfo A., "Object Recognition: Discovering the Parallelopipeds in a Visual Scene", Proceedings of the Hawaiian International Conference on Systems Sciences, January 1969.

Hegna, Hovarth, "On the Use of a High-Level Language in the Programming of Multics", (in Norwegian), NCC Travel Report, The Norwegian Computing Center, Forskringsveien 1B, Oslo 3, Norway, December 1968.

Jones, Malcolm M., "Some Current Problems of Development of Large Scale Highly Modular Software", Proceedings of the National Symposium on Modular Programming, July 1968.

Jones, Malcolm M., "Multiple-Access Computer Systems", Engineering Management Conference, October 1968.

Jones, Malcolm M., Philip M. Walker, and Stuart L. Mathison. "Data Transmission and the Foreign Attachment Rule", Datamation, February 1969.

Jones, Malcolm M., "Multiple Access Computer Systems", Law and Computer Technology, February 1969.

Knuth, Donald E., The Art of Computer Programming 2/Seminumerical Algorithms, Addison-Wesley, Reading, Mass., 1969.

Kohavi, Zvi, and Igal Kohavi, "Variable-Length Distinguishing Sequences and Their Application to the Design of Fault-Detection Experiments", IEEE Transactions on Electronic Computers C-17, 8, August 1968.

Kohavi, Zvi, Switching and Finite Automata Theory, McGraw Hill, New York (in press).

Kopplin, J. O., "Stimulating Change in Engineering Education", IEEE Spectrum 6, 1, January 1969.

Licklider, J. C. R., "A Sociotechnical Crux in the Application of Computers to Education", 18th Conference on Science, Philosophy and Religion in Their Relation to the Democratic Way of Life, Education, Institutions and the Future, August 1968.

Licklider, J. C. R., "A Picture is Worth a Thousand Words -- and it Costs...", AFIPS Conference Proceedings 34, 1969.

Liu, Chung L., "A Note on Definite Stochastic Sequential Machines", Information and Control 4, 14, April 1969.

Luconi, Fred L., "Output Functional Computational Structures", Proceedings of the 9th Annual Symposium on Switching and Automata Theory, published by IEEE, New York.

Madnick, Stuart E., "Design Strategies of File Systems, A Working Model", Proceedings of the International Symposium on File Organization, Copenhagen, November 1968.

Madnick, Stuart E., "Multi-Processor Software Lockout", Proceedings of the ACM National Conference, August 1968.

Madnick, Stuart E., "Script, An On-Line Manuscript Processing System", IEEE Transactions EMS (special issue), August 1968.

Madnick, Stuart E., "Time-Sharing Systems: Virtual Machine Concept vs. Conventional Approach", Modern Data Systems 2, 2, March 1969.

Madnick, Stuart E., and Joseph W. Alsop, "A Modular Approach to File System Design", Proceedings of the Spring Joint Computer Conference, May 1969.

Mandl, Robert, "Canonic Systems and Recursive Sets", Proceedings of the Third Annual Conference on Information Sciences, April 1969.

Manove, Michael, M. Bloom, and Carl Engleman, "Rational Functions in MATHLAB", Proceedings of the IFIP Working Conference on Symbol Manipulation Languages, North Holland Publishing Company, 1968.

Martin, William A. and David N. Ness, "Optimizing Binary Trees Grown with a Sorting Algorithm", Sloan School of Management Working Paper 421-69, 1969.

McNaughton, Robert and Seymour A. Papert "Syntactic Monoids", Algebraic Theory of Automata (ed. M. Arbib), 1968.

Minsky, Marvin M. and Seymour A. Papert, Perceptrons: An Introduction to Computational Geometry, M.I.T. Press, Cambridge, Mass., 1969.

Ness, David N. and James C. Emery, "A Man-Machine Budgeting System", Wharton School, University of Pennsylvania, Working Paper No. 90, August 1968.

Ness, David N., "Interactive Budgeting Models: An Example", Sloan School of Management Working Paper No. 345-68, 1968.

Ross, Douglas T. and John E. Ward, "Investigations in Computer-Aided Design for Numerically Controlled Production", M.I.T. Electronic Systems Laboratory (ESL-FR-351), April 1969.

Scott-Morton, Michael S. and James Stephens, "The Impact of Interactive Visual Display Systems on the Management Planning Process," IFIP Congress Proceedings, 1968.

Selwyn, Lee L. and Daniel S. Diamond, "Considerations for Computer Utility Pricing Policies", Proceedings of the ACM National Conference, August 1968.

Sisson, Roger L., "Computer Simulation of a School System", Computer and Automation 18, 3, March 1969.

Ward, John E., "Computer Graphics", McGraw-Hill, Encyclopedia of Science and Technology, 1969.

Weizenbaum, Joseph, "A Backward Look Over CTSS", Computer Systems Symposium, Erlangen, Germany, 1968.

UNCLASSIFIED
Security Classification

## DOCUMENT CONTROL DATA - R&D

*(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)*

1. ORIGINATING ACTIVITY (Corporate author)
Massachusetts Institute of Technology
Project MAC

2a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2b. GROUP
None

3. REPORT TITLE
Project MAC
Progress Report VI July 1968 to July 1969

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
Annual progress

5. AUTHOR(S) (Last name, first name, initial)
Collection of reports from Project MAC participants
Prof. J. C. R. Licklider, Director

6. REPORT DATE
1 July 1969

7a. TOTAL NO. OF PAGES
168

7b. NO. OF REFS
—

8a. CONTRACT OR GRANT NO.
Nonr-4102(01), (02)

b. PROJECT NO.

c. NR 048-189

d.

9a. ORIGINATOR'S REPORT NUMBER(S)
MAC Progress Report V I

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
—

10. AVAILABILITY/LIMITATION NOTICES
This document has been approved for public release and sale; its distribution is unlimited.

11. SUPPLEMENTARY NOTES
None

12. SPONSORING MILITARY ACTIVITY
Advanced Research Projects Agency
3D-200 Pentagon
Washington, D.C. 20301

13. ABSTRACT

The broad goal of Project MAC is experimental investigation of new ways in which on-line use of computers can aid people in their individual work, whether research, engineering design, management, or education.

This is the sixth annual Progress Report summarizing the research carried out under the sponsorship of Project MAC. Details of this research may be found in the publications listed in the Appendices at the end of this report.

14. KEY WORDS
Computers Machine-aided cognition Computation structures Graphics Time-sharing Multiple-access computers Interactive management Information systems On-line computers Programming linguistics Artificial intelligence Real-time computers Theory of automata Intelligent automata

DD FORM 1 NOV 68 1473 (M.I.T.)

UNCLASSIFIED
Security Classification