PROJECT MAC PROGRESS REPORT XI

E. Fredkin

Massachusetts Institute of Technology

Prepared for:

Advanced Research Projects Agency
Office of Naval Research

December 1974

BIBLIOGRAPHIC DATA SHEET	1. Report No. Progress Report XI	2.	3. Recipient's Accession No. AD/A-004966
4. Title and Subtitle Project MAC Progress Report XI			5. Report Date December 1974 6.
7. Author(s) Project MAC Participants-Prof. E. Fredkin, Director			8. Performing Organization Rept. No. MAC-PR-11
9. Performing Organization Name and Address Massachusetts Institute of Technology Project MAC 545 Technology Square, Cambridge, Mass. 02138			10. Project/Task/Work Unit No. 11. Contract/Grant No. N00014-70-A-0362-0006
12. Sponsoring Organization Name and Address Advanced Research Projects Agency 3D-200 Pentagon Washington, D.C. 20301			13. Type of Report & Period Covered Progress Report 6/73-12/73 14.

15. Supplementary Notes

16. Abstracts

Final summary Report of Progress made at Project MAC under this contract during the period 6/73-12/73.

17. Key Words and Document Analysis. 17a. Descriptors

Real-Time Computers
On-Line Computers
Multi-Access Computers
Dynamic Modeling
Heterarchical Programming
Computer Systems
Artificial Intelligence
Computer Languages
Computer Networks
Information Systems
Programming Languages
Computation Structures
Automata Theory

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement This document has been approved for public release and sale; its distribution is unlimited.	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 135
	20. Security Class (This Page) UNCLASSIFIED	22.

Work reported herein was carried out within Project MAC, a Massachusetts Institute of Technology interdepartmental laboratory. Support was provided in part by the Advanced Research Projects Agency of the Department of Defense, under Naval Research Contract N00014-70-A-0362-0006.

PROJECT MAC PROGRESS REPORT XI

JUNE - DECEMBER 1973

PROJECT MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

CAMBRIDGE, MASSACHUSETTS 02139

TABLE OF CONTENTS

INTRODUCTION

This annual progress report to the Advanced Research Projects Agency of the Department of Defense describes research performed at Project MAC, funded by that agency, and monitored by the Office of Naval Research during the period 6/1 - 12/31 1973.

During this period, Project MAC consisted of approximately 275 people, including 27 faculty, 62 research and support staff members, 80 graduate students, 45 undergraduates, and 11 visiting researchers and scientists. Project MAC is organized into four divisions -- Automatic Programming, Programming Technology, Computer Systems Research and Fundamental Studies. The work presented herein was conducted in the first three divisions.

Since the development and subsequent transfer to Honeywell of the Multics time-shared system, the main thrust of research at Project MAC has been in the Automation of Programming. In particular, the Automatic Programming Division is concerned with the acquisition, structure and utilization of expert knowledge in specific domains. For example, in the Automatic Programming Group of this division, knowledge-based systems are used to automatically generate special-purpose programs from general descriptions of procedures in the context of inventory control. In the Mathlab group, the knowledge base is on Algebra, and the resultant system, called MACSYMA, is intended to be a mathematical assistant. In the Engineering Robotics group, real-time scheduling programs are automatically generated for the computer control of physical processes.

The Programming Technology Division is concerned with the computer facilitation of human programming. The Dynamic Modeling System, under development by this division, is an interactive programming system with a large repertoire of subprograms providing an integrated set of advanced tools, including information retrieval, graphics, and computer network capabilities.

The Computer Systems Research Division is studying methods of transforming information system construction into a more methodical engineering discipline. The primary laboratory for this research has been the Multics System, now in use as a main-line time-sharing facility by the M.I.T. community. In addition, work is being done on the development of certifiably correct systems, modeling of system performance, and the integration of computer networks with a computer utility.

ADMINISTRATION

Prof. E. Fredkin	Director
Prof. S. S. Patil	Assistant Director
H. S. Hughes	Special Assistant to the Director
D. C. Scanlon	Administrative Officer
G. B. Walker	Business Manager
A. D. Egendorf	Director of Information Services
C. P. Doyle	Administrative Assistant
B. H. Kohl	Librarian
A. A. Platt	Information Services

Undergraduate Students

R. Elkin

D. J. Morgan

Support Staff

G. W. Brown
L. S. Cavallaro
R. B. Combs
L. L. Gammel
J. I. Griffin
D. Kontrimus
M. K. Martucci
G. A. Morris
L. M. Muise

A. E. Normand
D. S. Niver
A. A. Pandya
E. M. Roderick
D. C. Rutherford
V. J. Sutherland
J. L. Winbush
C. Woodard

AUTOMATIC PROGRAMMING DIVISION

Academic Staff

Prof. M. L. Dertouzos
Prof. M. Hammer
Prof. A. Hax
Prof. R. Marsten
Prof. W. A. Martin
Prof. J. Moses
R. J. Fateman
V. S. Pless
G. R. Ruth
P. S.-H. Wang

DSR Staff

H. G. Baker
E. R. Banks
R. A. Bogen
G. P. Brown
J. P. Golden
L. P. Hawkinson
J. P. Jarvis
A. Nevins
G. F. Pfister
R. Schroeppel
A. Sunguroff
J. L. White

Graduate Students

R. V. Baron
V. A. Berzins
S. P. Geiger
R. T. Genesereth
J. J. Golovin
D. L. Grabel
D. L. Isaman
K. M. Kohn
R. B. Krumland
J. Kulp
M. Laventhal
A. Malhotra
N. Malvania
W. S. Mark
B. McFadden
L. Meador
J. A. Melber
J. A. Meldman
A. K. Mok
M. L. Morgenstern
G. A. Moulton
C. J. Terman
B. M. Trager
S. R. Umarji
T. Victor
S. A. Ward
D. Y. Yun

Undergraduate Students

H. I. Badian
D. J. Littleboy
S. M. Macrakis
C. H. Mink
D. A. Moon
B. Niamir
E. C. Rosen
G. L. Steele
A. P. Swide
G. Thomas
R. E. Zippel

Support Staff

J. S. Lague
G. B. Moore
N. J. Robinson
N. B. Siporin

Guests

A. Miola
H. Watanabe
U. Pape

AUTOMATIC PROGRAMMING DIVISION

A. INTRODUCTION

The Automatic Programming Division concerns itself with the replacement, rather than the augmentation, of programmers. With this goal in mind we feel it is essential that the automatic programming system have knowledge of the application area for which it is to write a program. The appropriate knowledge is that possessed by experts in the various areas, such as business, medicine, law, or applied mathematics. Since these experts are largely unfamiliar with computer science techniques we must form teams consisting of experts in a particular field together with computer scientists. Each of the groups in the Automatic Programming Division is such a team. Each field requires a somewhat different approach, but a sharing of views among the groups of the division helps to sharpen the process of creating expert computer systems.

The Automatic Programming Division is made up of four different groups, three of which are included in this report: 1) The Automatic Programming Group concerns itself with automatic program generation; 2) The Engineering Robotics Group is developing tools for the automatic production of software for the control of physical processes; 3) Mathlab has concerned itself with the development of a timesharing system -- MACSYMA -- containing powerful and efficient algorithms in many areas of algebraic manipulation.

AUTOMATIC PROGRAMMING GROUP

Academic Staff

Prof. M. Hammer
Prof. A. Hax
Prof. R. Marsten
Prof. W. A. Martin
G. R. Ruth

DSR Staff

H. G. Baker
E. R. Banks
L. P. Hawkinson
A. Nevins
G. F. Pfister
A. Sunguroff

Graduate Students

R. V. Baron
V. A. Berzins
J. J. Golovin
D. L. Isaman
K. M. Kohn
R. B. Krumland
A. Malhotra
W. S. Mark
B. McFadden
L. Meador
J. A. Meldman
M. L. Morgenstern
G. A. Moulton
S. R. Umarji
T. Victor

Undergraduate Students

H. I. Badian
D. J. Littleboy
C. H. Mink
D. A. Moon
B. Niamir
A. P. Swide
G. Thomas

Support Staff

J. S. Lague

G. B. Moore

AUTOMATIC PROGRAMMING GROUP

A. INTRODUCTION

At the time of the last progress report, the Automatic Programming Group had been in existance for a year and a half. That time had been spent in defining the general approach which the group would take to automatic programming, and starting a number of relatively small projects, both with the aim of involving new people in the work and exploring which would be the most fruitful areas for major efforts. The beginnings of some of these projects, and the motivation behind them, were described in the last progress report.

In the past year our work has matured to the point where we can now identify the major thrusts we intend to pursue for the next two or three years. We started out to build a single integrated system which we called Protosystem I. The thrusts we have identified will make the construction of that system possible. Indeed, we expect to integrate the results of the various thrusts together without having to do a complete re-design. Nevertheless, if each of the thrusts can produce major results in its own right, our task of research project organization becomes much easier. Below we describe our work in each of these areas. They are:

1. Understanding how a user might interact with a knowledge-based application system.

2. Attempting to set down the knowledge possessed by expert consultants.

3. Design of the OWL system for building expert problem systems.

4. Study of systems modeling and analysis. How are simpler models which retain the important characteristics of systems found?

5. Study of the process of algorithm generation by specialization of algorithms written at a higher level of abstraction.

6. Development of a system for translating from a very high-level language into IBM/370 PL/I in the area of business data processing.

B. UNDERSTANDING HOW A USER MIGHT INTERACT WITH A KNOWLEDGE-BASED APPLICATION SYSTEM

Ashok Malhotra became interested in how an English language question answering system could aid a user at a terminal in trying to ask questions about business data. He postulated the imaginary company situation with rising sales and decreasing profits described in section H. He then asked several businessmen to pretend that he was a computer which contained data on sales, costs, and other variables. He would also answer questions on what data was available and how it was computed. The results of these first experiments showed that two thirds of the questions were about what data was available or how it was catalogued, and only one third asked for the data itself or functions computed from it. The protocols showed the process by which the manager moved from his initial model of the situation (apparently dictated by his or her background) to a model commensurate with the way the data was gathered.

These results lead to the more elaborate experiments which Mr. Malhotra is now conducting. He has programmed a small question-answering system. A manager in another room types a question at a console in English. Mr. Malhotra intercepts the question and, rephrasing it if necessary, enters it into the system. The answer is then typed back to the manager. Everything is saved for later analysis. This expriment is providing a good sample of the English constructions which must be handled, the types of questions which must be answered, and the overall structure of a console session.

We feel that our understanding of English dialogue is reaching the point where we can define a level of capability fairly precisely. Since managers seem used to the idea of rephrasing a question so that someone less experienced can understand it, we plan to conduct further experiments where the experimenter will only answer questions which fall within some postulated level of capability of the sytem.

Analysis of protocols seems to be a very important tool for us. We have also participated in experiments where one physician pretends to be a patient and another diagnoses him or her. The second does not know what is wrong with the patient before the experiment starts and thinks out loud as he or she proceeds. We plan to conduct similar experiments in management.

There are several systems on the market which allow a manager at a console to define simple models and investigate their behavior. These are used, for example, to figure real estate transactions, calculate cash flow requirements, or aid in marketing media selection. Mr. Rand Krumland has been investigating these systems

and cataloguing their elements, with an eye to building a general purpose system which could be specialized to one of these areas by an English language dialogue.

C. ATTEMPTING TO SET DOWN THE KNOWLEDGE POSSESSED BY EXPERT CONSULTANTS

We have been working with Professor Arnaldo Hax in the M.I.T. Sloan School. After formal training in operations research, Professor Hax had a number of years of practical experience consulting in the design of information and decision systems for operations management. Even before he knew of our efforts, Professor Hax had decided to try to put his consulting experience into an integrated framework. He wanted to make possible the evaluation of alternative designs on a more systematic basis and to provide a framework for teaching others.

As Professor Hax had no previous knowledge of artificial intelligence work, we decided it was best not to involve him initially in our efforts to find a better representation for expert knowledge. Instead, we have been helping him to build a system consisting of a series of packages which are configured according to the answers a user gives to a multiple choice questionaire. The initial programming of the packages has been completed; they are being gradually elaborated and improved. The packages make possible the implementation of many common strategies for management of operations in a production and distribution system from aggregate capacity planning to detailed scheduling. The difference between these packages and commercially available business software is that the latter tends to offer only operations such as accounts receivable which do not involve decision rules affecting the total behavior of the system. Professor Hax is a proponent of a hierarchically integrated set of models to support decisions that operating managers must make, and he has built his system accordingly. These packages give us a well-defined target for the output of the design process as represented by the questionaire.

As might be expected, the questionaire has given Professor Hax some difficulty. He initially tried to write it by moving through the production process from raw materials to finished goods distribution. However, he realized that the questions to be asked for even the relatively straight-forward area of raw materials procurement depended on the general nature of the organiztion under study. This has led to a version which attempts to identify the firm as, for example, a one-factory fabricator of inexpensive items. Questions relevant to that setting are then asked. As Professor Hax envisions the design process, after an initial series of questions, the system will do a preliminary analysis of the firm in order to identify its problems, estimate the values of standard remedies, and confirm that the user's answers and data

fall into a consistent and believable pattern. It will then print out the general nature of a suggested design for user study. If the user desires to go further the system will ask further questions to acquire the detailed information necessary for a complete design.

In a seperate effort, Mr. Jeffrey A. Meldman, a member of the Massachusetts bar and an M.I.T. doctoral candidate, has been attempting to set down the knowledge required to assist a user in locating the legal doctrine relevant to a case at hand. He has taken the area of battery as an example and has been constructing in our new language, OWL, a model of battery, which the system would attempt to instantiate by using the facts of the user's case and legal findings, such as that a tomato can be used as a weapon. Mr. Meldman has been appointed an Assistant Professor in the Sloan School and plans to continue working with us there.

D. DESIGN OF THE OWL SYSTEM FOR BUILDING EXPERT PROBLEM-SOLVING SYSTEMS

Last year we reported on a language, MAPL, which we proposed to use for building models of the world. At the time, MAPL was incomplete. In seeking to solve additional representation problems in MAPL we came to rely more and more on the English language as a guide to how things could be represented. MAPL evolved so far that it seemed sensible to declare it a new language, OWL.

A major insight in OWL is that an event can be represented quite well as action, such as hit, and a series of properties of the action derived from case grammer. We were investigating the notion of cases last year, but now we believe that there is a universal set of cases which can be built into the language. An important step in translating from English into the case grammer representation is to recognize that the prepositions which flag the cases also have other uses. For example, in the sentence "I looked up the pipe", "up" can select a meaning of "look", in which case the sentence means the pipe was looked up in some reference, or "up" can signal the trajectory case, in which case "up the pipe" tells where I looked.

We have improved the parser which was described last year and we are converting it to work with OWL. We have partially implemented an interpreter for OWL and we are attempting to write a program in OWL capable of the dialogue in Figure 1 in order to debug the system. An example of an OWL procedure is shown in Figure 2. Owl is embedded in LISP. Every list is backpointed to every list which contains it as indicated in Figure 3.

We consider OWL to be the key to our project and we have all four or our DSR staff working on it. Lowell Hawkinson is leading the implementation effort. We expect the language will have one or two major revisions before it reaches a semi-stable form which can be published.

E. STUDY OF SYSTEM MODELING AND ANALYSIS

System modeling and analysis is a very difficult area. Success is not critical to the success of our entire project but it would be very important. We currently have three paradigms under which this problem is being attacked.

First, our system, which translates a very high level language into PL/I, must pick between alternate organizations of the computations and data on the basis of efficiency. Efficiency criteria require estimates of the number of data items and the number of computations which will arise in a system. We have attempted to discover these facts by simulation, but a straightforward simulation often leads to the simulation of rare events. Mr. V. Berzins has done some work to discover under what conditions probability distributions of the needed variables can be estimated symbolically. The calculations arising in business data processing are quite simple; nevertheless, symbolic calculations get out of hand unless simplifying assumptions are made. The correct assumptions to use are probably those now used by workers analyzing such systems. These must be discovered and codified.

Second, William Long is seeking a process by which a system with an elaborate model of the world could simplify that model in order to write a program for a specific purpose. For example, the system might have the notion that seats are intrinsically unique. If a ticket-selling program which sells non-reserved seats is to be written, the system might deduce that only a count of the number sold need be maintained, with no need to identify each seat.

Finally, William Mark is investigating whether a system with a notion of a concept like negative feedback could recognize the conditions for that concept and apply it in a problem description given by the user.

F. STUDY OF THE PROCESS OF ALGORITHM GENERATION

At the moment we do not have an active project in the area of algorithm generation by specialization of algorithms written at a higher level of abstraction. Considerable light has been shed on this subject over the past year through the

Figure 1

Hello, I am Susie. What is your name?

* Bill.
* I need a program written to manipulate blocks.

I can help you solve the problem of writing a program which builds a scene using the blocks from another scene.

* That is what I want.

OK. I need a description of the initial and final scenes.

* The final scene contains 3 Blocks: B1, B2, B3.

OK.

* What else do you want to know about the final scene?

What are the positions of B1, B2, and B3 with respect to each other?

* B1 is on top of B2 and B3 is on top of B1.

OK.

* Initially, all of them are on the ground.

Do you mean in the initial scene?

* Yes.
* Can you write a program from that?

I have enough information to start writing the program. I can't determine all of the information which I will need until I write the program.

* OK.
* Start writing it.

Do you want me to decide implementation issues such as what to grasp a block with?

* Yes, if I can ask you about them later.

You can. Is there a block on top of B2 in the initial scene?

* The initial scene only contains B1, B2, and B3.

I see. I have finished the program. Do you want me to describe the procedure?

* Yes.

First I move B2 with my left hand, then I put B1 on top of B2 with my left hand, then I put B3 on top of B1 with my left hand.

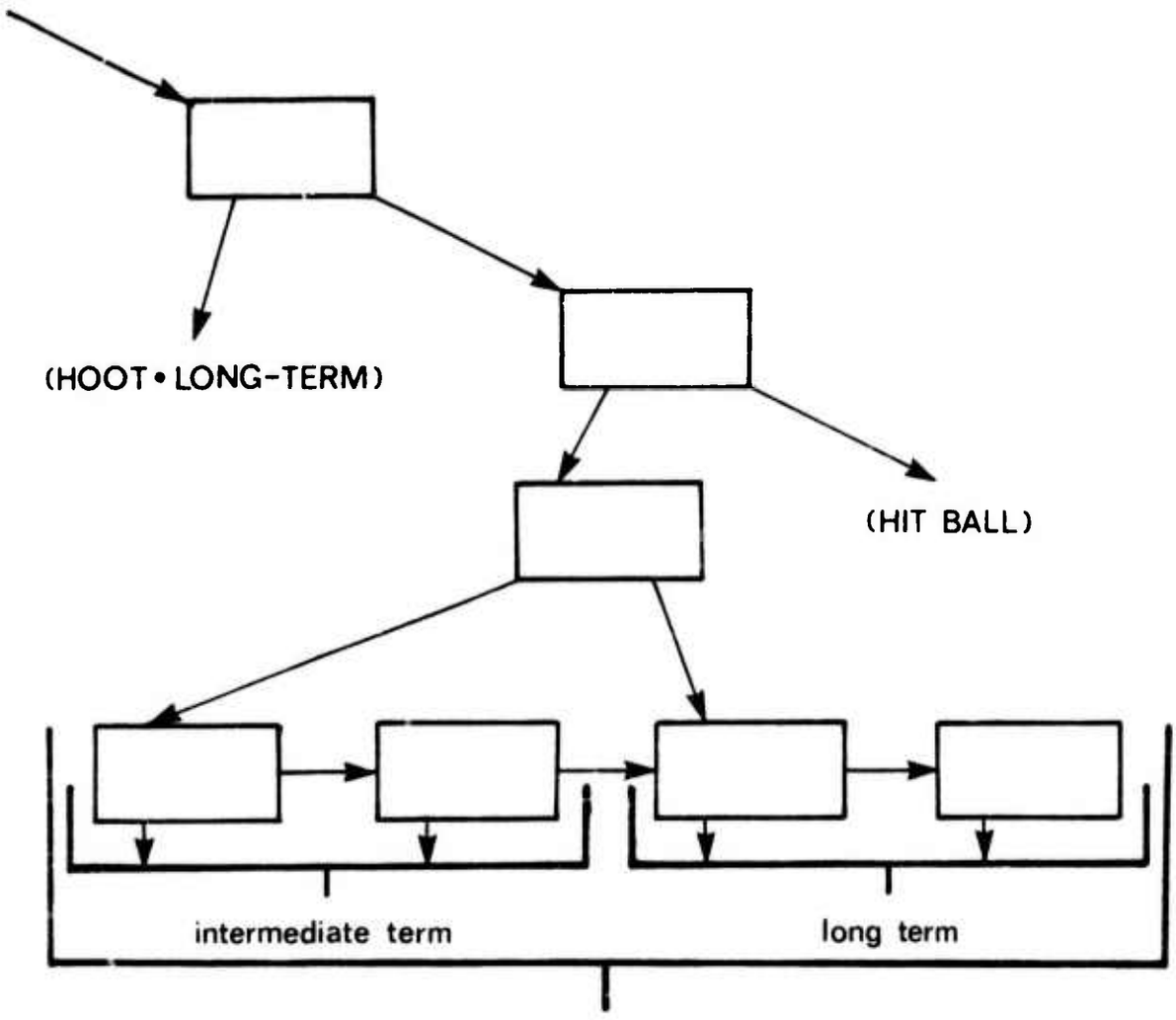
* How did you decide to use your left hand to move B2?

I needed to choose a hand with which to grasp B2. Both of my hands are free at that point in the problem. You told me to decide the implementation issues. I chose the first hand which I thought of.

Figure 1
A hypothetical dialogue with Susie Software.

```
(LEARN (DEFINE PROCEDURE (PUT-ON-TOP-OF BLOCK-1))
       (AGENT (PUT-ON-TOP-OF BLOCK-1) PERSON-1)
       (INSTRUMENT (PUT-ON-TOP-OF BLOCK-1) HAND-1)
       (PART AGENT HAND-1)
       (SPECIFIC-POSITION (PUT-ON-TOP-OF BLOCK-1)
                          (ON-TOP-OF BLOCK-2))
       (PRINCIPAL-RESULT (PUT-ON-TOP-OF BLOCK-1)
              (POSITION OBJECT SPECIFIC-POSITION))
       (METHOD (PUT-ON-TOP-OF BLOCK-1) (FIND SPACE-1))
       (POSITION SPACE-1 SPECIFIC-POSITION)
       (BENEFICIARY SPACE-1 OBJECT)
       (THEN (FIND SPACE-1) (GRASP OBJECT))
       (THEN (GRASP OBJECT)
              (MOVE (INSTRUMENT-1 (GRASP OBJECT))))
       (DESTINATION (MOVE INSTRUMENT-1) POSITION-1)
       (RESULT (MOVE INSTRUMENT-1)
              (POSITION OBJECT SPECIFIC-POSITION))
       (THEN (MOVE INSTRUMENT-1) (LET-GO-OF OBJECT))
       (Y-COORDINATE POSITION-1
              (PLUS 2
                 (Y-COORDINATE (POSITION (OBJECT (FIND SPACE-1))))
                    (MEASURE (HEIGHT OBJECT))))
       (X-COORDINATE POSITION-1
              (X-COORDINATE (POSITION (OBJECT (FIND SPACE-1))))))
```

Figure 2
Definition of PUT-ON-TOP-OF in OWL



Back pointers to all items containing this item as a top level element not in first position.

Figure 3
The long term memory element: (HIT BALL).

completion of the Ph.D. theses of Gregory R. Ruth in our group and Gerald Sussman and Ira Goldstein in the M.I.T. Artificial Intelligence Laboratory. All three of these persons are still at M.I.T. and this will help us in starting our project.

Ruth was able to find a series of productions, corresponding to design decisions, which could be used to generate various implementations of a sorting algorithm, such as a bubble sort. Given a sorting program written by a student in a beginning programming course in a simplified version of PL/I, Ruth's program would check it for errors. The basis of Ruth's scheme is to generate the same program as the student by referring to the student's program; discrepancies may suggest that general transformations such as algebraic simplification or rotation of the statements in a loop be applied to the generated program to bring it in line with the student program. Alternatively, discrepancies may suggest one of the common student errors known to Ruth's system. This allows Ruth's system to respond with comments like: "The loop test I<0 should have been I≤0, otherwise your program is ok."

G. DEVELOPMENT OF A SYSTEM FOR TRANSLATING FROM A VERY HIGH LEVEL LANGUAGE INTO IBM/370 PL/I

This large program, written in LISP, was the first project started by the group. In the first year the program reached the point where a simple example was put through the system and PL/I code generated. This was done, however, without an "optmizer", the routine which makes the design decisions. An interactive version of this routine was used. Over the past year, two designs for an optimizer have been proposed, one by Mathew Morgenstern and one by Steve Alter. The Morgenstern approach is to start at the inputs and move toward the outputs, generating all reasonable partial designs in the process. The alter approach is to make several passes over the whole set of computations. For example, all reasonable aggregations of data are found first, independent of what file organizations will eventually be used. Both approaches are being implemented.

The implementation of the entire system is now being headed by Dr. Gregory R. Ruth. He is writing a rather long memo describing the status of the system which should be ready by February 1974.

Although this project is rather complex, we feel it is a good one to have in our portfolio because it directly attacks a real world problem. It will be a good complement to the somewhat more theoretically oriented work of Professor Hammer.

H. A BUSINESS MODEL FOR AUTOMATIC PROGRAMMING - THE GLOBE UNION BATTERY COMPANY

Globe Union is an established manufacturer of lead batteries with head offices located in the mid-west. It has four plants where the actual manufacturing is carried out. These are spread out over the continental United States.

Globe Union manufactures fifteen variations of five basic battery types, for various purposes. Each distinct variety is identified by a unit number.

Globe Union sells mainly in bulk, to twenty major customers located all over the United States. Customers place long range "quotations" with Globe Union for specified quantities of a certain unit number. Globe Union supplies against these quotations on the receipt of orders from customer branches. Each branch is expected to order from a certain plant, usually the one closest to it. In general a given plant supplies customer branches in a set of states surrounding it.

Each plant manufactures all the types of units it supplies. The product is heavy, and transportation can make up a large proportion of product cost. Only in rare cases of shortages and lack of facilities to manufacture a specialized unit will batteries be supplied from other than the closest plant.

Plants manufacture according to certain inventory and production rules. They are expected to meet budgets on direct costs and overheads. Performance against budget as well as customer service are the main criteria for plant manager evaluation. Plants are not run as profit centers because prices on quotations are negotiated by the head office even though standard price lists exist.

It is early February 1974 and as President of Globe Union you are a little concerned at the results for 1973 which you have just received. Despite a 20% increase in sales over 1972, profits decreased by 1%.

You feel that the decrease in profit could be due to a combination of three causes: increase in overhead expenses, decrease in contibution margins (difference between selling price and direct cost) or a change in product mix toward less profitable units. You would like to investigate the cause of the decreased profit using the Globe Union Information System. Depending on what you find, you will make a decision to enforce strict control of the pricing or the quotations, review and reset list prices which are supposed to serve as guidelines for quotations prices, or introduce a cost control program. The purpose of this exercise is to determine which decisions are

appropriate under the circumstances.

As sales growth has bery healthy, you are inclined to disregard competitive actions in your analysis. You also assume that the cost and other data contained in the system is accurate.

The Globe Union Information System contains data on sales, costs, prices, and other indicators of Globe Union's operations during the last five years. It is capable of answering questions posed to it in simple English about the contents of the database and functions of these contents such as "profit" or "average price for unit 103". In addition, the system is capable of answering questions about itself, i.e., it can enumerate the data items it contains, explain the procedures embedded in the functions, etc.

The system can be queried much as one would use an assistant to answer questions, prepare reports, etc. It will provide appropriate responses to requests it does not understand or cannot accede to. A typical dialogue with the system may be:

Q: What data do you have regarding unit costs?
A: I have actual and budgeted costs for each unit at each plant.

Q: What was the cost of unit 103 in plant 8?
A: $78.23.

Q: What was the list price for unit 103?
A: $81.00

Q: Do you have a model for contribution margin?
A: Yes.

Q: How does it work?
A: it computes list (standard) price minus actual cost for the given unit.

Q: What was the contribufor unit 113 at Plant 2?
A: $9.20.

Q: What was the contribution for unit 81?
A: $9.30

Q: What was the avreage cost of unit 81?
A: Sorry, I don't know the word "avreage".

Q: What was the average cost for unit 81?
A: $78.67.

Q: What was the average budgeted cost for unit 81?
A: $76.00.

I. CONCLUSION

The directions of our group are now firmly established. We feel we have posed some good problems. We now turn to strengthening these directions and to solving the problems we have posed for ourselves.

Publications

1. Hax, A. C. and W. A. Martin, "Automatic Generation of Customized, Model Based Information Systems for Operation Management", Proceeding of the First Conference on Research in Organizations, Wharton, Pa., October 24-25 1973.

2. Malhotra, A., with C. L. Meador, "One-sided Cybernetics", Transactions of the IEEE International Seminar on Man, Systems and Cybernetics, Boston, Mass., November 7-9 1973.

ENGINEERING ROBOTICS

Academic Staff

Prof. M. L. Dertouzos

Graduate Students

S. P. Geiger
N. Malvania
A. Melber
A. K. Mok
G. F. Pfister
G. Sockut
C. J. Terman
S. A. Ward

Support Staff

N. Robinson

ENGINEERING ROBOTICS

A. INTRODUCTION

In the report submitted by this group (formerly called "Educational Computer Systems") for 1971-72 it was announced with the departure of Professor Weizenbaum for a two year period, the research objectives of the group would be oriented toward Engineering Robotics. This transition has proceded smoothly over the intervening two years, and this is expected to continue until the departure in July 1974 of Professor Dertouzos to serve as Acting Head of Project MAC.

The major long term research goal of the group continues to be the development of tools for the automatic production of software for the control of physical processes. Our approach has emphasized the utilization of descriptive information which is ultimately resolved (by the system) into control algorithms, rather than the explicit specification of the algorithms themselves. The designer of a control system, for example, specifies time constants associated with the process to be controlled rather than a detailed algorithm for the scheduling of the controlling program. Efforts in this area have, during the past year, been concentrated on scheduling algorithms and their relation to physical time constraints. Results include an optimal scheduling algorithm and the successful implementation of an initial language using this algorithm.

During the reporting period the group has also continued research initiated previously in the areas of (i) Systems for dynamic computer graphics; (ii) development of the DELPHI timesharing system; and (iii) theory of programming languages. Two doctoral theses have been initiated during this period, in the respective areas of dynamic graphics and programming language semantics. In addition one Master's and three bachelors theses were initiated in the area of Engineering Robotics.

B. ENGINEERING ROBOTICS

Continuing research in the area of computer control of physical processes has focused on the following problems:

1. The study of algorithms for scheduling multiple autonomous control tasks on a fixed number of processors;

2. The efficient implementation of these algorithms, and particularly their incorporation into high level compiled languages designed for process control;

3. The establishment of a laboratory environment in which software tools may readily be applied to physical processes for evaluation and demonstration.

Progress in each of these areas is described in a following subsection.

1. Scheduling Algorithms

The approach to control taken here features the use of multiple autonomous control tasks, each roughly corresponding to a classical servo control loop. Each task or daemon constitutes an independent locus of control and is, at least conceptually, executed continuously by the system. Thus a program to balance on inverted pendulum might contain daemons, assigned to the relatively independent tasks of maintaining balance in the x and y directions respectively.

Requests for service (daemon activations) may arise as the result of either programmed requests or external inputs (e.g. sensors); hence no assumptions regarding the synchrony or periodicity of daemon activations may be made in the general case. Associated with each request is a hard deadline (in absolute time) by which the request must be serviced. The theoretical problem of scheduling tasks so as to guarantee that all deadlines are met is consequently of some practical interest here, and has been the subject of investigations by Dertouzos and Geiger.

In cases involving a single processor and no a priori information regarding computation times, the "Earliest Deadline" algorithm has been shown to be optimal (in the sense that this algorithm fails only in those cases where every algorithm fails). Ongoing research (by Mok) is directed toward extending this result to cases involving multiple processors and additional a priori information (e.g. regarding computation times and distribution of requests over time).

2. Implementation of Robotics Languages

The initial implementation of a "daemonized" robotics system was completed in the fall of 1973 by Geiger. This system extended the PDP11 assembly language by a set of macroinstructions for the specification of daemons and control of sensors and actuators. Programs written using this implementation run directly on the timeshared PDP11/45.

An ALGOL compiler adapted to robotics use is currently under development by Terman. This implementation will feature code generators for several microprocessors (in addition to the PDP11), providing the DELPHI system with effective means for the production of microprocessor software. Significant technical problems being attacked in this project include the reconciliation of the ALGOL stack structure with the multiple loci of control dictated by the daemon structure.

An initial implementation of the Robotics ALGOL is expected to be operational during the fall of 1974.

3. Development of Robotics Laboratory

During the 1973 reporting period, development of a Engineering Robotics Laboratory was initiated. The laboratory uses the M.I.T. Electrical Engineering Department PDP11/45 DELPHI computer system, which has been augmented by additional memory and software for this purpose.

Our goal is to provide, through this laboratory, a "Mechano" environment in which a wide variety of representative physical devices may quickly and easily be interfaced to a controlling processor. The project involves design and construction of a variety of sensor and actuator modules, with a suitably general interface so that the modules are readily interchangeable.

An initial design of the interface has been operational since the fall of 1973. This preliminary version allows a number of plug-compatible sensor and actuator modules to be controlled by the PDP11/45 directly; future versions will include local microprocessors to relieve the timeshared 11/45 of the real time control task. This interface is currently used to control two operational experiments: (i) an inverted pendulum balance; and (ii) a recorder-playing apparatus.

A standard set of sensor and actuator modules is currently being developed by Malvania; this work is expected to be completed by December of 1974.

C. GRAPHICS

During 1973 there has been work on two projects in the area of Computer Graphics: the language DALI (described as continuing research in the report submitted for 1972-73) was completed by Gregory Pfister, and the development of a graphical animation system for the PDP11 was initiated by Gary Sockut.

DALI (Display Algorithm Language Interpreter) is a special purpose programming language for the creation and control of changing pictures which exhibit complex static and dynamic interactions among their elements. DALI allows complex organizations of interpolated ("smooth") change, discrete change, and change in the structure of a picture to be generated in a modular way, in the sense that picture elements determine their own behavior and hence manner of change.

In DALI, pictures are composed of elements called picture modules. These are analogous to procedural activations or processes, and contain arbituary event-driven procedures called daemons. Daemons are run under the control of global scheduling rules based on the functional dependence of daemons on one another.

These rules result in smooth inter-daemon (process) communication and cooperation with no implicit or explicit reference to semaphores or other synchronization primitives in user code, while at the same time providing for a high degree of parallelism. Circular inter-daemon functional dependence is possible, and results in iteration or relaxation. The environment structure used is predominantly stack-oriented.

The system currently under development by Gary Sockut will run on a PDP11/45 connected by a relatively low speed link, to a DEC GT40 display. As the GT40 display includes a local processor, the effective display of dynamic pictures necessitates running certain of the animation programs locally; the system thus includes provision for communication of procedural display data to the GT40 along with more conventional picture descriptions. The system is expected to be operational by September 1974.

D. DEVELOPMENT OF TIMESHARING SYSTEM

The initial phase of the DELPHI Timesharing System, funded by the M.I.T. Electrical Engineering department, was completed in January 1973 and is described in the report submitted for 1972-73. The system has provided reliable service to 6.031 students since the Spring of 1973.

The additional use of DELPHI for research in Robotics has necessitated its further expansion and development. Significant improvements made during the past year include:

1. Expansion of primary memory (core) to 104K words;

2. Reorganization and generalization of the mechanism for dynamic allocation of memory;

3. Implementation of a general file system.

The initial DELPHI implementation was highly specialized for the limited requirements of 6.031 students. Its evolution over the past year, in response to the requirements of more sophisticated Robotics users, renders DELPHI a system of respectable general utility without compromising its usefulness as as economical resource for large numbers of students. Much of the recent development has been directed toward providing an environment amenable to the efficient use of high level compiled languages, e.g. the sharing of pure portions of user-compiled programs. Currently under development are:

1. BCPL, an initial implementation of which has been installed; and

2. ALGOL, which is expected to be operational by September 1974.

The ALGOL compiler will serve as a host language for a Daemonized Robotics system (see section A).

Future plans for this system include the expansion of its secondary storage from two single-platter disk cartridges to one or more multiplatter disks. Further upgrading of the system software will involve recoding much of the system in a compiled language, probably a derivative of BCPL.

E. LANGUAGE SEMANTICS

Research by Ward during the past year has been directed toward the semantics of applicative languages, in which there is an assumed correspondence between procedures of a language and abstract mathematical functions. Principal results of this work include the development and semantic justification of two new applicative constructs: EITHER and *-conversion.

The syntactive mechanism of *-conversion provides means for reduction of applicative expressions to approximations of those expressions, resulting in a syntactic relationship between expressions which seems closely analogous to the semantic constructions of Dana Scott. The addition of *-conversion to the lambda calculus allows every expression to be reduced to one or more normal forms, and it has been shown that the semantics of an expression x in the lambda calculus is completely characterized by the set of normal forms derivable from x. This result provides a new technique for proving the (extensional) semantic equivalence of expressions in applicative languages.

The EITHER construct allows the expression of certain functions which are inexpressible in the conventional lambda calculus. The semantic and implementational interpretations of EITHER are, respectively:

1. EITHER{a,b} corresponds, in the semantic lattice of Scott, to the least upper bound of the elements a and b. Thus EITHER provides unique least upper bounds for sets of semantically distinct expressions, a provision which is conspicuously absent from the Scott formalism.

2. The pragmatic interpretation of EITHER{a,b} suggests the dovetailed evaluation of expressions a and b. Thus EITHER provides an applicative model for multiprocessing.

Plans for future work in this area include further exploration and formalization of relationships between these mechanisms and the Scott constructions. In addition we intend to explore the possibility of eliminating the distinction between the respective semantics of a language and its operating system by thoroughly integrating their interpretative mechanisms. This approach would, for example, combine

file system and dynamic environment structures into a single, uniformly accessible hierachy.

Publications

1. Geiger, S. P., A User's Guide to the Macro Control Language, Project MAC, TM-36, December 1973.

MATHLAB GROUP

Academic Staff

Prof. J. Moses
R. J. Fateman
V. S. Pless
P. S.-H. Wang

DSR Staff

R. A. Bogen
J. P. Golden
J. P. Jarvis
R. Schroeppel
J. L. White

Graduate Students

R. T. Genesereth
D. L. Grabel
J. Kulp
B. M. Trager
D. Y. Yun

Undergraduate Students

S. M. Macrakis
E. C. Rosen
G. L. Steele
R. E. Zippel

Support Staff

J. S. Lague
G. B. Moore

Guests

A. Miola
H. Watanabe
U. Pape

MATHLAB

A. INTRODUCTION

Implementation of Project MAC's SYmbolic MAnipulator, MACSYMA, began in July, 1969. The system has quintupled in size since the first paper describing it appeared in 1971 [20]. It therefore seems appropriate to describe the goals of the project and its major features once again. We first describe some of our early design decisions and how, in retrospect, they fared. We then indicate the major features of the current version of MACSYMA. We assume that the reader has some familiarity with features present in other algebraic manipulation systems.*

The original design decisions for MACSYMA were made in 1968 by C. Engelman, W. Martin, and J. Moses. The system was intended to be useful to a wide variety of users without losing much efficiency in running time and working storage space. The emphasis of the design decisions for MACSYMA were on ease of time-shared interaction with batch operation available for production runs. The original design assumed we would use algorithms (GCD, factorization, integration, simplification) known in 1969. Because we realized there were faults in the known algorithms, such as the lack of generality or basic inefficiencies, we began research on new algorithms. As a result, the MACSYMA system contains a number of uniquely powerful and efficient algorithms in many areas of algebraic manipulation, and much of whatever has been produced elsewhere.

The original design assumed that users with relatively small problems wanted a great deal of built-in machinery so that the solution time using a computer would be significantly less than that of a pencil and paper calculation. Users with large problems were presumed willing to spend more time optimizing their programs in order to achieve space and time efficiencies. Several distinct representations for expressions were considered necessary in order to achieve such efficiencies.

The system utilizes four major internal representations; general, rational, power series, and Poisson series. The general representation is the default representation for expressions. It offers great flexibility and is quite useful in interactive situations since the internal form of the expression is quite close to the displayed form and the user's input. The rational representation is designed for greater efficiency and offers a canonical representation needed in many algorithms (e.g. GCD). Several generalizations of this representation exist (e.g. factored form), which give greater efficiencies in space and time in certain situations. The power series representation is used mostly in obtaining a Taylor (or Laurent) expansion of a function

---

*The bulk of this report will appear in a paper by Joel Moses entitled: "MACSYMA - The Fifth Year", Proceedings of the EUROSAM 74 Conference, ACM, August 1974, pp. 105-110.

about a point. This representation is largely a polynomial representation with rational functions as coefficients. The Poisson series representation is used for manipulating large expressions involving only polynomials and trigonometric functions.

We realized that a very large system would result from the decision to use multiple representations and many built-in facilities. We assumed that memory costs would decrease markedly in order to make such a system economical. MACSYMA currently resides on the Project MAC's "Mathlab" PDP-10 which has 512K of 2 usec memory and on the MULTICS system operating on a Honeywell 6180. PDP-10 memory costs have decreased in our experience from $.03/bit to $.01/bit in the last 3 years while memory speed has increased almost three-fold in that period. A much greater decrease in costs, though with no comparable speed increase, is indicated for the next 5 years due to LSI Technology. Certain IBM vice presidents have publicly predicted that the cost per bit of main memory might be as low as 0.01 cents/bit in 1980 [7, p. 220]. With such dramatic cost reductions, large systems such as MACSYMA will be quite economical in the near future.

The current version of MACSYMA requires 175,000 words of memory on a PDP-10 for the first user, including the underlying LISP system and 15,000 words of free storage. Each additional user requires 35,000 words for data and working areas. These areas may expand during a computation. An additional 65,000 words of programs may be loaded from a disk during a session.

Another effect of the large size and generality of the MACSYMA system has been the sizable number of bugs due to the interactions between modules. These bugs were mostly prevalent as new modules were being integrated into the system in past years. The system is sufficiently stable now that many projects do not encounter bugs in several weeks of daily use. Some modules (e.g. Laplace Transforms) are not relied on by any of the others. Bugs in such modules were not noted until these modules were heavily used. Since most new features are currently added only as a result of user requests, immediate use of such features is guaranteed and leads to stability in short order. Even relatively esoteric capabilities such as summation of finite and infinite series find users who will need them and experiment with them further. On the whole, the system is presently in a fairly stable state. We owe this in large part to the many users who helped us discover bugs and missing features in the past years. Their help is gratefully acknowledged.

A third effect of a large system is the difficulty users have in learning all of its features. For small problems requiring a few commands, there is little difficulty. Users have been known to solve nontrivial problems after reading a twelve page Primer. As the problems grow in complexity and efficiency considerations enter, knowledge of a large number of facilities in MACSYMA may be required and a better understanding of algebraic manipulation will be needed. We know of no easy way to surmount the educational problems that are encountered in such areas. In the past,

users working on large projects have maintained some contact with our group, either by phone or directly. As we gather more experience regarding the difficulties users have, we expect to develop a set of tutorials and automatic aids which will overcome many of the problems. Large projects will still require a local expert whose training may take several months.

The MACSYMA system was made available over the ARPA network in May, 1972 while the system was still evolving quite rapidly. Several large projects have begun using the system in the past two years. By a large project we mean one requiring several hours of interaction daily for at least six months (usually by more than one person). The largest of these projects has been the work of Professor A. Bers of M.I.T. and his students (notably J. Kulp and C. Karney) in plasma physics. This required, among other features, development of machinery for keeping expressions in a sum of vector-matrix products from simplifying (using boxes to surround them) so that one could determine the physical phenomena which contributed most to the final answer. Drs. H. Yilmaz and R. Pavelle of Perception Technology Inc. have used MACSYMA in calculations in general relativity. Some of these calculations have been fairly classical (e.g. Riemann tensors). Others involve use of many symmetry identities to simplify large expressions involving tensors. Dr. C. Andersen and his colleagues at NASA-Langely have been using the system to generate FORTRAN programs to numerically solve partial differential equations using finite element techniques. The integrations required in setting up the elements are done in MACSYMA. Professor B. Rosen of Stevens Institute of Technology and his students have used the system in studying long-range weather prediction. R. Gosper of M.I.T. has worked on techniques for improving the convergence of series. Gosper's work relies heavily on the multivariate factorization algorithm in MACSYMA.

Hundreds of other people have used the system in the past two years. While many were clearly playing with the system and learning its capabilties, there were many other significant projects done using it. Some examples of areas of application known to us are: gas chromatography, tree searching strategies, hydromechanics, statistics, optimal control, algebraic coding theory, complexity theory, nuclear reactor design. About one half of such users have been local to the M.I.T. - Harvard community, about a half have used the system through phone lines and the ARPA network. We feel that the decision to have a large and varied system was proved correct because of the markedly different needs in terms of algorithms, data representations and interactive capabilities required in the solution of these problems.

B. DESCRIPTION OF MACSYMA

To describe MACSYMA we have divided the modules comprising the system into 7 major packages. These packages vary from 15,000 words of LISP code for the Taylor series package to 35,000 words for the LISP system itself. Many of the facilities were the work of several people, but we usually indicate the major

contributors only.

1. Language and Interactive Facilities

The first feature of MACSYMA that a user will likely notice is its output formatting. The two-dimensional output module of MACSYMA's is its most stable one. It was originally written by W. A. Martin and follows his description [19] fairly closely. Probably its most novel feature over the closely related Charybdis display program in Mathlab [2] (which follows Martin's earlier approaches) is the nice break-up of expressions too long to fit on a single line of output.

The input parser, on the other hand, has changed drastically over the years. The original parser by W. A. Martin was a Knuth LR(1) parser. This was changed to a Floyd operator precedence parser by S. Saunders and E. Rosen. Our latest, and best, parser is a Pratt parser [14,29] written by M. Genesereth. We feel that the Pratt parser gives us a clean, fast, and highly extensible parser. The current syntax for the MACSYMA language is Algol-like with blocks, various FOR statements and the like.

The programming language interpreter, originally written by W. A. Martin, but heavily modified since, is LISP-like with special machinery for algebraic manipulation. Variables with no assigned value represent themselves and can become parts of algebraic expressions. Arrays may be dimensioned or undimensioned. Undimensioned array elements are stored using hash-coding techniques. Matrices, incidentally, are data objects which are distinct from arrays.

The interpreter has associated machinery allowing one to interrupt and trace user-defined functions. One can also translate such functions into LISP and compile the LISP, thus losing certain debugging capabilities. In return a significant improvement in speed (up to a factor of 50) can be had in certain cases if one is able to declare the types of his variables and functions (e.g. real, polynomial, array of rational functions). The translation program was written by M. Genesereth.

The supervisor module controls the input-output process during a session. Usually all intermediate user inputs (C-lines) and MACSYMA's output (D or E lines) are stored in memory. By setting a flag, intermediate information can be automatically stored on disk to be retrieved by the system when it is needed. Commands can invoke modules (e.g. integration) which are not normally in main memory and these will be quickly loaded. The user may store some or all results of a session on a disk to be reloaded into a fresh MACSYMA at a later date. Special control characters can interrupt the system to obtain run time statistics or debugging information. The supervisor is the work of J. Golden.

The editor allows one to correct an expression typed into the system. The parser pinpoints the location of a parsing error. The commands to the editor are

modelled after the text editor TECO [8].

The graph module allows one to plot a graph of a function. The module will automatically calibrate the coordinate system in order that the plot will fit on the printing device being used. The editor and graph modules were originally written by W. A. Martin and by J. Golden

2. General Representation - The simplifier and basic commands

All inputs to MACSYMA, function definitions included, are converted to general representation. This representation is a natural one for list structure oriented systems. Classically, LISP-based simplifiers would convert the expression X + 1 to the list (PLUS X 1). In MACSYMA this format is generalized to ((MPLUS) $X 1). Here we see that user-defined variables are lexically modified by adding a dollar sign so that they do not accidentally conflict with free variables used in the system. Furthermore, the operator PLUS is changed to an operator list. The significance of this change is noted in the simplified form of the expression which is ((MPLUS SIMP) 1 $X). The simplified expression has been sorted and its operator list contains the indicator SIMP which will prevent resimplification of the expression. After factoring the expression, we would obtain ((MPLUS SIMP IRREDUCIBLE) 1 $X) which conveys additional information about the expression.

The simplifier is based on Korsvold's general simplifier, but modified to the point of nonrecognition, largely by J. Moses [23]. Part of its task is to perform automatic conversions of numbers and data representations. For example, rational numbers are converted to integers when possible and floating point arithmetic is contagious. Likewise the rational representation is contagious. Hence multiplying an expression by a 1 represented in rational form converts the whole expression to rational form.

The operation of the simplifier is controlled by global flags. For example, the NUMER flag determines if SIN(1) is to be converted to a floating point value. The flag %EMODE will determine if constants of the form $e^{i\pi/m}$, are to be replaced by algebraic expressions.

MACSYMA knows a great deal about trigonometric functions. It knows how to simplify, SIN(π/3). It can convert SIN(5 X) to a polynomial in SIN(X) and COS(X) and vice-versa. It can convert trigonometric functions to complex exponential form and vice-versa. Much of this is true of hyperbolic functions as well. These modules were written largely by P. Wang and M. Genesereth.

The simplifier also interacts with the matrix module. Non-commutative multiplication of vectors and matrices is provided. Commutative multiplication is element-by-element as in APL. The user can decide by changing values of flags at

which point in the computation to perform the indicated operations. Boxing facilities allow one to look at several terms in a sum without combining the terms. The matrix package was written by P. Wang. The interaction with the simplifier is the work of J. Kulp.

There is a set of commands for getting at parts of an expression (e.g. numerator of quotient, third term in a sum). These allow the user to perform intricate manipulations on an expression. These commands and the the differentiation and substitution routines were written by J. Moses.

The major pattern matching module in MACSYMA is attached to the general simplifier. Pattern matching rules allow one to add to or override transformations in the simplifier. This module is the work of R. Fateman [10].

3a. The Rational Function Representation

The rational function representation is used either directly for efficiency or indirectly in major algorithms such as factorization or integration. The internal representation for the polynomial, (X + 4)Y**3 + 5 is (($Y . 1) 3 (($X . 2)1 1 0 4) 0 5). Several facts can be noted in this representation. First, it is recursive with the main variable Y having the rank number 1 associated with it, and the secondary variable X having rank 2. Second, the representation is sparse, with missing terms not present. Rational functions form a dotted pair, numerator and denominator, with a header containing the ranked list of variables. Thus the rational function corresponding to the polynomial above is (in LISP):

((MRAT SIMP (($X . 2) ($Y . 1))) ($Y . 1) 3 (($X . 2) 1 0 4) 0 5) . 1)

It is possible to use rational functions with different ranking for variables in the same computation. The variables in the rational function are not limited to atoms (e.g. X, Y). Any expression which is not a sum, product or integer power can be used as a variable in the representation,(e.g. SIN(X+1),e**X/3). Factored representation, similar to that in ALTRAN [5], is also available. Thus, X/(X+1)**3 can be operated on without being expanded. Factored representation can be of great value in many computations by avoiding GCD computations. Research is under way on a canonical representation using partial fractions.

The polynomial representation is due originally to W. Martin. The rational function representation is due to R. Fateman. The factored representation is the work of B. Trager.

3b. Major Algorithms for Polynomials and Rational Functions

MACSYMA contains most of the major algorithms for manipulating polynomials and rational functions. In the case of the GCD algorithm a user can choose between the Reduced [4], Modular [4] or EZGCD algorithms, the EZGCD algorithm being the default algorithm. MACSYMA's factorization algorithm is based on Berlekamp's mod-p factorization algorithm. The EZGCD and the factorization algorithms rely on an extension of the Hensel lemma approach for factorization originally suggested by H. Zassenhaus. These algorithms are due to Moses and Yun (EZGCD) [28,34] and Rothschild and Wang (factorization) [32].

The factorization algorithm has been extended to coefficients which are algebraic numbers (i.e. given as roots of polynomial with integer coefficients). This algorithm relies on Berlekamp's latest factorization algorithm for factorization mod p**r [1], r the degree of the algebraic number. The implementation of this algorithm is being completed by P. Wang. The algebraic number manipulation algorithms are due to B. Trager.

Currently there are two resultant algorithms in MACSYMA: one is based on the Reduced GCD algorithm and one on Collins' modular version [6]. Research is under way to find alternative algorithms. The resultant algorithms were written by B. Trager.

Algorithms for the solution of linear equations present in MACSYMA are Lipson's variant of Gaussian elimination [18] and Gentleman and Johnson's [15] variant of the minors method. The Lipson algorithm was written by P. Wang.

The method of solving systems of polynomial equations by eliminating variables was written by D. Yun [35]. His approach relies on factorization to reduce the complexity of the intermediate systems and a resultant algorithm for performing the elimination. Due to the inefficiency of existing resultant algorithms and the growth of the degree of intermediate systems, this algorithm is useful for small systems only.

Experiments in the utilization of discrete FFTs and various fast multiplication schemes for polynomials have used this subsystem [3]. In particular cases (dense polynomials), alternative polynomial multiplication and powering algorithms are available [12].

The MACSYMA command SOLVE attempts to determine which of several solution techniques is most appropriate to a given input. Linear and polynomial systems of equations are solved using techniques already mentioned. Single polynomial equations will be factored. If the factors are of degree ≤ 4, the appropriate formulas will be used.

The RADCAN is a powerful simplification algorithm. The algorithm is, in our

terminology, a regular one. That is, it determines a set of algebraically independent expressions from which to obtain an expression equivalent to the original expression using rational operations. The algorithm is regular for all expressions involving logarithms and exponentials. It is canonical in certain situations as well (e.g. first order exponentials). RADCAN contains as a subcase an efficient canonical simplification algorithm for roots of polynomials. Unfortunately, this subcase can not handle even roots of unity in an entirely canonical manner. RADCAN is used both as a simplification algorithm and as a front end to the Risch integration algorithm. RADCAN was designed and implemented by R. Fateman [11].

4. The Integration Subsystem

The integration subsystem in MACSYMA is composed of 5 major modules: integration of rational functions, the SIN indefinite integration program, the Risch integration algorithm, the limit program and the definite integration program WANDERER. The first three are the work of J. Moses, the last two are due to P. Wang.

The method for integration of rational functions is fairly classical [24]. It uses a square-free decomposition of the denominator followed by partial fraction decomposition. Irreducible polynomials of degree ≤ 2 are solved and the logarithmic parts for these are produced. A new partial-fraction algorithm for this method is being produced by B. Trager.

The SIN program [25] has been improved by D. Grabel and modified to use MACSYMA's representations and general simplifier. Its pattern matcher, SCHATCHEN, has been made available to the rest of the system (it is used to perform simplifications involving combinatorial terms). The original third stage of SIN, a heuristic integration by parts method, has been replaced by the Risch algorithm. The special integration methods in SIN are retained because of their efficiency and, in algebraic cases, because of their power.

Our implementation of the Risch algorithm [30] handles the full exponential and logarithmic cases and by using SIN, algebraic cases of genus 0 (e.g. $\sqrt{x}$ is removed by substitution in SIN). The RADCAN routine is used to generate the ranked list of exponential and logarithms required by the Risch algorithm. After integration, trigonometric functions previously transformed to complex exponentials are reintroduced by obtaining the real and imaginary parts of the integral. Our implementaion of the Risch algorithm can also handle some special functions in its input and produce them in its output (e.g. error functions) [26]. Work on increasing the number of such special functions is under way.

The current limit program [33] relies on making certain simplifying transformations and a classification of the inputs into various classes. Different

methods for obtaining limits are available in each class. Parts of this heuristic program are about to be replaced by algorithmic versions which rely on obtaining a Laurent series of the expression at the limit point.

The definite integration module [31] also attempts to classify its input. Among the many methods available to it are several variants of contour integrations and indefinite integration. This module uses more machinery than any other module in MACSYMA.

5. The Power Series Subsystem

The power series representation in MACSYMA is similar to a polynomial representation, but allows coefficients to be rational functions. Moreover the power series representation will remember truncation information about variables. The representation is more general than a similar one in ALTRAN [5] in that it allows for negative exponents (thus obtaining a Laurent as well as Taylor series representation) and fractional exponents (thus allowing for representation of branch curves). There are only a few functions which cannot be represented in this manner at a point. Examples with which this representation has difficulties are essential singularities (e.g. SIN(1/X) at X = 0).

The TAYLOR command in MACSYMA obtains a truncated Laurent expansion of a function at a point. Rather than using the inefficient method of obtaining the expansion by differentiation, the program converts all functions in the input to power series and performs the indicated power series operations on them.

In addition to the obvious direct applications of the power series representation there is an indirect application to the computation of limits. Essentially, the only heuristics that are required in addition to the machinery in TAYLOR would be ones needed to handle essential and isolated singularities and one-sided limits. The power series subsystem is due to R. Zippel.

6. Miscellaneous Facilities

The Poisson series representation is the fourth representation of expressions used in MACSYMA. It is restricted to sums of trigonometric functions with polynomial or power series as coefficients. Its novel feature relative to other Poisson series manipulators is the use of a tree sort to optimize Poisson series multiplication times. This work is due to R. Fateman [13].

The Laplace Transform module provided for the direct transform of a class of expressions involves polynomials, exponentials, trigonometric functions, derivatives and integrals. The inverse transform of rational functions is also provided. This facility is a slight extension of a similar facility in MATHLAB [9] and is due to R. Bogen.

Heuristic methods for finding closed form sums of classes of expressions over finite (e.g. 0 to N) or infinite ranges were written by R. Zippel. There is machinery for simplification of sums of certain combinatorial terms, thus moving closer to solving one of Knuth's 50 point problems [17, sec. 1.2.6].

A third pattern matching facility in MACSYMA is a variant of REDUCE's LET facility [16]. This pattern matcher uses the rational function representation, rather than the general representation to perform the required transformation. The LET facility is due to K. Nishihara.

7. The MACLISP System

Development of a LISP system for the PDP-6/10 computers at M.I.T. was begun in 1965 by R. Greenblatt and S. Nelson. By 1967 the interpreter, support routines and compiler were sufficiently stable that the system was exported to Stanford as LISP 1.6. Improvement of LISP 1.6, largely for the needs of MACSYMA, was undertaken by W. Martin and J. White in 1968. An efficient compiler which accepts mode declarations was developed by 1972 by J. Golden, E. Rosen, and J. White. Tests by R. Fateman indicate that in certain inner loops the code produced by the LISP compiler was more efficient than the DEC PDP-10 FORTRAN's compiler. The dynamic type-checking of other LISP systems usually leads to a loss of a factor of 20-30 to FORTRAN in such situations.

Due to the large size of MACSYMA, sharing is an important issue. Since the code produced by the compiler is "pure", sharing of programs is fairly straight-forward. Our LISP, now called MACLISP [22], allows each user to decide whether he wants to be in debugging mode (and experience a factor of four slow down in speed) or in execute mode. The same code is shared in both cases, through writable transfer-vector pages.

In addition to sharing programs, MACLISP can also share fixed data (e.g. differentiation rules). About 2/3 of the data in MACSYMA is sharable. Furthermore, the impure data areas (e.g. free storage) are dynamically expandable during a computation. This partially obviates the need for a user to guess at the size of his intermediate expressions and generate a system large enough to handle the worst situation. Both large and small users share the code and pure data. The effect of all this sharing is to reduce the memory cost for each simultaneous user beyond the first to 35,000 words. Ten simultaneous users of MACSYMA are then possible without requiring swapping to slower memories. The code for sharing is due to G. Steele and J. White.

C. SUMMARY

It is difficult to do justice to a system such as MACSYMA with a report of this size. Since the system has evolved so rapidly in the past three years, it is clearly desirable to indicate, however roughly, the range of its current facilities.

Some of the facilities we have not discussed are several dealing with user-definable extensions to various modules in the system (e.g. special display formats, new differentiation rules). There are also certain unusual linguistic issues which arise in a symbolic manipulation system. For example, suppose you sum S + I with index I ranging from 0 to 5. Suppose S is a variable whose value is an expression containing I. What should be the result of the sum?

Among the credits we have omitted are those for writing the reference manual [2] (R. Bogen), the Primer [27] (J. Moses), and for over-all maintenance of the system (J. Golden).

The above summarizes the accomplishments of the past five years of research. The major activities of the past year were in the following areas:

1) Factored representation of rational functions.

2) Algorithms for manipulation of algebraic numbers.

3) Factorization of polynomials with algebraic coefficients.

4) Completion of the EZGCD algorithm.

5) Algorithms utilizing the Fast Fourier Transform.

6) Poisson series manipulation.

7) Data sharing facility in MACLISP.

REFERENCES

1. Berlekamp, E. R., "Factoring Polynomials over Large Finite Fields," Math. of Comp., vol. 24, no. 111, July 1970.

2. Bogen, R. A. et al, MACSYMA Reference Manual, version 6, Project MAC, M.I.T., Cambridge, Massachusetts, Jan. 1974.

3. Bonneau, R., Fast Polynomial Operations Using the Fast Fourier Transform, Ph.D. thesis, Department of Mathematics, M.I.T. (to appear).

4. Brown, W. S., "On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors", JACM, vol. 18, no. 4, pp. 478-504, Oct. 1971.

5. Brown, W. S., The ALTRAN User's Manual, Bell Telephone Labs, Murray Hill, N.J., 1973.

6. Collins, G. E., "The Calculation of Multivariate Polynomial Resultants", JACM, vol. 10, no. 4,, pp.515-532, Oct. 1971.

7. Datamation, May, 1973.

8. Dowson, M., How to get on the MAC/AI System, Memo 215, A.I. Lab, M.I.T., April 1971.

9. Engelman, C., "The Legacy of Mathlab 68", Proc. 2nd Symposium on Symbolic and Algebraic Manipulation, pp. 29-41, ACM, March 1971.

10. Fateman, R. J., "The User-level semantic matching capability in MACSYMA", Proc. 2nd Symposium on Symbolic and Algebraic Manipulation, pp.311-323, ACM, March 1971.

11. Fateman,R. J., Essays in Algebraic Simplification, Tech. Report 95, Project MAC, M.I.T., April 1972.

12. Fateman, R. J., "On the Computation of Powers of Sparse Polynomials", Studies in Applied Mathematics (to appear).

13. Fateman, R. J., "On the multiplication of Poisson Series", Celestial Mechanics (to appear).

14. Genesereth, M.R., A Grammar Primer for MACSYMA, Project MAC, M.I.T., July 1973.

15. Gentleman, W. M. & Johnson, S.C., "Analysis of Algorithms, A Case Study: Determinants of Polynomials", Fifth Annual ACM Symposium on Theory of Computing, Austin,s, April 1973, pp. 135-141.

16. Hearn, A.C., "Reduce-2 User's Manual", Report UCP-19, University of Utah, March 1973.

17. Knuth, D., The Art of Computer Programming, vol. 1, Addison Wesley, 1968.

18. Lipson, J.D., "Symbolic Methods for the Computer Solution of Linear Equations with Applications to Flow Graphs", Proc. of the 1968 Summer Institute on Symbolic Math. Conf., R. Tobey (ed.) IBM, June 1969.

19. Martin, W.A., "Computer Input/Output of Mathematical Expressions", in Proceedings 2nd Symposium on Symbolic and Algebraic Manipulation, pp.78-89, ACM, March, 1971.

20. Martin, W.A. and Fateman, R.J., "The MACSYMA System", Proceedings 2nd Symposium on Symbolic and Algebraic Manipulation, pp. 59-75, ACM, March 1971.

21. Miller, J.K., "CHARYBDIS: A LISP Program to Display Mathematical Expressions on Typewriter-like Devices", Interactive Systems for Experimental Applied Mathematics, Academic Press, 1968, pp.155-163.

22. Moon et al., MACLISP Reference Manual, Project MAC, M.I.T., (to appear).

23. Moses, J., "Alic Simplification - A Guide for the Perplexed", CACM, vol. 14, no. 8, pp.527-537.

24. Moses, J., "Symbolic Integration - The Stormy Decade", CACM, vol. 14, no. 8. pp. 548-560.

25. Moses, J., Symbolic Integration, Technical Report 47, Project MAC, M.I.T., Dec. 1967.

26. Moses, J., "The Integration of a Class of Special Functions with the Risch Algorithm", SIGSAM Bulletin, no. 13, ACM, Dec. 1972, pp. 14-27.

27. Moses, J., MACSYMA Primer, Project MAC, M.I.T., 1973.

28. Moses, J. & D.Y.Y. Yun, "The EZ GCD Algorithm", Proc. 1973, ACM National Conference, Atlanta, Ga., Aug. 1973, pp. 159-166.

29. Pratt, V.P., "Top Down Operator Precedence", Proc. ACM Symposium on Principles of Programming Languages, Boston, Oct. 1973, pp. 41-51.

30. Risch, R. H., "The Problem of Integration in Finite Terms", Trans. AMS, vol. 139, Mar. 1969, pp. 167-189.

31. Wang, P., "Automatic Computation of Limits", Proceedings 2nd Symposium on Symbolic and Algebraic Manipulation, ACM, March 1971, pp.

32. Wang, P. and Rothschild, L., "Factoring Multivariate Polynomials over the Integers", SIGSAM Bulletin 20, ACM, Dec. 1973, pp.21-29, and Math Computation (to appear).

33. Wang, P., Evaluation of Definite Integrals by Symbolic Manipulation, Report 92, Project MAC, M.I.T., Oct. 1971.

34. Yun, D.Y.Y., The Hensel Lemma in Symbolic Manipulation, Tech. Report, Project MAC, M.I.T. (to appear)

35. Yun, D.Y.Y., "An Algorithm for Solving Systems of Polynomial Equations", SIGSAM Bulletin 27, ACM, Sept. 1973, pp. 19-25.

Publications

1. Fateman, R. J., "A Case History of Interactive Problem Solving Systems", SIGSAM Bulletin No. 28, December 1973.

2. Moses, J., and D. Y. Y. Yun, "The EZGCD Algorithm", Proceeding of the ACM National Convention, August 1973.

3. Pless, V., "Self-Dual Codes Over GF(q) Satisfy a Varshamov Bound", (with John Pierce) Information and Control, August 1973.

4. Pless, V., "Attitudes About and of Professional Women: Now and Then", Career Guidance for Women Entering Engineering, Edited by Nancy Fitzroy, Proceedings of an Engineering Foundation Conference, New England College, Henniker, New Hampshire, August 1973.

5. Yun, D. Y. Y., "On Algorithms for Solving Systems of Polynomial Equations", SIGSAM Bulletin, September 1973.

PROGRAMMING TECHNOLOGY

Academic Staff

Prof. J. C. R. Licklider

A. Vezza

DSR Staff

A. K. Bhushan
E. H. Black
M. F. Brescia
M. S. Broos
H. I. Badian
L. G. Daniels
S. W. Galley
J. F. Haverty
P. D. Lebling
J. C. Michener
C. L. Reeve
N. D. Ryan
R. W. Weissberg

Graduate Students

P. M. Allaman
A. Chan
S. E. Cutler
B. K. Daniels
J. D. DeTreville
G. J. Farrell
J. W. Johnson
W. J. Long
G. D. McGath
H. F. Okrent
M. S. Seriff
R. A. Stern

Undergraduate Students

S. A. Bengelloun
J. H. Harris
A. G. Jaffer
J. H. Morrison
J. D. Sybalsky
G. A. Thompson
T. To
K. E. Van Sant
J. Westcott
C. K. Yap

Support Staff

S. B. Pitkin

Preceding page blank

A. INTRODUCTION

The major goal of the research and development effort of the Programming Technology Division is the automation of the technology of programming. The research of the division is directed toward the development of programming methodologies, programming tools, and programming aids that can lead to significant technical advancements in computer program production methods. Our major efforts during the reporting period have been concerned with completing the CALICO system with its well-documented library of slightly more than 2000 assembly-language subroutines, improving the programming facilities of the language MUDDLE, planning the implementation of a MUDDLE library system, planning the development of a system for automating program documentation, and designing a message system.

Work on automatic programming described in the last report has been deferred to accommodate Professor Licklider's imminent leave of absence to temporarily serve the government in Washington.

B. PROGRAMMING TECHNOLOGY

1. CALICO

Development of the programming environment CALICO and its 29 or so subsystems [1] is nearing completion. For this reason parts and aspects of CALICO will be discussed in more detail than a report of this nature would normally warrant.

The CALICO project has thus far produced a subroutine library that is large, is easily modified and enlarged, is heterogeneous (in that the subsystems produced from the subroutines in the library serve a variety of purposes) and promotes its own growth. The implicit availability of a wide variety of directly-callable subroutines visibly increases programming productivity. The increased productivity in turn encourages CALICO programmers to add to the library all new subroutines they create. Aids for on-line search and off-line maintenance of the library are available, and some principles that make the library feasible have been developed:

(1) Protocols for such things as error handling and data structuring are necessary if arbitrary subroutines are to coexist harmoniously. As the CALICO library grew, ad hoc protocols were developed as they were needed, and utilized by new subroutines. In retrospect these protocols provide excellent insight into the protocols needed in a library environment. Furthermore the experience with CALICO indicates that use of the protocols must be enforced, and they must be used uniformly. This latter requirement implies that changes to protocols must be effected in library programs retroactively and automatically. The use of a sufficiently-rich programming language makes it easier for programmers to follow

the protocols.

(2) Abstracts of subroutines -- condensed descriptions used to aid design and maintenance of software -- must be structured sufficiently so that they can be manipulated and (insofar as possible) understood by programs. Aids to help a programmer create and edit properly-formatted CALICO abstracts are available. However, experience indicates that a large portion of each abstract should and can be generated by a program that is similar to the analysis phase of a compiler, leaving the human programmer to supply only those parts to be used exclusively by humans.

(3) Subroutines and abstracts must be tested and pass standards before being accepted into the library. In CALICO there are three stages of subroutine testing: by the author (the subroutine residing in a personal disk area), by the group (in a "development" library), and by the outside world (in the standard public library). In addition the subroutine and abstract are evaluated by three staff members -- with all the fallibilities of humans -- for aspects of quality: format, protocol, correctness, generality, efficiency, etc. Our experience indicates that, for library systems to work well in an operational environment, much of the validation of the structure and protocol used in programs and abstracts currently performed by humans can and must be done by programs.

(4) The library must be easy to use by both humans and programs. This dictum applies to all aspects: finding a subroutine to do a given task, specifying a call to it, loading it, finding bugs in it or (usually) in its caller and reporting the former to the responsible party, submitting a new subroutine, updating an old one, discovering the side effects of an update, publicizing the update or addition, protecting the subroutine data base from accidents.

a. Events of the year

The CALICO subroutine library is central to the methodology of programming in the CALICO environment. During the year, the CALICO library was cleansed (Broos, Galley, Michener, Haverty, Lebling). All obsolete entries were expunged and most non-obsolete entries without documentation were abstracted. The library clean-up resulted in a program library that is better than 95% abstracted. The cleansing expunged several hundred subroutines; even so, the size of the library increased by 642 subroutines during the year from 1412 to 2054 (as measured by the abstracts in the abstract library).

Four additional subsystems, BATCH (Seriff, Morrison) [2], TAILOR (Seriff) [3], CONDIT (Seriff) [4], and RUN (Galley) [5], became operational, and a fifth, a rudimentary COMSYS (Haverty) [6], was implemented. In addition, the console interface was redesigned (Seriff, Galley, Lebling, Michener, Bhushan, Haverty, Vezza, Broos) and

implemented (Seriff) [7].

The CALICO BATCH subsystem provides a facility for: (1) automatic rescheduling of tasks that are run periodically, and (2) future scheduling of tasks to be run during slack periods for absentee users. The primary reason for the existence of BATCH resulted from the need to schedule periodically (daily, weekly and monthly) certain computer tasks to relieve the creative programmer of the burden of performing repetitive, mundane tasks. A partial list of such tasks includes: program and abstract library updates (daily); retrievals from the Datacomputer and graphing of host availability data (daily, weekly, and monthly); weekly retrieval, from the OFFICE-1 computer, of the official host list for updating the SURVEY and other local programs' host lists; personal directory house-keeping tasks. The future scheduling feature provides for a more uniform distribution of load. Tasks of this latter type are typically compilations, cross-reference listings and assemblies. The BATCH processor is capable of running any job in the ITS environment without modification. Jobs run under BATCH perform their console I/O through a pseudo-console.

The TAILOR and CONDIT subsystems taken together provide a full macro capability for the CALICO command interpreter (it goes beyond simple string or symbol substitutions). TAILOR allows a user to 'tailor' the user command interface to suit his or her personal idiosyncrasies and also to define new commands in terms of two or more of the existing ones. The CONDIT subsystem provides for the capability of conditional commands in a stored command sequence that specifies the TAILORed CALICO.

The RUN subsystem provides a CALICO user with the ability to run one job inferior to the CALICO job. It was implemented mainly to provide the library subroutines necessary to allow the BATCH processor to support an inferior job. COMSYS will be discussed in greater detail in the section on the PTD Message Facility.

The new CALICO console interface provides a uniform interface throughout the CALICO subsystems for both programs and users. All programs that receive input or send output to the console do so through the standard console interface. The standard console interface obviates the need for designing and implementing a console interface for each separate subsystem and also makes easier the task of providing a user interface that is uniform in appearance, throughout all the CALICO subsystems. Thus, if a user learns how to use one subsystem, that knowledge can be applied to learn about other subsystems. Features of the CALICO console interface are these:

(1) An enforced standard prompting scheme on all requests for user input. A prompt always consists of two parts: a semantic part, telling the user what meaning will be attributed to what is typed, and a syntactic part, telling in what form it should be typed. The following is a typical prompt for input:

name of item (SYM):

This tells the user that what is typed will be interpreted as the 'name of item'. It also says that symbol input is currently available.

(2) Symbol completion for symbol input. When the input expected by the program is a symbol, a user has the option of only partly specifying the input, with CALICO supplying the unspecified characters. In addition, should a user not know what can be input, the character control-F can be typed at any time. CALICO will respond by listing the name of each current symbol that begins with the characters that have been input thus far. If nothing has been input, all current legal symbols are listed.

(3) A mechanism to allow a user to gain additional information about required input. If a user types '?' immediately following a prompt for input, CALICO will respond by typing a more verbose (2 or 3 lines), and hopefully more informative, prompt. If the user types '?' again, then CALICO will check its library of 'help messages' to see if there is one associated with the current input. If there is, the user will see the first few lines of the message. Each additional time that the user types '?', a few more lines of the help message will be seen.

(4) Erasure of single characters, words, lines or the entire buffer, and redisplay of the entire buffer (with leading prompts).

(5) Flexible appearance. The TAILOR subsystem allows almost all of the characters used by the command READER to be modified (including terminators, deletion characters, etc.). Using TAILOR and CONDIT, CALICO can be made to look like many different systems, for example, the TENEX monitor. A user can, therefore, make CALICO behave in a way that suits his or her exact idiosyncratic needs.

b. Scenario

A CALICO program abstract can describe a single subroutine or a collection of subroutines, each of which is described by an abstract. The concept of a CALICO abstract is simple: it is supposed to be a definitive statement about what a subroutine does -- not necessarily how it does it, albeit that may be of paramount importance in some instances. The abstract attempts to provide enough information for a programmer to decide whether a subroutine would be useful as a part of some larger program, and, if so, how to use it [8].

The programming methodology employed by CALICO programmers undertaking a programming task typically follows this sequence: (1) create a rough design of the programming task; (2) survey the library for subroutines that are

potentially useful; (3) refine the design; (4) select from the library appropriate subroutines to become part of the program; (5) code the top-level program and additional subroutines necessary to complete the task; (6) test and debug the program; (7) modify the design and appropriate parts of the program if warranted; (8) document the top-level program (or collection) and all new lower-level subroutines created; (9) submit the program and all new subroutines and abstracts to the library. The process of surveying the program library is aided by a general information retrieval system, IRS (Broos) [9,10]. IRS is designed to be interactive so the documentation about a program can be easily obtained when needed. An inverted file data base is used to insure rapid retrieval. Using IRS, a programmer can locate subroutines in the library that are appropriate candidates for incorporation into some larger program.

Upon hearing of the subroutine library, visitors often ask "What kind of subroutines are in it?". Unfortunately, this is very analogous to the question "What does Macy's sell?". Common sense prevents anyone from trying to enumerate, and a simple example or two would more often than not give the wrong impression. For instance, telling an uninitiate that the IRS subsystem is composed entirely of CALICO library subroutines very often gives the quite erroneous impression that the library is composed almost wholly of routines used by IRS. Because IRS is useful in extracting from the computer system not only the items in its data bases but also some general information about the aggregate, IRS can be used to shed some light on what kinds of subroutines make up the CALICO library. The capabilities of IRS are perhaps best illustrated by a scenario. The scenario will be used to illustrate aggregate data about the library and how an individual might use IRS to aid a programming task.

In the following scenario, user input of commands is indicated by upper-case portions of lines beginning with a '@' (plus a 'space', that doesn't print on the console, at the end of each upper-case portion). Upon input of the 'space', CALICO responds by completing the remaining unambiguous portion of the symbol. (CALICO indicates that the symbol is still ambiguous by printing a dollar sign ($) on the console. This dollar sign disappears when the user types another character.) When the command specification is complete, the user types: (1) a break character indicating a wish to activate it (in this instance, the CALICO was TAILORed to allow 'space' to be a break character in addition to a completion character -- in an unTAILORed CALICO 'escape' is the break character); (2) the argument, if the command requires one; and (3) an 'escape' after the argument if an argument is required. (The reader may be struck by the length of each command name and the verbosity of the output and think that, even though completion is provided, waiting for it to print would be tedious. Such is not the case because all local programming consoles are displays capable of communicating with the central computer system at a 2000 character/second rate.)

First some general CALICO features are illustrated to show what facilities (commands) IRS provides. Second, some information about the subroutine library

aggregate will be extracted. Third, a subroutine to do a specific task will be found. The parts of the scenario are delimited by three dashes with descriptive prose between each section.

To look at what facilities IRS provides, its command tables are printed by using the COMMANDS command.

@COMmands # (int): 0

CURRENT CALICO COMMAND TABLES:

#	TITLE
1	INFORMATION RETRIEVAL SYSTEM USER COMMANDS
2	IRS CALICO SUBROUTINE LIBRARY USER COMMANDS
3	IRS LEVEL MANIPULATION COMMANDS
4	LIBRARY ACCESS COMMANDS
5	PERSONAL COMMANDS
6	CALICO DEBUGGING COMMANDS
7	GENERAL SYSTEM COMMANDS
8	GENERAL CALICO COMMANDS

@COMmands # (int): 1

INFORMATION RETRIEVAL SYSTEM USER COMMANDS

irs.simple.search irs.basic.search irs.old.find.object
irs.search irs.distribution.table irs.up.level
irs.down.level irs.top.level irs.print.status
irs.print.object irs.print.current.level irs.list.class.names
irs.list.attributes irs.list.association.attributes
irs.activate.standard.data.base irs.activate.data.base
irs.deactivate.data.base

@COMmands # (int): 2

IRS CALICO SUBROUTINE LIBRARY USER COMMANDS

irs.print.abstract

@COMmands # (int): 3

IRS LEVEL MANIPULATION COMMANDS

irs.frequency.table irs.sort.current.level.by.frequency
irs.prune.current.level.by.frequency irs.save.current.level
irs.list.saved.levels irs.restore.saved.level

```
        irs.merge.saved.levels

@COMmands # (int):  4

LIBRARY ACCESS COMMANDS
        library.print.file     library.retrieve.file
        library.print.abstract     library.retrieve.abstract
        library.print.irs.info     library.retrieve.irs.info
        library.find.file     library.check.file     library.list.files
        library.new.files     library.bad.files
        library.get.listing.tab.number

---
```

Now some characteristics of the CALICO subroutine library data base are illustrated.

```
---

@IRs.List.Class.names

ARGUMENT.TYPE
AUTHOR
CATEGORY
DESCRIPTOR
EXTERNAL
GLOBAL
INSERT.FILE
NAME.OF.ABSTRACT
OBJECT.TYPE
RESULT.TYPE
SOURCE.FILE
STATIC.VARIABLE

---
```

These are the names of the abstract fields that are inverted. Most names are self-explanatory; EXTERNAL means other subroutines that are called by a subroutine (in the CALICO environment any subroutine or collection that calls another subroutine must declare it external); OBJECT.TYPE tells whether the subroutine is mediated at call and return time, or unmediated, or a display subroutine; STATIC.VARIABLE means a variable accessible only to the subroutine, in which it can store information between calls to it.

The categories into which a subroutine falls are assigned by the programmer by choosing from a controlled list. Most of the subroutines with no category

(NO.ATTRIBUTE) are old ones that joined the library prior to the time the category field was added to the abstract. The distribution of subroutines into categories is shown next:

@IRs.DIstribution.table over (sym): CAtegory

ATTRIBUTE	FREQUENCY	PERCENTAGE
UTILITY	725	35%
I/O	400	19%
DATA.MANAGEMENT	254	12%
DISPLAY	237	11%
DATA.TYPES	225	10%
STRING	183	8%
CHILL1	150	7%
NO.ATTRIBUTE	132	6%
CARE/C2	131	6%
STAT/MATH	116	5%
NETWORK	98	4%
INTERRUPTS	13	0%
SIGNALS	8	0%

Descriptors applicable to a subroutine are assigned freely by the programmer. The common ones are listed next:

@IRs.DIstribution.table over (sym): DEscriptor

ATTRIBUTE	FREQUENCY	PERCENTAGE
DATA	263	12%
NO.ATTRIBUTE	185	9%
DISPLAY	173	8%
STRING	172	8%
VECTOR	138	6%
FILE	138	6%
ARRAY	131	6%
OUTPUT	121	5%
CHARACTER	103	5%
PRINT	99	4%

DISK	92	4%
BLOCK	90	4%
COMMAND	85	4%
LIBRARY	84	4%
DICTIONARY	83	4%
IRS	82	3%
CHILL	78	3%
SEARCH	77	3%
LIST	76	3%
ASCII	74	3%
INPUT	71	3%
BUILD	71	3%
TYPES	70	3%
READ	70	3%
NLS	68	3%
ESP	67	3%
FUNCTION	66	3%

.
.
.

By examining the distribution of authors, one finds that three authors contributed nearly half of the subroutines in the library, but 25 different authors contributed something:

@IRs.DIstribution.table over (sym): AUthor

ATTRIBUTE	FREQUENCY	PERCENTAGE
JFH	376	18%
MS	354	17%
DL	270	13%
JM	242	11%
MSS	212	10%
SG	122	5%

.
.
.

The next distribution table gives an idea of how interdependent the subroutines in the library are. The first column contains a subroutine name, the second the number of subroutines or collections that declare it external (call it) and the third the percentage of the library that this number represents. Note that NO.ATTRIBUTE indicates the subroutines that call no others, that is, bottom-level routines. An interesting point about the subroutines most often EXTERNed, that is, DATREL (data release), VCTBLD (vector build), etc., is that they are part of the data type system of CALICO.

@IRs.DIstribution.table over (sym): EXternal

ATTRIBUTE	FREQUENCY	PERCENTAGE
DATREL	577	28%
VCTBLD	314	15%
NO.ATTRIBUTE	284	13%
INTBLD	273	13%
BLKBLD	206	10%
DATCPY	196	9%
VCTAPI	181	8%
DATCPD	153	7%
RDWANT	106	5%
CHERR1	94	4%
VCTINF	90	4%
VCTINI	80	3%

.
.
. .

The list is of course very long. Using the entire list, one discovers that:

8 subroutines are called by at least 100 others;
30 " " " " " 50 others;
134 " " " " " 20 others;
410 " " " " " 10 others.

The total number of external references is 13 690, of which about 45% are declared external in a subroutine written by an author other than the one who wrote the subroutine EXTERNed.

Some statistical properties of the (static) trees implied by (unexecuted) call statements from one program to another within the CALICO library were also obtained. A naive measure of such a tree is its depth and width, measured in nodes. We

considered each assembler source file ("program") to be an atomic unit, since it is too difficult to tell, in unstructured assembly language, which control paths leading from which entries into the program can actually execute a particular call. This simplification also implies that the various entries into a program (which may differ only in the number or kind of arguments passed) are not considered distinct. The 2054 entries in the library are reduced to 1316 programs. Of these, 258 (19%) are "top-level", in the sense that they are never called by any other program. These top-level programs are designed as such, to be called only by a console user.

Given the programs potentially called by each program in the library (the average program contains calls to 4.3 programs), the 258 "call trees" can be constructed. It turns out that the size of the forest is enormous, even if recursive branches are pruned off. A lack of (computer) time precluded exploring it in depth, but the partial results are interesting. Two deep static call trees were explored. The depth of one was 34; the other 39. The inner tree silhouette shown in Figure 1 was obtained by taking the logarithm of the sum of the node counts at each level of calling for these two trees. The maximum node count is 254 213 at a call depth of 23. In addition, all call trees were explored to a depth of seven. The solid part of the outer silhouette shown in Figure 1 shows the corresponding node counts. The dashed part is the hypothesized extrapolation of the shape of the entire forest. Counting nodes at each level, the forest appears to be pear-shaped, with its maximum width below 20 levels down; the first seven levels have node counts of 258, 922, 5 495, 25 855, 102 179, 348 712, and 1 136 232. The maximum width of the hypothesized forest appears to be on the order of 10**9 nodes. It is possible that a good number of these calls are within the CALICO data system, or the CHILL interpreter, or in other functional areas that would not appear in the MUDDLE library. But even ignoring those calls, a complete examination of the reduced forest is still too time-consuming.

The next part of the scenario will illustrate the use of IRS to find a subroutine to do a specific task. Suppose one needs a subroutine that will find a string in a set of strings. The STRING category would probably be examined first. Note that '?' is typed at one point to get an additional prompt for the current input. The syntactic prompt '(mult-sym)' means that multiple symbols can be input for an argument.

READY AT TOP LEVEL WITH 2054 OBJECTS IN DATA BASE.

@IRs.SImple search on (sym): CAtegory applying (sym): ?
Specify logical operation to be applied in search. Default is "OR".
(Symbol is acceptable.)
: Or to (mult-sym): STRing

READY AT LEVEL 1 WITH 182 OBJECTS.

SEMILOG$_{10}$ SILHOUETTE OF CALICO-LIBRARY STATIC CALL FOREST

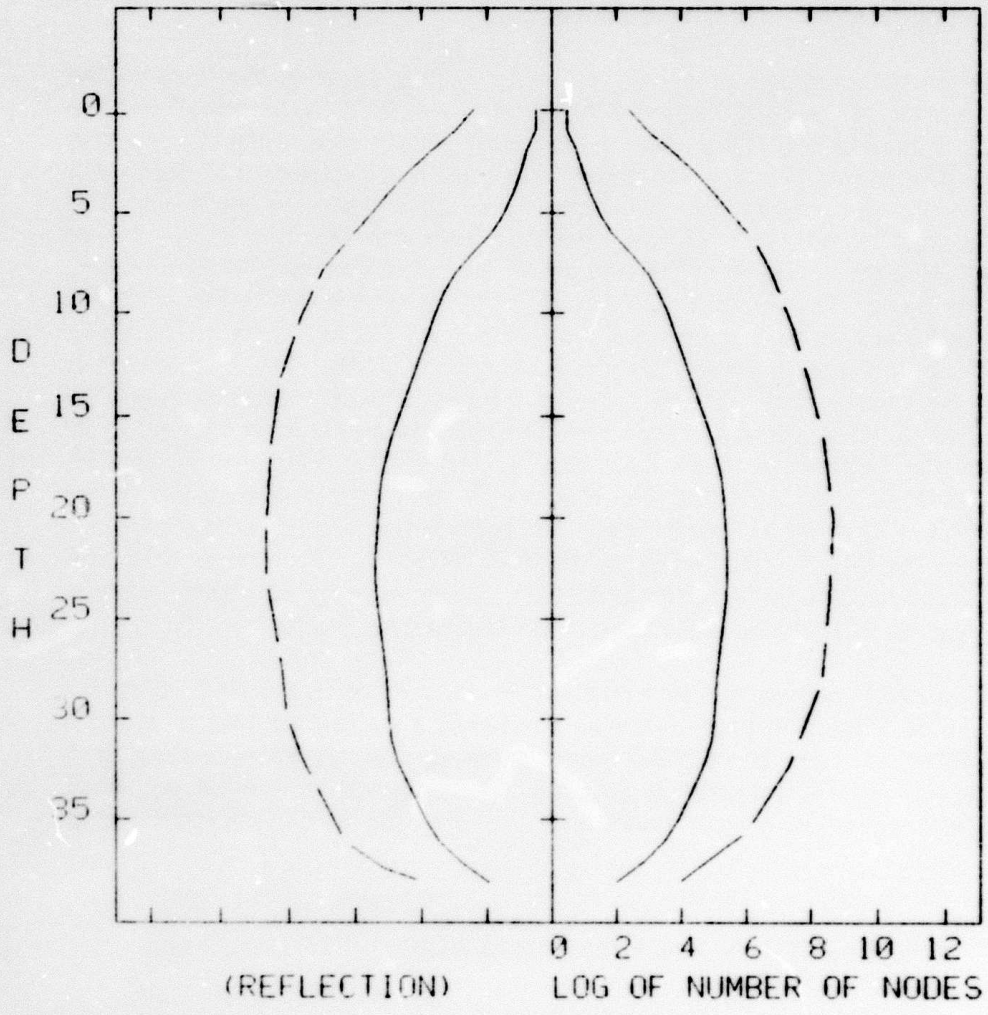


FIGURE 1.

```
[["OR" "CATEGORY" "STRING"]]

---
```

Now one can get more specific by using descriptors. For illustration purposes, a different IRS searching command is used, which allows arbitrary Boolean combinations of search conditions but does not provide completion of partially-typed names. (The terminator for the search request is represented by '$'.)

```
---

@IRs.Basic.search with (txt):  ["AND" "DESCRIPTOR" "SEARCH" "EQUALITY"]$
                    *** NAMES OF OBJECTS FOUND ***
FXDSER
FXDSSV
SERCH
VSSRC

READY AT LEVEL 2 WITH 4 OBJECTS.
[["OR" "CATEGORY" "STRING"] ["AND" "DESCRIPTOR" "SEARCH" "EQUALITY"]]

---
```

The names of the four subroutines which are possible candidates for the task were output because the list was shorter than twenty. Printing their descriptors gives an idea of their characteristics.

```
---

@IRs.Print.Current.level class (mult-sym):  Name,
  class (mult-sym):  Descriptor

                    (Frequency = 1)
NAME:               FXDSER
DESCRIPTOR:                         SEARCH, PATTERN, BYTE POINTER, SUBSTRING,
                    STRING MANIPULATION, CHAR, ALTERNATIVES, TECO, ASCII, SIXBIT,
                    MATCH, COMPILE, COMPARISON, EQUALITY, TESTING

                    (Frequency = 1)
NAME:               FXDSSV
DESCRIPTOR:                         SEARCH, PATTERN, BYTE POINTER, SUBSTRING,
                    STRING MANIPULATION, CHAR, ALTERNATIVES, TECO, ASCII, SIXBIT,
                    MATCH, COMPILE, COMPARISON, EQUALITY, TESTING

                    (Frequency = 1)
NAME:               SERCH
```

DESCRIPTOR: SEARCH, PATTERN, BYTE POINTER, SUBSTRING, STRING MANIPULATION, CHAR, ALTERNATIVES, TECO, ASCII, SIXBIT, MATCH, COMPILE, COMPARISON, EQUALITY, TESTING

(Frequency = 1)
NAME: VSSRC
DESCRIPTOR: STRING, VECTOR, COMPARISON, EQUALITY, SEARCH

READY AT LEVEL 2 WITH 4 OBJECTS.
[["OR" "CATEGORY" "STRING"] ["AND" "DESCRIPTOR" "SEARCH" "EQUALITY"]]

All but VSSRC have identical large descriptor sets, indicating that the three constitute a subroutine collection of some power. In addition, the descriptor SUBSTRING indicates that the collection may be too powerful for the intended purpose. (Of course the abstract(s) could be examined to confirm this notion.) VSSRC seems to be the prime candidate, so its abstract is printed. (The abstract format and semantics, including the meaning of abbreviations and jargon (such as 'std' for standard and 'tptr' for typed pointer, etc.) are all set forth and explained in a set of documents known as Convention II [8,11-15]. Briefly, the physical form of the abstract is a set of comment lines (starting with a ';') at the beginning of the source program. The parts of the abstract are indicated by various heading and spacing conventions which must be strictly adhered to, lest the abstract parser -- which sees only a stream of characters -- fail to find everything.)

GIRs.Print.Abstract called (sym): VSSRC

```
;---

TITLE 'SSRC MJM005  J. MICHENER (JM) 3 FEB 73

;                    *** VSSRC SIMPLEX ***

; ABSTR

; VECTOR OF STRINGS SEARCH

; SEARCH FOR A STRING AMONG A VECTOR OF STRINGS

; LANG:  MIDAS

; VSSRC:                          This rsr accepts a std string and a std vector of std
```

```
; strings.  It compares the std string to each element of the std vector
; (using STRCEQ (REF1)) and returns the index of the first match found in
; the vector.  If no element equals the given string, -1 is returned.

;               CRSF:       A/          <tptr to std string>
;                           B/          <tptr to std vector of std strings>
;                                       PUSHJ       P,@VSSRC
;                                       ...         ; R0: Always return here

;               RTRNS:      A/          Index of first matching string; -1 if none.

; REF1:  Abstract of STRCEQ in STREQU MJFHxx

; CONT:  RSR SIMPLEX
; ATTR:  Nonrecursive, Reentrant
; DEF:  Nothing
; NEEDSA:  Nothing
; NEEDSN:  Nothing
; HDW ENV:  PDP-10
; SFTW ENV:  ITS, CALICO
; CAT:  String
; DESCRPTR:  string, vector, comparison, equality, search
; WARN:  STRINGS OF DIFFERENT BYTE SIZES WILL EQUAL EACH OTHER IF
;                THEIR CONTENTS ARE THE SAME, DUE TO THE USE OF STRCEQ AS
;                OPPOSED TO STREQU.

;...
```

VSSRC fills the bill; so one would choose a vector as the data structure in which to store the set of strings and use VSSRC to find the individuals. As an example of how VSSRC might be called, here is a portion of assembly-language code, with the call to VSSRC in the instruction labeled 'CALL':

```
                .
                .
                .
; Search thru rest of args for mode strings:
MODSCN:         AOBJP       N,OPCH1                 ; Done??
                MOVE        A,(N)                   ; next arg
                AGETYP      0,A                     ; Get its type.
                CAIE        0,STGTYP                ; If it's not a string,
                JRST        WRGTYP                  ;   return error code.

; Find out which mode it is:
                MOVE        B,MDTABL                ; pntr to mode table
CALL:           PUSHJ       P,@VSSRC                ; Search for arg string.
                JUMPGE      A,STGFND                ; Jump if found.

; Mode string unrecognized:
                SCALL       CHERR1,[BADMOD]         ; Generate error.
                JRST        ERROUT
BADMOD:         ASTMAK      [UNRECOGNIZED MODE SPECIFICATION]

; Mode string found:
STGFND:         XCT         INSTAB(A)               ; Execute appropriate insn.
                JRST        MODSCN                  ; And continue scan.

; mode table:
MDTABL:         VCTMAK
                ASTMAK      [READ]
                ASTMAK      [WRITE]
                ASTMAK      [ASCII]
                ASTMAK      [IMAGE]
                ASTMAK      [UNIT]
                ASTMAK      [BUFFERED]
                ASTMAK      [PARSE]
                ASTMAK      [PAGED]
                VCTEND      -1
                .
                .
                .
```

2. MUDDLE

Several major developments centered about the MUDDLE programming language [1,16] the past year. These included: (1) the development of the compiler to a state where it produces moderately efficient code; (2) the design and implementation of a mechanism to allow users to share compiled code; (3) the design and implementation of a mechanism to allow MUDDLE programs to possess core images of greater than 256K (K=1024); (4) the implementation of a new garbage collector; (5) the implementation of a mechanism for communicating with other processes; and (6) the design of a library system.

The MUDDLE compiler (Reeve) currently open-compiles almost all of the arithmetic SUBRs (+, -, *, /, MIN, MAX, MOD, ABS, FIX, FLOAT), many of the predicate SUBRs (G?, L?, G=?, L=?, 1?, 0?, ==?, ASSIGNED?, TYPE?, NOT) and the structure manipulators (NTH, REST, LENGTH, EMPTY?, PUT, PUTREST). For many other SUBRs the compiler generates fast calls into the interpreter rather than going through the somewhat expensive call mediator. In addition, groups of functions can be compiled together, removing the expense of the standard call between the functions in the group. Currently compiled code achieves speedup factors over interpreted code of between 20 and 200 -- the 50 to 100 range being the norm.

Compiled (and assembled) code can easily be made pure and sharable. Its form is such that it can be "loaded" by simply having its disk file included in MUDDLE's memory map (Reeve). If the memory map overflows, any page containing only pure compiled or assembled code can be removed and later added again automatically, in effect giving MUDDLE an expanded address space.

The new garbage collector (Reeve) also uses the memory-mapping mechanism. During a garbage collection, all useful list structure is copied into a temporary address space in another process and compacted to reduce future page faults. When copying is completed, the new pages are simply mapped into the permanent address space and the temporary process is destroyed.

The inter-process communication facility (Ryan) was implemented in MUDDLE to allow programs to send messages to autonomous (daemon) processes. Daemon processes are used, for example, by the Message Facility (see below) and the batch-mode MUDDLE compiler (Ryan). The facility has also been used to rescue MUDDLEs belonging to remote ARPANET users when normal console communication has gone awry and to rescue daemons under development.

In keeping with our goal of promoting the sharing of software, a software library management mechanism for MUDDLE was designed. This mechanism, in addition to performing the housekeeping chores necessary to build and maintain library data

bases, will impose a discipline which will allow the software produced by different programmers to coexist peacefully.

3. Automation of Program Documentation

At present, the CALICO programmer who creates a program is required to write an abstract, preparing it from personal knowledge of the program. This technique works well, though not perfectly, in our environment. The regimen and discipline demanded of the programmer to produce an acceptable abstract is significant. Frequently when programming methodology and program documentation is discussed, a remark that has been voiced by several PTD programmers is, "Convention II and abstracts are the best thing that ever happened to us, but that doesn't mean I must like them." Even with the aid of TECO macros that prompt and aid the construction of a CALICO abstract, it is still an unrewarding task for most PTD programmers to prepare abstracts. Because the documentation task is so unrewarding, we are beginning to formulate plans to automate the process, in order to relieve the programmer of much of the unrewarding tedium currently required in abstract preparation. A program similar to the analysis phase of a compiler could generate automatically much of the information needed in a program abstract. Accordingly, we are beginning to design such a program.

4. Project Reporting System

A first effort was made at providing an automated mechanism for monitoring the status of the various on-going projects and sub-projects in which PTD members are involved (Vezza, Broos). This mechanism was dubbed the "Project Reporting System" and was implemented as an extension of the existing information retrieval system (IRS). With this mechanism we were able to: monitor the status of individual projects; determine the rate of progress of individual projects, by examining their status histories; and survey the overall status of arbitrary subsets of projects, using statistical distributions.

With the Project Reporting System it was found that, while the mechanism itself worked very effectively, the current user interface used to update project status information is inadequate, resulting in a lack of motivation among its users to keep the status information for their projects up to date.

5. Application Programs

The development of a number of application programs was undertaken by students (with some advice and help from the staff) associated with the Programming Technology Division (Cutler, Long, Seriff, To, Yap), a member of the Robotics group of the Automatic Programming Division (Pfister) and two members of the Research Laboratory for Electronics (Pangaro, Raymond).

The development of the application programs provides a good mechanism to test the DMS on real users, albeit friendly ones. A number of times, the development of application programs has provided the stimulus to add a feature that may not have been added without the stimulus. They also serve to provide benchmarks of capability. An example is the program DALI, whose author (Pfister) provided invaluable aid to the developer of the MUDDLE compiler (Reeve). DALI is a sufficiently complex program that it uses a large subset of MUDDLE, and uncompiled (or compiled by an early version of the compiler), runs excruciatingly slow. Because DALI's author desired to see pictures on the CRT move in real time, he had many suggestions about what compiler features might next yield the largest gain in the efficiency of compiled programs. In his enthusiasm to achieve an efficiently compiled DALI, he invariably offered to compile DALI with each new compiler about to be released, thus testing and helping to debug the compiler. The efficiency gained by compiling DALI with a new compiler was also used by others to determine the cost benefit -- in terms of program efficiency -- of recompiling programs. Some of the earliest MUDDLE compilers provided only a factor of two or three speed-up in execution time over interpreted DALI code. This meant that ten milliseconds of CPU time were required to effect movement of a line on the CRT. The current compiler produces code that accomplishes the movement in hundreds of microseconds.

a. DALI

The development of a Display Algorithm Language Interpreter (DALI) is nearing completion (Pfister). It is a special-purpose programming language for the creation and control of changing pictures which exhibit complex static and dynamic interactions among their elements. DALI allows complex organizations of interpolated ('smooth') change, discrete change, and change in the structure of a picture to be generated in a modular way, in the sense that picture elements determine their own behavior and hence manner of change.

In DALI, pictures are composed of elements called picture modules. These are analogous to procedural activations or processes, and contain arbitrary event-driven procedures called daemons. Daemons are run under the control of global scheduling rules based on the functional dependence of daemons on one another. These rules result in smooth inter-daemon (process) communication and cooperation with no implicit or explicit reference to semaphores or other synchronization primitives in user code, while at the same time providing for a high degree of parallelism. Circular inter-daemon functional dependence is possible, and results in iteration or relaxation. The environment structure used is predominantly stack-oriented.

b. Multi-Processor Micro-Computer Simulation System

A simulator was designed (Cutler) providing an environment that allows up to eight independent micro-processors connected in a network to be simulated concurrently. Each processor simulation is to be a separate ITS job, but the design allows common code and data to be shared.

The core image of each simulation job is divided into four parts: (1) storage for internal registers of the simulated micro-computer (accumulators, flags and data); (2) storage for the core space of the simulated computer; (3) storage for the simulator itself; (4) storage for interprocessor communication. This last area can be written as well as read by any of the processors being simulated.

The simulator design is for an assembly-language program that is table-driven and is capable of simulating any micro-computer. Currently the data base is being prepared to simulate the INTEL 8080 micro-computer.

c. World Model and DYNAMO

An implementation of a version of the DYNAMO II language for modeling non-linear feedback systems was completed (Seriff, Long). The implementation consists of a series of MUDDLE functions to provide all but a very few of the features of DYNAMO (although with a completely different syntactic appearance). The PTD implementation is capable of running any simulation that could be run under DYNAMO II (although again the input syntax is different). It is also interfaced to the MUDDLE Console Graphics Package [17] to allow a user to plot the results of the simulations.

The PTD DYNAMO has several features which are not available in DYNAMO II, the most important of which are these:

(1) The ability to stop non-destructively a partly-completed simulation to examine intermediate results or modify parameters. The simulation can then be continued as if no interruption had occurred.

(2) The implementation is completely integrated with MUDDLE, and any MUDDLE statement can be used in the description of a model. Therefore the behavior of the model is not limited to certain predefined modes.

A basic version of the PTD implementation was written and debugged in approximately 2 person-hours. Another 5-7 person-hours was spent adding features (including the plotted output). As a test, Forrester's World Model (from World Dynamics) was converted to PTD format and run. The conversion of the World Model required approximately 2-4 person-hours' effort. As one would expect, the PTD World Model runs slower than the TSO one.

d. Graphing of System Statistics

Since early 1973, when our operating system began paging and swapping memory contents, a daemon process, called the Dragon, has recorded system-performance statistics on disk. During this year a project was begun to display these statistics as graphs on display terminals (Yap). The operating system was modified to record in system memory additional performance measures, and the Dragon program was modified to obtain all the measures from the system and record them on disk in MUDDLE-readable format (Brescia). The Dragon now records all logging in and out of humans and daemons, and, at five-minute intervals: the number of human users; the number of running jobs; the total size of virtual memory and the total number of swapped-out pages, for running jobs and for all jobs; the amount of unused core; and accumulated idle time, execution time for all jobs, and requests for swapped-out pages. The capability to easily graph the system's behavior under real load conditions will provided some insight that we hope will help us understand how such systems behave and lead to improved system performance.

e. Computer Simulation of a Bifurcating Axon

The PTD system is being used by two members of the Research Laboratory for Electronics (S. A. Raymond, Professor of Biology, and P. Pangaro) to model and simulate the way synaptic activity affects nerve membranes. The computer is being used to model and simulate a hypothesis concerning the dependence of nerve threshold on impulse discharge patterns imposed upon it. The hypothesis is difficult to explore biologically because of the volume of recordings from axon endings required. The PTD system is being used to gain insight into what physiological experiments on single nerve fibers would prove fruitful in testing the hypothesis. In the computer, simulated sections of nerve membrane, each having an independently-specified dependence of threshold on activity, are coupled together to form a bifurcating tree. The simulation output is presented in graphical and animated forms on the PTD Evans and Sutherland display. The simulation is quite rapid (compared to time required for a physiological experiment) and the output is easily digested. The information displayed consists of input records, threshold curves for any branch in the tree, post-synaptic cells that connect to the axons, long-term output graphs, and two copies of the tree, one showing invadability, the other actual invasion by an impulse into the tree.

f. Teleconferencing

An experimental system was implemented (Lebling, Thompson) this year that allows several users to interact through the computer as a group in conference. Each participant interacts directly with a program that sends messages to and receives them from other incarnations of the program, corresponding to other participants. The program processes messages in real time, decides which ones should be shown to the user, and outputs those to the console in a user-controllable format. Messages are

sent among users' programs by being put into a shared in-core data base. All interactions can be recorded, so that the "proceedings" are available for later inspection. To test the system, a game was devised wherein conferees attempt to find one another in a simulated maze of hallways, each user seeing broadcast messages and a schematic view of the immediate surroundings and other visible players. The high degree of interaction and communication necessary for this game was achieved.

g. An Interactive Statistics Package for the Social Sciences

Social science oriented statistical packages are often used naively. This is partly due to a lack of real user-system interaction, partly due to deficiencies in design of the systems, and partly due to errors and ignorance on the part of the user. After considering ways in which these problems can be alleviated, a system called ISP (Interactive Statistics Package) was designed to eliminate the problems and was implemented in the CALICO environment (Lebling) [18].

ISP contains a unitary and uniform command structure and data storage and retrieval system. Its operations are user-expandable. ISP contains a matrix and statistics oriented desk calculator facility and an interactive graphics capability. Emphasis is placed in four areas: documentation, expandability, ease of interaction, and system cognizance of the assumptions under which its operations are taking place.

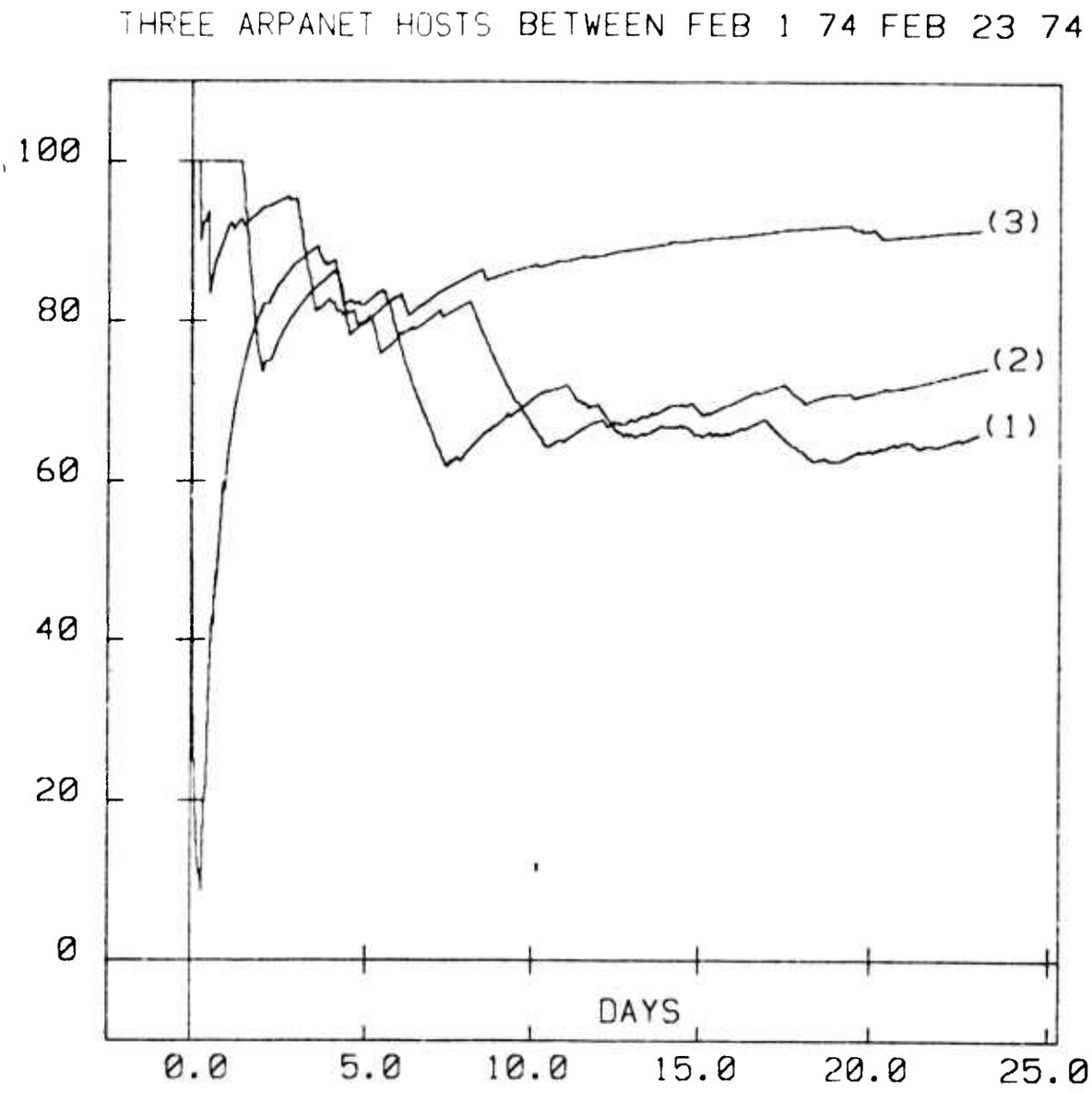
C. NETWORKS

The PTD continued this year to focus on tools that allow the ARPA computer network and its resources to appear to be an integral part of the DMS environment, as elaborated below. As a serving host (MIT-DMS), we (Chan) implemented for the "new" Telnet protocol a server program that is compatible with the "old" protocol.

1. Datacomputer Experiments

We have continued our experiments with the Datacomputer, an ARPANET service run by Computer Corporation of America providing storage and retrieval capabilities in a mass store -- potentially 10**12 bits. The SURVEY program was modified to send its ARPANET host availability data only once a day (normally at midnight) to minimize interference with process- and user-initiated retrieval requests. Provision for surveying arbitrary socket addresses was added.

A specialized interactive program, SURRET (Bengelloun, Westcott), used as an interface between MUDDLE and the Datacomputer's Datalanguage to request retrieval of SURVEY data from the SURVEY data base at the Datacomputer, was integrated with the MUDDLE Console Graphics program (Ryan) to provide for display of the SURVEY data in graphic form on an IMLAC, on a storage tube display (ARDS), and



PERCENTAGE OF SURVEYS LOGGER AVAILABLE
SINCE START OF TIME PERIOD

FIGURE 2.

on Xerox Graphic Printer output. A typical graph is shown in Figure 2.

2. PTD Message Facility

A PTD Message Facility was designed. Its structure was determined by several basic design goals (Haverty, Black, Vezza, Sybalsky, Chan, Bhushan). First, the system is meant to be easily usable by both humans and processes. Second, a user must not be required to wait needlessly while the message transmission and other processing are accomplished; only processing to support interactive facilities, such as editing the text of a message, is done while the user waits. Third, the system must be capable of being 'programmed' to a large extent. Users must be able to set up their own particular entries in the data base to control how the message system behaves for them. This is especially valuable for experimentation purposes.

The Message Facility design logically divides the system into two functional parts. The first part is composed of two elements: the communication daemon COMSYS (Haverty) [33] and the ARPANET interface NETOUT (Chan). They handle the actual transmission and delivery of messages. In addition, COMSYS controls the scheduling of other processing that can be performed on or at the direction of the message.

The second, interactive, part of the facility is to be composed of three integrated elements: the composer COMPOS (Black), the READER (Sybalsky), and the retrieval system IRS (Broos). These systems are to be used only by human users; processes presumably do not need the facilities provided, or, if they do, a process can either implement the required function itself, or communicate directly with the daemon, as the interactive systems sometimes do, to accomplish its ends.

a. Communication Daemon and Network Interface

The communication daemon COMSYS is to handle virtually all of the non-interactive processing involved in transmitting a message. It maintains several data bases, which are used to keep track of messages as they are processed, and to hold information to tailor the processing done to the specifications of the sender and receiver, as appropriate. Simply modelled, the daemon is a process with several inputs and several possible outputs, with a large dynamic data base. Messages and requests for information enter the daemon via one of its inputs, and messages and data leave the daemon over one or more of its outputs. For example, one input path is via a publicized file name. If such a file is written, containing the specifications of a message to be sent, the daemon will read the file and process the message. The message will possibly be placed in another file, in the recipient's directory, if that is the output path selected.

COMSYS will possess the framework to envoke additional processes, called

'author' or 'recipient' processing, as commanded by data bases, author or recipient actions via a console, and a flag in or attached to the message. These additional processes can perform arbitrary tasks, such as inhibiting sending of the message until all authors agree to send, or redirecting an incoming message to another or for that matter several recipients.

The daemon additionally will handle requests to perform several types of standard processing on messages, such as copying it to the printer or a disk file, inserting it into an IRS data base, converting a group addressee into individual addressees, etc.

The network interface NETOUT will handle all protocols concerned with the current ARPANET implementations. It will provide isolation from ARPANET dependencies to permit the communication facility to be developed without any restrictions imposed by the present network protocols, since one goal of the project is to determine what, if anything, is needed in a protocol to support such a facility. Additionally, the network interface handles problems related to multi-computer systems, such as what to do when the addressee's site computer is currently down.

b. COMPOS, READER and IRS

The three interactive elements are to be an integrated unit so the user can easily use functions of any element without inconvenience. This would permit, for example, a user to read his or her mail, retrieve a message to which a reply has just been received, and refer to it in composing a new message that is the reply to the one just received.

The composer will provide an environment for writing and editing a message (including the use of intelligent terminals for editing), and specifying how the message is to be handled and transmitted by the daemon, querying the status of messages previously sent, retrieving information about messages sent or received for use in composing a new message. The composer system is intended to handle virtually all interactive processing involved with an author's composition and control of her or his messages. Additionally, it will provide commands to edit the COMSYS daemon's data bases.

The current composer design allows the user to edit any of the message's fields at any time up until the message is transmitted. The design specification currently admits the following fields: action addressees, carbon-copy addressees, blind carbon addressees, from (sender), authors, subject, message (text), files for output, references (list of reference numbers), reply-to (message number this message is response to), keywords.

The READER is to be an interactive program through which human users

examine and act on the messages they have received. It will provide indexing facilities for categorizing messages according to their urgency, author, etc., and commands to specify how to dispose of the messages. The daemon provides several basic processing capabilities which the READER can trigger, such as printer output, file output, etc., which will be provided as READER commands. Additionally the READER system will keep track of the status of each message, so that the user can inquire at any time to find out what messages are still pending, need replies, etc.

The Information Retrieval System, IRS, is a generalized data base interrogation system, that was developed separately for use in subroutine library systems. It will be interfaced to the PTD Message Facility so that messages can be entered into an inverted data base, and commands used to extract messages by keyword, author, recipient, carbon copies reference, etc. A user will be able to direct that all received mail be placed in such a data base automatically, so that old messages can, at any time, be examined, sorted according to author, etc.

D. OTHER

1. Multi-screen Console Program

A new console-control program for the IMLAC programmable display terminals was implemented (Lebling) and is available at the user's option. Its main feature is the capability to maintain up to eight "virtual screens", that is, display buffers for text and/or graphics. Each buffer can be visible or invisible and has settable location on the display screen. A user can switch between buffers by making first one and then the other the only visible one. The contents and parameters of all screens are accessible and incrementally changeable (including graphical data) from both the console keyboard and the PDP-10. There is no set limit on the number of characters in each buffer, only on the sum for all buffers. If a buffer has more lines in it than will fit on the screen, only a contiguous subset is displayed, the subset window moving by scrolling action. An ability to change command functions or character fonts, by overlay loading different subprograms from the PDP-10, is designed but not implemented.

The future use of this console program will be in associating multiple display buffers with a user's multiple tasks. A user will be able to carry on independent tasks concurrently, with each task's console I/O going in its own buffer, which may be viewed only when needed. For example, the delivery of a message by the Message Facility can be to a suitable buffer, without disrupting the appearance of editing operations or abstract-checker output in other buffers. Protocols for communication between the operating system and consoles of graphical and multi-screen data are being designed (Brescia, Lebling).

2. Operating System

During the years that the PTD PDP-10 had no paging hardware, the operating system ITS, obtained originally from the M.I.T. Artificial Intelligence Laboratory, diverged from the AI and Mathlab version through upgrading of both. Because the PTD, Mathlab and AI hardware bases are now sufficiently similar, the merging of ITS versions was completed this year (Brescia, Cutler of PTD; Stallman, Greenblatt, Knight of AI; and Jarvis of Mathlab). Thus only one source file is maintained and upgraded, with assembly-time switches accounting for minor hardware differences. The commonality also allows sharing utility programs (for example, DDT, TECO, MIDASI) on all systems without modification.

Part of the operating system is the Network Control Program, which interfaces user-level programs to the ARPA computer network. This year the NCP was upgraded (Brescia) to give user-level programs all available data about the up/down status of remote hosts, and to supply our status to remote hosts prior to service interruptions. The output speed of network traffic was increased through a redesign of buffering in the NCP, allowing user-level programs to output before buffer allocation has been received from the destination host.

REFERENCES

Note: The form XXX.nn.nn denotes a PTD document.

1. Project MAC Progress Report X, 1972-73, Section III.

2. Seriff, Marc, The BATCH Processing Facility, SYS.14.12 (unpublished).

3. Seriff, Marc, The TAILOR Facility, SYS.14.11 (unpublished).

4. Seriff, Marc, CALICO Conditional Commands, SYS.14.07 (unpublished).

5. Galley, S. W., Managing Inferior Jobs in CALICO, SYS.14.13 (unpublished).

6. Haverty, John F., The CALICO Message Facility, SYS.14.14 (unpublished).

7. Seriff, Marc, How to Write Programs for the CALICO Environment, SYS.14.04 (unpublished).

8. Reeve, Chris, Marty Draper, D. E. Burmaster, and J. C. R. Licklider, Convention II: Standards for Listings. Listing Abstracts, GA.01.09.02, August 1971.

9. Broos, Michael, IRS -- Information Retrieval System, SYS.14.01, January 1973.

10. Broos, Michael, IRS -- Generating and Maintaining Data Bases, SYS.14.02 (unpublished).

11. Licklider, J. C. R., and Karolyn Martin, Convention II: Overview of Glossaries, GA.01.02.00, January 1972.

12. Licklider, J. C. R., and Karolyn Martin, Convention II: Glossary of Standard Notation, GA.01.02.01, August 1971.

13. Licklider, J. C. R., and Karolyn Martin, Convention II: Glossary of Standard Terms, GA.01.02.02, January 1972.

14. Licklider, J. C. R., and Karolyn Martin, Convention II: Glossary of Abbreviations and Expansions, GA.01.02.03, January 1972.

15. Martin, Karolyn, Convention II: Standards for Naming Files, GA.01.05, August 1971.

16. Pfister, Gregory F., A MUDDLE Primer, SYS.11.01, December 1973.

17. Ryan, Neal, MUDDLE Console Graphics, SYS.11.11, October 1973.

18. Lebling, P. David, Interactive Statistics Package for the Social Sciences, S. B. and S. M. Thesis, MIT, Department of Political Science, August 1973.

Publications

1. Licklider, J. C. R., Psychological and Technological Dynamics of Information Science, NATO Advanced Study Institute, Aberystwyth, Wales, August 1973.

2. Licklider, J. C. R., "Potential of Networking for Research and Education," Networks for Research and Education, Martin Grenbeger, Julius Aronofsky, James McKenney, and William F. Massy, editors, The MIT Press, Cambridge, Mass., 1974, Chapter 5.

3. Licklider, J. C. R., "Communication and Computers," Communication, Language, and Meaning, George A. Miller, editor, Basic Books Inc., New York, 1973, pp 196-207.

4. Weissberg, R. W., "A Tool for Interactive Graphical Simulation of a Hospital Emergency Room", Proc. IEEE Conf. on Systems, Man, and Cybernetics, Nov. 1973.

Theses Completed

1. Daniels, Bruce K., A Compiler for MUDDLE: Optimization with Partial Knowledge, S.M. Thesis, MIT, Department of Electrical Engineering, August 1973.

2. Farrell, Gerald J., A Graphics System for MUDDLE Programming, S.M. Thesis, MIT, Department of Electrical Engineering, August 1973.

3. Lebling, P. David, Interactive Statistics Package for the Social Sciences, S.B. and S.M. Thesis, MIT, Department of Political Science, August 1973.

4. Metcalfe, Robert M., Packet Communications, Ph.D. Thesis, Harvard University, Division of Engineering and Applied Physics, June 1973 (MAC TR-114).

5. Weissberg, Richard W., TIGERS: A Tool for Interactive Graphical Emergency Room Simulation, S.B. and S.M. Thesis, MIT, Department of Electrical Engineering, August 1973.

COMPUTER SYSTEMS RESEARCH

Academic Staff

Prof. F. J. Corbató
Prof. J. H. Saltzer
Prof. M. D. Schroeder
Prof. E. Wada
D. D. Clark

DSR Staff

R. K. Kanodia
R. J. Mabee
E. W. Meyer, Jr.
M. A. Padlipsky
K. T. Pogran
E. L. Thomas
D. M. Wells

Graduate Students

A. J. Benjamin
R. G. Bratt
R. J. Feiertag
R. M. Frankston
B. S. Greenberg
D. H. Hunt
D. P. Reed
L. J. Scheffler
J. A. Stern
V. L. Voydock

Undergraduate Students

R. H. Gumpertz
J. B. Williams
D. H. Moon

Support Staff

O. D. Carey
S. M. Clark
D. E. Cohen
S. D. Grant
D. L. Jones
M. F. Webber

COMPUTER SYSTEMS RESEARCH

A. INTRODUCTION

The current research activities of the Computer Systems Research Division can roughly be described as trying to discover and implement the minimum mechanism essential to support a full-scale computer utility system. Its activities are pragmatic, which means that most ideas are subjected to practical implementations as part of their development. The division uses the Multics development facilities to test special modified versions of the system.

The largest portion of current work is inspired by the need to certify the correctness of privacy-achieving and other information-isolating mechanisms in a shared-user system. On the basis that certification of correctness should be easier if the mechanism being certified is simpler, work is proceeding to first identify and then minimize the complexity of the central protection kernel of Multics.

The second major area of interest is simplifying and better understanding the attachment of the ARPANET to Multics. Because the ARPANET involves an element of distributed computations connected by communication lines, it relates directly to a longer-range interest in the future of computer-utility systems in an era when logic and memory costs are predicted to make personal computers as commonplace as today's hand-held calculators. The central utility is still needed for communication, sharing of information, and handling peak loads, but its interfaces and possibly its functions must certainly be different from those of today's systems.

Another activity reported here is called "technology transfer", a collection of efforts to communicate with industry and users the results of previous research, particularly the development of the Multics system.

This report is organized in four sections. The first three describe the major work: certification/simplification projects, ARPANET-related activities, and the technology transfer activities. The fourth reports other miscellaneous activities of the division.

B. CERTIFICATION OF COMPUTER SYSTEMS

This year the Computer Systems Research Division began a new research project with the goal of making possible the certification that the data security facilities in a large-scale, multiuser computer system have been correctly implemented. This effort, the primary research activity of the Computer Systems Research Division for approximately three years, has the goal of producing computer systems which guarantee prevention of unauthorized release, modification, and denial of use of the information they contain. The need for such certification arises when a single system provides computation and information storage service to a community of users. As the economic and functional advantages of such shared systems have been recognized, so has the need to include facilities for controlling the access of the various users to the contained information. Without these facilities, sensitive information can be handled only if the user community is carefully restricted to be a highly homogeneous group. Many systems now include protection mechanisms for enforcing intricate, externally specified policies on information access. The presence of such mechanisms, however, is not enough. Users, whether they be individuals or private or governmental organizations, must have confidence in the integrity of the protection mechanisms before they can entrust sensitive data to a system. The system must be certified to implement without failure the desired policies for controlling access to the contained information.

There are three ways in which the security of information stored in a computer system can be violated:

1. Unauthorized release: an unauthorized person is able to read, and take advantage of, information stored in the computer. Concern sometimes extends to "traffic analysis", in which the person observes only the patterns of use of information and from those patterns can infer some content;

2. Unauthorized modification: an unauthorized person is able to make unexpected changes to stored information;

3. Unauthorized denial of use: an unauthorized person can prevent legitimate access or modification, even though he or she may not be able to access or modify the information, for example by causing a system "crash".

Complicating things in a shared computer is the fact that the unauthorized person with respect to a specific act may be an otherwise legitimate user of the system.

In practice, producing a system that actually does prevent all such unauthorized activities has proved extremely difficult. Sophisticated users of most currently available systems are probably aware of at least one way to "crash" the system. Penetration exercises involving a large number of different systems have shown that, in all systems confronted, a wily user can construct a program that can obtain unauthorized access to information stored within the system.

The primary reason for these failures is the presence of design and implementation flaws that provide paths by which the access constraints supposedly enforced by the system can be circumvented. Underlying this cause are two interacting difficulties. The first is that preventing all unauthorized acts is a negative kind of requirement. It is intrinsically quite hard to prove that this requirement has actually been achieved, for one must demonstrate that no means for violating data security exist. The second is the well-known tendency for the operating systems of shared, general-purpose computers to be extraordinarily complex, large in size, difficult to maintain, and awkardly organized. This tendency interacts badly with the need to prove non-existence of paths for violating data security, by providing a very complex environment in which to attempt such a proof.

There seem to be several reasons for the tendency toward complexity, such as:

1. attempts to stretch the functional capabilities of the system as far as possible;

2. working in a hardware environment that was determined before software requirements were fully understood;

3. attempts to squeeze the system to its absolute limit of performance;

4. attempts, because of the high cost of system development, to get the system running in the absolutely shortest time possible.

Of these four, probably the last two are the strongest contributors to overall complexity, since both encourage shortcuts to be taken and modularity to be violated against the better judgement of the system designer.

The certification of a system means that someone has signed-off on a statement of adequacy. By signing, the certifier states that the security provided is adequate to the intended application. He also assumes the responsibility for failures. A system is certifiable if the certifier can be convinced to sign. With currently

In practice, producing a system that actually does prevent all such unauthorized activities has proved extremely difficult. Sophisticated users of most currently available systems are probably aware of at least one way to "crash" the system. Penetration exercises involving a large number of different systems have shown that, in all systems confronted, a wily user can construct a program that can obtain unauthorized access to information stored within the system.

The primary reason for these failures is the presence of design and implementation flaws that provide paths by which the access constraints supposedly enforced by the system can be circumvented. Underlying this cause are two interacting difficulties. The first is that preventing all unauthorized acts is a negative kind of requirement. It is intrinsically quite hard to prove that this requirement has actually been achieved, for one must demonstrate that no means for violating data security exist. The second is the well-known tendency for the operating systems of shared, general-purpose computers to be extraordinarily complex, large in size, difficult to maintain, and awkardly organized. This tendency interacts badly with the need to prove non-existence of paths for violating data security, by providing a very complex environment in which to attempt such a proof.

There seem to be several reasons for the tendency toward complexity, such as:

1. attempts to stretch the functional capabilities of the system as far as possible;

2. working in a hardware environment that was determined before software requirements were fully understood;

3. attempts to squeeze the system to its absolute limit of performance;

4. attempts, because of the high cost of system development, to get the system running in the absolutely shortest time possible.

Of these four, probably the last two are the strongest contributors to overall complexity, since both encourage shortcuts to be taken and modularity to be violated against the better judgement of the system designer.

The certification of a system means that someone has signed-off on a statement of adequacy. By signing, the certifier states that the security provided is adequate to the intended application. He also assumes the responsibility for failures. A system is certifiable if the certifier can be convinced to sign. With currently

available commercial systems, there is no way for a potential certifier even to start developing the confidence in a system prerequisite to signing. Most are of a size and complexity to preclude even reading all of the code, much less comprehending the entire mass in detail.

The Computer Systems Research Division research effort is aimed directly at the size and complexity. The overall plan is to evolve an existing, commercial, multiuser computer system -- Multics -- which is easily modifiable and which has advanced protection mechanisms, into a prototype operating system with all the essential features of the present Multics system, but with a small and simple central core that is susceptible to certification through line-by-line review by an expert. The goal is a system sufficiently small, well-structured and easy to understand that a certifier can read it all, understand the reason for every line of code, and develop confidence that its correct operation is adequate for most applications.

1. Method of Attack

The problem of constructing a certifiably secure system recently has attracted considerable interest and is being attacked with a variety of different strategies by many research groups in addition to the Computer Systems Research Division {Saltzer: "Ongoing Research and Development on Information Protection", ACM Operating Systems Review, July, 1974}. It is generally recognized that the key to the ultimate solution is methodical design and construction techniques which systematically exclude flaws that can be exploited to produce security violations. Many imagine ultimately being able to construct a formal specification for a system, prove desired security (and other) properties about the specification, and then, by essentially mechanical steps, construct a matching operational system. To this end, many research groups are conducting investigations into methods of proving assertions about programs and program-like specifications, methods for formally describing properties like security, and techniques of top-down program construction by successive refinement of descriptions of algorithms and data structures. On the other end of the spectrum, several groups are engaged in finding and cataloguing flaws in existing systems with the aim of convincing skeptics that the problem is real and of understanding the sort of flaws that can be exploited. Somewhere between these two extremes are several groups, including our own, looking for the simplest possible structures with which to securely implement the full set of functions that seem desirable in a multiuser, general-purpose computer system. An understanding of simpler ways to organize such systems will contribute to the development of the ultimately required mechanical construction techniques. But of more immediate importance, it will also allow us to build less complex systems to do the same job, thus providing a less complex environment in

which to establish the absence of security flaws.

Given that problems of structure are to be attacked, two approaches are possible. The first is to wipe the slate clean and design a new system from scratch. The accumulated knowledge of past successes and failures could be brought to bear in an attempt to produce a new design that is well-organized, simple, and concise. By starting from scratch a great deal of freedom is gained to organize the entire system and its specifications to facilitate demonstration of a lack of flaws. The second approach is to modify in an evolutionary way an existing system so as to simplify its organization to the point where the absence of security flaws can be demonstrated. Our choice of this second approach requires some comment.

The first approach, while appealing, has the defect that there seems to be no way to release the designer of a new system from the pressures toward complexity which were mentioned earlier, especially the pressure to get a system operational as soon as possible because of the development expense. Any attempt to mitigate this pressure by stopping short of producing an operational system seems to have two problems. First, with the current state of understanding of computer systems, it is hard to have confidence that the full implications of a system structure are understood without complete implementation. Second, if an operational system is not the goal, it is very easy to leave out many of the complexity-producing convenience features that users demand of a production system. A structure which gracefully supports a toy system may be badly strained under the load of conflicting features required in its real descendant. Put another way, design and implementaton flaws representing potential security violations tend not to be a problem in toy systems.

To avoid these difficulties, the Computer Systems Research Division has adopted the approach of evolving an existing operating system to simplify its structure and reduce its bulk. The greater danger of this approach is that the system chosen for evolution will prove so resistant to graceful alteration that no evolution of structure is possible, short of starting over. Thus, it is extremely important to pick a suitable subject. We are using the Multics system, previously developed by the Computer Systems Research Division of Project MAC. Multics is better organized than most systems for evolution and modification, because it is relatively modular, is largely written in PL/I, and was originally constructed with evolution as a primary objective. Also, Multics has been developed from the ground up to protect the information it contains from unauthorized access. It already includes protection mechanisms as advanced as any available, including special hardware features such as protection rings. Thus, the system both exhibits a set of protection features that would be interesting to certify and provides protection features that will make the job of certification

easier. Finally, because Multics is a commercially available product and new ideas developed in the course of this research should be relatively easy to retrofit to the standard system, the result, if successful, can be easily exported in a directly useful way.

Although the original design of Multics was very methodical, and the system is, if anything, already less complex in organization than most contemporary computer operating systems with similar functional goals, potentially it could be supported with mechanisms that are much simpler yet. The intense pressure of initial implementation did not permit time for contemplation and development of simpler supporting structures. The basic premise of this research is that one wave of simplification applied to the central core of the system will produce a badly needed example of a structure that is significantly easier to understand.

2. The Security Kernel

The total volume of software in a system like Multics is enormous. In addition to the supervisor and other system-provided software such as subroutine libraries, compilers, and specialized applications packages, a large community of active users produces many programs of its own. If the security of information in such a system depended upon the correctness of the entire collection, then our task clearly would be hopeless. To make progress, the system must be arranged so that the security of each user's data depends only on the correct operation of some subset of all the software contained in the system -- the smaller this subset the better. Indeed, the programs produced and executed by other users must be able to be excluded from the subset affecting any one user, for the potential malicious activity of another user is the presumed threat.

The overall structure used by Multics to control user access to stored information is to provide each user computation with its own process and address space. A process has no ability to access the address space of another. In the address space of every process are the procedures and data of the Multics supervisor. Among other things, the supervisor manages hardware resources and creates processes and their address spaces. The ring protection mechanism of the hardware processors is exploited to restrict the access to the supervisor of the user code executing in a process.* Supervisor components cannot be directly referenced; only specially designated entry points may be called. Once called, the supervisor procedures have direct access to other supervisor procedures and data. The supervisor manages all on-line storage, and can be requested to add a segment of on-line data or procedure to a process address space where it may be referenced by the

---

*The rings actually provide a process with eight different protection states rather than the two implied here, but that level of detail is unnecessary for this discussion.

user code of that process. Such a request will be granted only if the supervisor has been informed that the user controlling the process is authorized to access the segment.

The security of the data stored in the system certainly depends upon the correctness of the supervisor, for it is the primary path by which one user's computation can influence another's data or computation, legitimately or otherwise. For example, an attempt by one user to gain unauthorized access to data, or to deny access to an authorized user, might be made by invoking a supervisor entry with an unexpected pattern of arguments, perhaps causing the supervisor to mistakenly cause problems for another user. Because of the size and complexity of the supervisor, the chances of a clever attacker ultimately succeeding in uncovering an exploitable flaw are good.

The supervisor is a software mechanism that is common to all users of Multics. A mechanism is common to a group of users if it can be used by one to influence the data or computation of another in the group, legitimately or otherwise. At the heart of every common mechanism must be some group of data items whose value one user's computation can influence and another's can notice. The influence and notice may be very direct -- one writes into a data item and another reads it -- or quite indirect -- the invocation of a procedure by one somehow alters its internal state so that the outcome of a later invocation by another is affected. Common mechanisms are required to implement any explicit or implicit communication among a set of users. If no such communication or coordination is involved, however, then a common mechanism is not required to implement such a function. It is precisely the existence of common mechanisms that allow one user the possibility of exerting unauthorized influence over the computations or data of another. Malicious users must exploit flaws in common mechanisms to work their will. To prevent such malicious activity it is the common mechanisms that must be certified to contain no exploitable flaws, and once certified must be protected against tampering.

The Multics supervisor is bigger and more complex than it needs to be. This is partly the result of the presence in the supervisor of procedures and data-providing functions that need not be implemented with common mechanisms because they include no element of communication or coordination. Yet, by being part of the supervisor, with the attendant access privileges, effectively they are part of the common mechanism. Flaws they contain may be exploited as illicit access paths.

A primary strategy of the research project is to evolve the current supervisor into a security kernel that contains only functions required to be implemented as common mechanisms, removing all other functions to execute as user programs in each process, where they cannot be exploited as illicit interuser access paths. The security kernel produced should be the least amount of common mechanism necessary to implement the patterns of information sharing, interprocess communication, and physical resource multiplexing that are required in the system. As the common mechanism is made small its structure will be simplified also, the goal being a smallness and simplicity sufficient to permit certification of the resulting kernel. It appears feasible to extract a kernel with 4,000 to 8,000 lines of source code from the present supervisor of approximately 50,000 lines of source code.*

Does a certified security kernel, as just defined, really produce a system guaranteed to prevent all unauthorized attempts to release, modify or deny access to contained data? The answer, unfortunately, is no. Any non-security kernel software which executes in a process that has access to some data has the potential to compromise that data. These programs can be grouped in four categories. First there are the system-provided programs -- the library subroutines, compilers, and application packages available in most systems plus all the programs thrown out of the old supervisor in the minimization of the new security kernel. These system-provided programs are not common mechanisms, even though in Multics all processes share the same non-writeable segments of code that embody their algorithm, for a private copy of the alterable part of these procedures, the variable data, is provided for each process. Because they are private mechanisms, no interuser interaction can occur through them. They may still contain errors if they are not certified, but these errors can be triggered only by the actions of the process that they might damage as a result of the triggering. By presuming that the system programmers who constructed them are non-malicious and did not willfully plant "trojan horses", it seems justified to assume that the mistakes caused by these system-provided procedures will decrease in time as all normally used functions are exercised, and that the security threat posed

---

*It is expected that almost all of the source code will be in the PL/I language. Using PL/I to generate the kernel seems to require that the PL/I compiler be certified, as well as the kernel, a troubling thought since the compiler itself is a 25,000 line PL/I program. In the case of the compiler, however, certification may be less of a problem than for the kernel. The kernel needs to work correctly for all possible input; the compiler need compile correctly only for the specific programs of the kernel -- not all possible programs. Thus, the compiler's effect on the kernel can be certified by comparing the source code for each kernel module with the compiler-produced object code, a task much simpler than certifying the compiler correct for all possible source programs.

by a potential random error causing undesired release or modification or a users' data is acceptable for most applications. Unlike the software mechanisms of the security kernel, these are not susceptible to willful exploitation by other users. In any case, a user unsatisfied with their trustworthiness may, in Multics, choose not to use them, substituting his own procedures.

The last comment suggests the second category of procedures executing in the user environment of a process -- procedures constructed by that user. Any security threat posed by errors in these is the user's own problem. The only possible help would be providing tools to aid the user in certifying his own programs.

The third category, possible in Multics, is procedures borrowed from other users. These are a real danger to the security of the borrower's data. Because they will execute with all the access authority of the borrower's own procedures, they can contain "trojan horse" code maliciously constructed to cause a security violation.* A user should only borrow procedures from another when the borrower has reason to trust the lender. The inclusion of security kernel facilities to support user-constructed protected subsystems provides a tool to reduce the potential damage such a borrowed trojan horse can do, but a user-initiated certification of the borrowed program is the only complete protection against this threat.

The fourth category is common mechanisms set up among a group of users by their mutual consent to implement some function involving interuser communication or coordination. Such a mechanism makes the group susceptible to undesired interaction in the same way that an uncertified supervisor does for the whole user community. If a user agrees to become party to such a common mechanism, then he must satisfy himself of its trustworthiness.

In considering these four categories, it is apparent that none is so important to system security as the common mechanism of the security kernel, for every user of the system is forced to rely upon it. Because it appears to have maximum leverage on the security problem, the Computer Systems Research Division is concentrating on abstracting the security kernel and on simplifying its internal structure. A Multics with a certified security kernel would provide a usefully greater level of security than the present system provides.

---

*Note that this is a special case of a common mechanism. The data items whose value the lender can cause to change and thereby influence the data of the borrower is the code of the lent procedure itself. Even if the procedure is non-writable when lent, it was written by the lender when constructed.

3. Specifics

The effort by the Computer Systems Research Division to produce a security kernel for Multics can be broken into four interrelated categories of activity: reviewing, removing, simplifying and partitioning. This section describes these categories, giving examples of each. The next section provides a complete list of the work performed during this report period.

The review category covers all efforts to understand better the specific problems of the current Multics supervisor. In addition to trying to understand the reasons for the size and complexity of the current supervisor, an effort is being made to identify and correct existing security flaws. A list of all known Multics security flaws is maintained. Each flaw reported is analyzed to determine how it happened, how it can be fixed, and how similar flaws can be avoided in the future. Several audits have been made to uncover suspected new flaws. One highly successful search for new flaws was undertaken as a result of a suggestion by Richard Bisbey II, at the Information Sciences Institute of the University of Southern California, who has been trying to abstract from many system penetration exercises some general patterns that lead to security flaws in different systems. He reported that multiple references by supervisor code to user-provided arguments can be exploited in many systems to cause supervisor malfunction, and pointed out one example he had uncovered in Multics. The problem is illustrated by the following procedure which might be used to implement the segment deletion function in a system like Multics:

```
delete_seg:  procedure(name, code);                          1
             call verify_permission(name, "delete", code);   2
             if code = proper_access                         3
                then call delete(name, code);                4
             return;                                         5
end;                                                         6
```

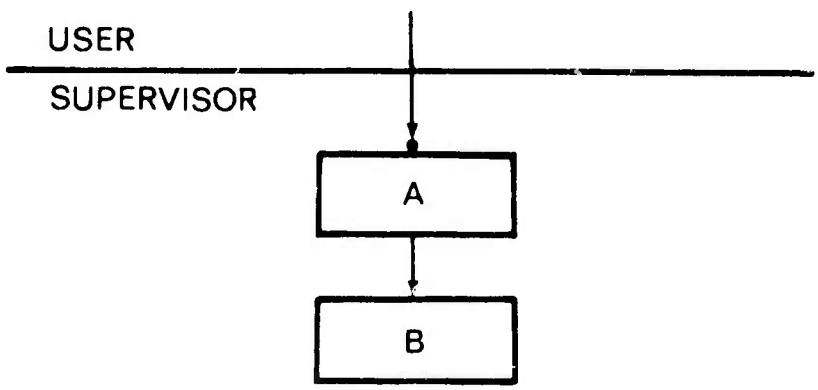
Assume that "name" and "code" are user-provided arguments passed by address. The first identifies the segment to be deleted from the file system while the second provides a place for a return error code. The call to "verify_permission" verifies that the user controlling this process has permission to destroy the named segment. If the "code" returned indicates that the user has delete permission for the segment, then the "delete" procedure is called to perform the requested act. Now imagine that the user contrives to change the value of "code" between lines 2 and 3. Then clearly he

can cause the deletion of a segment for which he has no delete permission. The same problem exists with the "name" argument. In almost all systems, including Multics, the user can cause such a carefully timed change in the value of an argument in a methodical way.*

As a result of Bisbey's suggestion, an audit of the 170 entries to the Multics supervisor was made, looking for this pattern. The audit uncovered 50 entries that made multiple references to arguments. Of these, 8 clearly were exploitable security flaws, 34 looked safe, and 8 were questionable. The problem can be systematically avoided by requiring all supervisor entries to copy their arguments before using them.

So far, all of the flaws uncovered by the review activities are isolated and easily repaired. No major design flaws have been found.

The second category of activity is removing from the supervisor those mechanisms not implementing functions of information sharing, interprocess communication, or physical resource management, i.e., those functions not required to be implemented as common mechanisms. In many cases removal involves undoing a pattern caused by a performance characteristic of the Multics implementation for the Honeywell 645 computers. For that older machine, protection rings were simulated in software and cross-ring calls were quite expensive. Thus, a call that went from a user protection environment to the supervisor cost much more than a call which did not change protection environments. The result was an effective pressure to include many functions in the supervisor that did not need to be implemented as part of a common mechanism. The reason for this pressure can be seen from the following figure:



---

*In fact, it is particularly easy on the Multics Honeywell 6180 processor, because of the existence of a mode of addressing where the value of an indirect address stored in memory will increment automatically wth each use. Placing an auto-incrementing indirect address in an argument list of a supervisor call can generate the desired change of argument value at just the right moment.

A and B are procedure modules in the supervisor. Imagine that a single invocation of A (by a user procedure) can result in a flurry of calls from A to B. Then there is a clear performance cost in moving the user/supervisor boundary to between A and B, even if only B need be part of the protected, common supervisor.

The new hardware base for Multics, the Honeywell 6180, implements the protection rings in hardware. One result is that calls from one ring to another now cost no more than calls inside a ring. Thus, the performance penalty associated with supervisor calls has been removed, and many modules included in the supervisor for performance reasons rather than protection reasons now can be removed.*

The actual removal activities are much more complex than suggested by the example of the previous paragraph. In most cases the common and private parts of a facility are not so neatly packaged in separate procedures, but are intricately intertwined in the same procedures and data bases. Insight and ingenuity are required to separate the private and common parts of a mechanism, leaving a reasonable interface. Also, supervisor procedures execute in a slightly different environment than other procedures. Code written for the supervisor environment often depends upon the special way in which the supervisor execution environment is initialized, the availability of internal interfaces implementing powerful but primitive operations, and the ability to access all segments in the address space of a process, regardless of the protection rings. Even if a module implementing only a private function were found, it might not execute outside the supervisor environment without being carefully modified.

This year the most important removal activities have been centered on the file system. In a project now almost completed, the functions of dynamic intersegment linking and directing the search of the file system to satisfy a symbolic reference have been removed from the supervisor. This project is notable for two reasons. First, it removed an especially vulnerable and complex mechanism from the supervisor. The vulnerability is a result of the linker having to accept user-constructed code segments as input data; the chances of such a complex "argument", if maliciously malstructured, causing the linker to malfunction while executing in the supervisor were demonstrated to be very high by numerous accidents. The complexity is apparent in that the linker's removal eliminated 10% of the gate entry points into the supervisor. The second interesting result of the linker's removal was the demonstration that linking procedures together across protection boundaries, i.e., rings, could be done without resorting to a

---

*There may still exist other performance penalties associated with removing functions from the supervisor that will inhibit production of the smallest possible kernel. One goal of the research is to understand better the performance cost of security.

mechanism common to both protection regions.

A second project related to the file system is the removal from the supervisor of the facilities for managing the association between names and the segments in the address space of a process. This project, now in its initial implementation phase, requires that a data base central to the management of the address space, the known segment table, be split into a private and a common part, and that the supervisor learn to lie convincingly on occasion about the existence of certain file system directories. The project will result in a new, simpler interface to the file system portion of the supervisor. Instead of identifying a directory by the sequence of character string names locating it in the directory hierachy, a segment number for the directory will be used. The notion of a tree name locating an element of the hierachy is thus removed from the supervisor to be implemented by procedures executing in the user protection environment (the actual file system hierachy still remains protected inside the supervisor).

Another removal project of a different flavor is investigating the possibility of moving most of system initialization from executing inside the supervisor each time the system is started to executing once in a user environment of a previous system. The idea is to produce as a system tape a bit pattern which, when loaded into memory, manifests a fully initialized system, rather than letting the system bootstrap itself in a complex way each time it is loaded from a tape containing the separate pieces. One pattern of operation may be much simpler to certify than the other.

The third category of activity is simplifying those mechanisms that must remain in the kernel. Such activities can reduce both the size and the complexity of the kernel. Simplification activities cover a broad range. In some cases a piece of the kernel can simply be eliminated because its function can be duplicated by another kernel mechanism. For example, the possibility of replacing all mechanisms for performing external I/O (to terminals, tape drives, card readers, card punches, and printers) with the ARPA Network attachment is being explored. This would remove from the kernel a large bulk of special mechanisms for managing the various I/O devices, leaving behind a single mechanism for managing the network attachment. Using network technology to provide the only path for external I/O to Multics appears feasible. Internal I/O functions (for managing the virtual memory, performing backup, and loading the system) would still be managed in the kernel.

Another example of simplification involves a less obvious duplication of mechanisms. A new buffering strategy for input and output from the network has been devised which, by utilizing the virtual memory, provides a core resident buffer which

appears to be of infinite length. The infinite buffer scheme is much simpler than the old circular buffer which had to be used over and over again, with attendant problems of old messages not being removed before a complete circuit of the buffer was made. The old buffer scheme was really providing a special purpose storage management facility, and the simplification was to use the standard storage management facility of the system -- the virtual memory -- for this function.

Several specific simplification projects involve using mulitiple parallel processes to implement kernel functions. A characteristic of the current Multics implementation is that processes are relatively expensive, for each must have an independent address space. As a result, many system functions involving inherently parallel activities are forced into sequential algorithms. The cost is increased complexity. As a basis for reimplementing such functions taking advantage of their natural parallelism, a facility providing low cost processes is being implemented. The processes are made cheap by having several of them share the same address space. Several applications of cheap processes are underway. Each interrupt handler will be assigned its own process in which to execute, rather than being forced to inhabit whatever user process was running when the interrupt occurred. As a result, the system interrupt interceptor will simply turn each interrupt into a wakeup of the corresponding process. By virtue of being full-fledged processes, the interrupt handlers can use the normal system interprocess communication mechanisms to coordinate their activities with one another and the user process, greatly simplifying their structure.

Another important application of low cost processes is in simplifying the structure of system resource management algorithms. The mechanism for moving pages among the three levels of the memory hierarchy is a good example. Whenever a missing page fault occurs in a process, the fault handler attempts to initiate the transfer of the desired page from bulk store or disk to core. This can only be done if a free core block is available. If not, then the fault handler first must move a page from core to the bulk store to make room. This, in turn, is possible only if a free block of bulk store is available. If not, a page must be moved from the bulk store, via core, to a disk by the fault handler. This complex series of steps occurs sequentially with page control executing in the process which took the page fault and then in various other user processes that happen to receive the subsequent I/O interrupts. The new scheme involving multiple dedicated processes is much simpler. One process runs in a loop making sure that some small number of free core blocks always exist. Whenever the number of free core blocks drops below that number, this process is awakened to transfer pages to bulk store. Another keeps space free on the bulk store by moving pages to disk when required. The core freeing process is activated by wakeups from

processes that have taken a page fault and discovered a lack of free core blocks. The bulk store freeing process is driven in a similar manner by the core freeing process. The path taken by a user process on a page fault is greatly simplified. This process can wait until a core block is free and then initiate the transfer of the desired page into core. The overall structure looks as though it will be much simpler than that currently employed.

The various simplification activities will eventually extend to all parts of the kernel, and to the overall structure of the kernel. Careful attention will be paid to the proper modularization of the entire kernel.

The final category of activity is partitioning the kernel into differently protected pieces that can be certified separately, some perhaps less carefully than others. While the specific projects in this category are less well developed than those for other categories, two techniques for partitioning seem worth exploring. The first is dividirg the kernel that is part of each process into mutiple layers in different rings of protection. For example, the bottom layer might implement a file system in which all segments were named by system generated unique identifiers. The next layer would implement a user named directory hierarchy on top of the primitive first layer file system. Another suggestion is that mechanisms to provide absolute compartmentalization of users and stored information be implemented at the bottom layer, and mechanisms to allow controlled sharing within the compartments be implemented at the next layer. This last suggestion is particularly intriguing, because if correctly done the notion of minimizing common mechanisms would be well supported. The second layer mechanisms would be common only within each compartment.

The second partitioning technique under investigation is separating the policy component from the mechanism component of resource management algorithms by putting the policy algorithms in a non-kernel protection ring in special system processes. For example, the process described earlier that removed pages from core memory could be arranged as a multi-ring process. The standard system kernel would execute in the most privileged ring along with some special gate entry points to implement movement of a particular page from core to a particular free block on the bulk store. Other gates would provide usage information on pages in core. The policy algorithm that decides which page to remove when another free core block needs to be generated would execute in a less privileged ring. The special gates into the supervisor would be used to do the actual moving, once a decision was made. The policy algorithm, however, could never read or write the contents of pages, learn the segment to which each page belonged, or cause one page to overwrite another, because the supervisor gates would be programmed to prevent these actions. The

result is that the policy algorithm could never cause unauthorized use or modification of the information stored in the pages. It could only cause denial of use. Under the circumstances that denial of use was deemed less serious than the other security violations, the policy algorithm need not be as carefully certified as the rest of the kernel. It appears that the idea of separating policy from mechanisms applies to all resource management algorithms.

This completes the discussion of the specific techniques to be used by the Computer Systems Research Division to produce a certifiably secure kernel for Multics. The next section details the specific tasks performed during the progress report period.

4. Tasks in the Certification Project

The following lists all tasks on which progress was made during the year. Since the list is complete, some of the tasks mentioned here duplicate the samples of the previous section.

a. Census of Ring 0.

As a first step in the certification of the Multics system, it was necessary to get some rough idea of the magnitude and structure of the present kernel of the system. To provide this information, Victor Voydock prepared a summary of the size of all the ring 0 modules, and listed these modules according to what subsystem they were a part of, and according to their source language. This overview of the present system was very helpful in determining which components of the system ought to be attacked first.

b. Removal of the Linker from Ring 0.

The project of removing the linker from ring 0 was undertaken by Philippe Janson as a Master's thesis, which will be completed in May, 1974. It was important that the linker be removed, since several other components of the system, in particular the management of reference names, could conceivably be removed from the kernel after the linker had been removed. The user ring version of the linker which he created is now completely operational; the only task remaining before the linker can be installed in the standard system is to insure that performance of the new linker is at least equivalent to the performance of the linker currently being used in the kernel.

c. Removal of Name-Space Management From Ring 0.

One of the functions of the Known Segment Table, or KST, is to remember the associations between segment numbers and reference names on a per-process basis. One of the principal users of this facility is the linker, which must resolve named references into segment numbers. With the linker now existing in the user ring, it is possible to consider removing portions of the KST manager into the user ring as well. Richard Bratt, as part of his Master's thesis research, has proposed a scheme for moving this function to the user ring which eliminates the drawbacks of allowing the user to directly initiate directories. The only function which remains inside the kernel of the system in his proposal is the association between segment numbers and unique ID's.

d. Removal of the Storage Hierarchy from Ring 0.

The two previous tasks represent removal from the kernel of the system of much of the per-process segment name management. Douglas Hunt is also considering whether certain of the system-wide name management mechanism could be removed from the kernel or at least partitioned in a separate area of the kernel. In particular, he is considering whether the concept of the storage hierarchy could be removed from the kernel, leaving within the kernel only a catalog of segments indexed by unique ID. In the outer ring an association would be maintained between name, unique ID, and segment number.

e. Removal of User I/O from Ring 0.

A thesis completed by David Clark, now available as Project MAC Technical Report TR-117, discusses a strategy for handling user-initiated I/O which operates almost completely in the user ring. The only function which is required within the kernel is the management of multiplexed devices. The scheme uses as the buffering strategy for I/O the virtual memory management algorithm of the system itself. The scheme described in the thesis effectively removes I/O from the kernel of the system; however, it requires an I/O controller with capabilities slightly greater than the one currently available on Multics, so that this particular removal will not actually be implemented in the near future. However, Honeywell has implemented an interface to the I/O system, which has some of the same features as the scheme described in the thesis.

f. Simplification to Page Control.

Andrew Huber is considering ways to simplify the memory management algorithm of the Multics system. In particular, he is considering a reorganization in which most of the functions of page control are executed in separate asynchronous processes, so that the only task which the user process executes is the actual fetching of a missing page. It is felt that by isolating functions in separate processes, and constraining them by restricting the interprocess communication paths, it will be easier to understand and certify the overall algorithm. One of the other benefits of structuring page control in this way is that it should be possible for several processors to take and handle a page exception simultaneously, without interfering with each other. We are currently preparing, in a high level language devised by Bernard Greenberg, a version of page control structured in this fashion, which can be used for discussion.

g. Simplification of Traffic Control.

The group is attempting to restructure the process scheduling algorithm of the Multics system. In particular, we are interested in separating two ideas, the actual switching of the processor from one process to another, and the decision making algorithm which determines which processes are eligible to run. It is hoped that this will speed up the act of switching from one process to another, and also make the algorithm easier to understand.

h. Removal of the Answering Service from the Kernel of the System.

The Answering Service, those algorithms which authenticate users, create processes, and manage teletype lines, must currently be considered within the kernel of the system, even though they are not within ring 0 but within a separate process. It is very desirable that we identify some component of this mechanism which, if properly isolated and certified, would eliminate the need to certify the remainder of the answering service. Warren Montgomery has proposed a strategy for user-authentication and process creation which would effectively remove from the kernel almost all of the current answering service mechanism. He is currently attempting to discover what problems arise from this proposed division of the function.

i. The Use of Multiple Processes as an Organizational Tool.

As the discussion of page control suggested, an approach to understanding the structure of the system is to separate portions of it into separate processes. There are a variety of projects underway to explore this organizational structure. We are currently developing a special kind of process which runs only within ring 0, which has limited capability, and is very efficient to execute. It is our intention to use this kind of process in several applications within the system kernel. The example of page control has already been given. Another application of these processes is in the handling of interrupts. Code which executes at interrupt time is subject to several special constraints, for example, it may not abandon the processor and it may not loop on a lock. This makes code which runs at interrupt time much more complex and prone to errors. Running this code instead in a different process will eliminate these sorts of problems. Robert Mabee is currently proceeding with the implementation of this sort of process, and as a test case intends to modify the typewriter control software so that the code now running at interrupt time runs instead in a process.

Another experiment with the use of multiple processes involves modifications of the user ring environment so that the single process of the user is conceptually shared between a number of tasks, all running in the same virtual address space. This structure, in addition to simplifying the user ring environment, seems to have several beneficial effects on the structure of the kernel. The handling of the Multics "quit" is an obvious example. The propagation of this signal through the kernel from its receipt by the interrupt handler to the ultimate process interrupt is very complex and tortuous. It appears that the simplest thing for the kernel to do with a "quit" signal is to translate it immediately into a wake-up for some process. It is not clear, however, what arrangement of processes in the user ring can best take advantage of this interpretation of a "quit". The current experiment with the user ring environment will let us explore how to take advantage of this interpretation of the "quit".

j. Restructuring of Network Control Program.

Because of our group's direct involvement in the connection of Multics to the ARPA Network, we have developed significant expertise in that portion of the system. For this reason, the network is a good candidate for investigation of techniques for simplification of the system kernel. Having the network software properly structured is especially important, since it is possible that a suitable network could serve as the sole form of external I/O. For this function it seems appropriate to use multiple processes as a tool for simplification. In this case, it may be possible to

remove some of the processes from the kernel thus reducing directly the bulk of the supervisor code related to the ARPA network. Notice that the ARPA network is especially interesting in this respect, since, being a multiplexed facility, some protection is required of the de-multiplexing and multiplexing function. Any technique that removes this from the kernel, without compromising its security, is an especially valuable modification to the system.

k. System Initialization.

In order to certify the Multics system, it will be necessary to certify the "initial state" of the system; the ad hoc initialization techniques currently used makes such a certification very difficult, if not impossible. The intention of this study is to discover new ways of initializing the system which are more amenable to certification.

l. Recovery from System Errors.

Related to the issue of system initialization is the issue of recovery from errors. They have in common the requirement that it is necessary to certify or assure that the system is in some known state. We are interested in determining whether there are some particular structures for data bases and algorithms which makes it much easier to assure that the data base is in fact consistent and correct. One particular project which we are carrying out in an attempt to learn more about the structure of data bases is a comparative analysis of Multics and the Burroughs operating system, currently being prepared by Ben Williams. The Burroughs Master Control Program apparently recovers very effectively from a wide variety of errors, and we are hopeful that insight gained from an understanding of this system can help the Multics system recover from errors more gracefully and reliably.

m. High-Level Description of System Functionality.

As part of any attempt to certify a system, it is necessary to have some description of the intended functionality of the system itself to serve as a standard against which to certify. Several members of the project have tried various notational schemes for describing the functionality of various parts of the system. A representation of system data bases and related algorithms in the Vienna Definition Language was performed by Richard Bratt, using the known segment table as a case study. A similar description with directory control using English as the descriptive language was performed by Douglas Hunt. Finally, Bernard Greenberg, as part of his thesis (Project MAC Technical Report TR-127), has devised a language for describing programs with complexly structured data bases, which attempts to avoid implications

concerning the implementation of the data base structure. This language is now being used to represent various alternative algorithms being considered as part of the restructuring of page control.

n. Formulation of Criteria for Inclusion of Modules Within the Kernel.

Richard Feiertag is currently attempting to develop a specific set of rules which would determine whether a module should or should not be included within the kernel of the system. He is attempting to identify these rules by studying a number of specific parts of the current system and identifying the trade-offs related to moving these particular parts out of the kernel. He is currently studying the separation of policy from mechanism in page control.

o. Study of Multics Security Holes.

An understanding of how system bugs arise and what is required to fix them gives insight into the problems of certification. For this reason we are interested in the discovery and understanding of the flaws in the current system. We attempt, periodically, to catalog all known ways to violate the security of Multics, and to identify the general class of problems of which each bug is a specific example.

p. Implementation of New I/O Buffering Strategy

As part of designing a simple structure for I/O, we are producing a buffer management mechanism which uses the virtual memory manager algorithm. This is described in the section of the progress report which describes CSR division work on the ARPA network.

C. ARPA NETWORK ACTIVITIES

This last year has been a year of transition for those working on the ARPA network, as the development effort necessary to get the network operational on Multics is gradually phasing out, to be replaced with more research-oriented projects.

These research efforts on which the division is now embarking represent the best indication of the direction in which we see ourselves going in the next year.

As part of this transition, the number of staff assigned to this area has dropped from five full-time to one full-time and two half-time, with the half-time staff spending the remainder of their time working on projects related to the division's

certification effort. This sharing of staff is part of our effort to unify the various efforts of the division.

The various network tasks in which the division has engaged are detailed below.

1. Conversion of Multics to New Hardware

Before and during the Summer of 1973, the Multics operating system was modified to run on a Honeywell 6180, rather than a Honeywell 645. It was necessary, as part of this transition, to modify the Multics software so that the ARPA network would operate with the 6180. Several sub-tasks were implied by this transition. First, it was necessary to redesign a portion of the Network Control Program so that it would interface to a full duplex rather than a half duplex network interface. Our experience, in common with that of the network community at large, was that the half duplex interface was undesirable; we seized this opportunity to convert to full duplex. It was also necessary, as part of the move, to construct a new hardware interface between the 6180 and the IMP. The interface designed for the 645 was inappropriate, first because it was half duplex, and second because it did not support the distant interface, which we intended to use.

It has been our intention to hand off the day-to-day operation of the ARPA network to the staff of the Information Processing Center. For this reason, we invested some effort during the move in developing and integrating the network software so that it could be manipulated by the Multics system operators, as part of their standard procedures. This effort, which involved automating the starting, stopping, and recovering from errors of the network, was successful to the extent that we now need intervene only in very exceptional circumstances, such as a network crash or a hardware failure.

In the switching from one machine to another, the physical location of the hardware changed. For this and other reasons, the M.I.T. community obtained a second IMP. A certain amount of effort was required on our part to change over from the old to the new IMP. This task has been completed, and the operational responsibility for this IMP has been given to the Information Processing Center's staff.

2. Improvements to Network Service

During the course of the year, the Computer Systems Research Division has completed several tasks to enhance the quality of the network service provided by Multics. Several programs have been rewritten, either to enhance performance or to eliminate bugs. For example, the user interface to the Telnet protocol, which previously existed as a privately maintained program on Multics, has been upgraded and made a part of the official Multics network software library. There is a continuing effort to enhance the mail sending facility, both in-bound and out-bound. Multics now supports a facility which allows unsendable out-bound mail to be queued for later retransmission. In-bound mail can now be delivered without the sender knowing the project ID of the recipient. We have recently designed a modification to in-bound mail handling which would decrease the cost and increase the reliability. This feature should be installed in the near future.

The network documentation has been upgraded substantially during the year. We assembled a draft version of a Network User's Supplement to the Multics Programmer's Manual, which gives an overview of the ARPA network on Multics, and describes how to get into Multics from the ARPA network, and how to get out to the ARPA network from Multics. In addition to the Network User's Supplement, we are also producing a description of internal interfaces and program strategies of the system programs which support network use, to be published as one of the series of Program Logic Maunuals which Honeywell is producing to describe the Multics operating system.

3. New Protocols

During the year our group has participated in the development and implementation of several new network protocols. Code to implement the new Telnet protocol was implementd and installed by Douglas Wells. Kenneth Pogran has contributed to the current redesign of the file transfer protocol and plans to implement this new protocol whenever the specificiation has stabilized. In addition to these two new protocol revisions, which have required significant effort on our part, Rajendra Kanodia has also proposed modifications to the host protocol which attempt to deal with lost messages in a more graceful manner than the current protocol.

4. Research Topics

The division has carried out several research tasks in the network area this year; this is perhaps the most interesting work done relating to the ARPA network. In an attempt to increase the efficiency of transmitting data to and from the network, and

at the same time gain a more basic understanding of the interaction between input/output functions and virtual memory computer systems, Rajendra Kanodia and David Clark have been devising a new buffering strategy for reading and writing from the network. The result of this design is a buffer which, by utilizing the virtual memory, appears to be infinite in length. This use of the virtual memory eliminates any need to compact or otherwise manage the buffer area, thus reducing overhead. The buffer is also directly accessible to a user process, since it is in the virtual address space; this avoids the necessity of copying data in order to make it accessible. It is our intention to attempt to use this same buffering strategy to interface the typewriter control processor as well as the ARPA network; it appears that this strategy can be exploited successfully for all devices for which the system's nucleus must take responsibility. This unification of buffer management, by reducing the bulk and complexity of the kernel, is a significant contribution to our certification project discussed earlier in the report.

As part of our research into the use of the ARPA network as a vehicle for communication between active processes, Warren Montgomery and Kenneth Pogran have been studying and implementing the RSEXEC protocol devised by Robert H. Thomas of Bolt, Beranek and Newman. Initial investigation of this protocol suggests that parts of it can be adapted easily for our operating system, but the other parts, in particular the manipulation of files, are difficult to dovetail into our particular view of a storage system and user interface thereto. More research is intended in this area, as time and personnel allow; we hope that a partial implementation of RSEXEC will become available on Multics.

In a project related to the previous one, we have currently established a process in Multics which is always available for the purpose of providing services related to the ARPA network. Currently, this process will queue and retransmit mail which for some reason is currently undeliverable across the network. In the near future, we intend that this process will handle in-bound mail, and support those functions of RSEXEC which we are able to implement.

As part of our redesign of the buffering strategy for the network, Arthur Benjamin has attempted to increase the precision of the meters which tell us about traffic flow between Multics and the ARPA network.

Michael Padlipsky has developed the specification for a network-wide text editor, called neted, which has been implemented and used at eight different sites on the ARPA network. It is felt that this editor serves as a small but valid example of the fashion in which one standard command interface can be adapted to various time

sharing systems, with their various user interfaces.

5. Network Committees

During the year our group has contributed to the network effort by active participation in several ARPA network committees. These include the "USING" subcommittees which are specifying the common command language and neted; the committee which is redesigning the file transfer protocol; the Review Committee for the ARPA network terminal system (ANTS) and the ELF operating system for the PDP-11; the Graphics protocol committee; and the informal group redesigning the ARPA network host-to-host protocols.

D. TECHNOLOGY TRANSFER

The term "technology transfer" may be loosely applied to making research results accessible to industry, government, and educational institutions. Since, in undertaking the Multics project, the Computer Systems Research Division developed a sizable store of technology, transfer of that technology has recently been a major activity. Several specific points indicate the progress of this transfer:

1. In April, 1974, Honeywell Information Systems Inc. will announce availability of an upgraded hardware system for Multics, the model 68/80, which includes a cache memory in each processor and support for a large (2-8 million word) directly addressable memory.

2. Honeywell 6180 Multics installations were completed or scheduled at M.I.T., the Air Force Data Services Center, Ford Motor Co., and General Motors. These installations are in addition to the older Honeywell 645 installation at Rome Air Development Center and also Honeywell internal Multics sites in Billerica, Mass., Waltham, Mass., Phoenix, Arizona, Paris (Honeywell Bull) and Tokyo (Toshiba). In total, as of June 30, 1974, there will be three 645 sites and seven 6180 sites running Multics. Each of the non-Honeywell sites has initiated contact with M.I.T. to facilitate more direct transfer of knowhow and, in some cases, of M.I.T.-developed Multics subsystems.

3. The M.I.T. Information Processing Center and the Telefunken Co. completed negotiations for Telefunken's acquisition of knowhow and rights to utilize the Multics PL/I compiler.

4. As development of the Multics ARPANET attachment has matured, transfer of

that technology to Honeywell has begun. Although not yet fully supported by Honeywell, the network attachment is currently available to customers as an option, using M.I.T.-supplied software.

5. One M.I.T. staff member versed in ARPANET development has moved to the MITRE Corporation to help implement a government network similar to the ARPANET.

6. Of the ten undergraduate and graduate students of CSR who have completed or will soon complete their educational programs, four accepted positions with Honeywell.

7. Faculty members of the Computer Systems Research Division have been actively consulting with several different government and industrial organizations; these consulting activities largely consist of translating the Multics experience into forms applicable to other system designs. Organizations include Control Data Corporation, IBM Corporation, General Telephone and Electronics, and System Development Corporation, and groups within the Department of Defense. In a related activity, the Computer Systems Research Division entertained several industrial visitors interested in Multics technology, many making contact through the M.I.T. Industrial Liaison Office.

8. The first draft of a tutorial paper on the protection of information in computer systems was completed by Professors Saltzer and Schroeder. This paper captures for educational purposes many of the insights discovered in the course of Multics development. Also made available in Project MAC Technical Report form was the introduction to the Multics Programmers' Manual.

9. Japanese interest in computer systems technology has been high for several years; in most recent periods we have had one or more Japanese visitors. This year, Professor Eiiti Wada of the University of Tokyo joined the division as a visiting Associate Professor. Also, two books about Multics were published (in Japanese) by former visitors. The first was a translation of Elliott Organick's book The Multics System, by Akio Sasaki and Toyohiko Kikuchi. The second, a new book entitled Structure of Computer Utility - Anatomy of Multics, was written by Professor Katsuo Ikeda of Kyoto University.

E. OTHER ACTIVITIES

During the current reporting period Masters' theses by Robert M. Frankston and Lee J. Scheffler have been completed under the supervision of F. J. Corbato.

In the thesis by Frankston, entitled The Computer Utility as a Marketplace for Computer Services, the implications of widespread commercial use of a computer utility were explored. As stated in the abstract:

> "Most contemporary computer systems are oriented towards users who run programs. The environment for services puts different requirements on the computer system than do the needs of programmers, so as to permit all the participants in the market to make effective use of its facilities without requiring dedicated facilities and without interfering with each other. As with any marketplace, it must be convenient to do business within its framework."

The thesis, available as MAC TR-128, also includes as an example a design evolved from the Multics System which implements a particular marketplace model.

Performance Evaluation of Rotating Disk Subsystems in Multiprogrammed Computer Systems is the title of the thesis by Scheffler. This thesis analytically derives, using primarily the mathematics of Operations Reseach, expressions (or in some cases bounds) for the performance of various common classes of disk subsystems subject to work loads typical of large-scale multi-programmed computer utilities. As stated in the abstract, this study

> "...is fundamental to: 1) predicting the overall performance of the system; 2) enhancing disk performance to meet evolving secondary memory performance requirements; and 3) choosing disk subsystems to meet stated performance requirements."

The derived expressions were validated against measurements taken on the Multics System at M.I.T.

Another Master's thesis, by Bernard Greenberg, developed a set of measuring tools for exploring the paging rate of Multics for very large memory sizes. This work involved modifying Multics to intercept all movements of data to and from the disk and thereby record an address reference string. The reference string is

recorded during actual Multics operation, and later analysed with a least-recently-used algorithm simulator to discover what disk traffic rate would have resulted with larger main memory sizes. The measurement is complicated by a variety of real-life factors such as Multics cleverly not writing out pages which happen to contain all zeroes, and Greenberg succeeded in providing suitable corrective measures for each of them. Unfortunately the development and proof that the measuring tool worked was a thesis in itself; a systematic program of measurements with the tool awaits some other student.

Publications

1. Introduction to Multics, Project MAC Technical Report TR-123, November 1973.

2. MULTICS Programmer's Manual, Part I (Introduction), Revision 14, September 30, 1973.

3. MULTICS Programmer's Manual, Part II (Reference Guide), Revision 15, November 30, 1973.

4. Schroeder, M. D., with R. M. Graham, "Proceedings of the ACM SIGPLAN/SIGOPS Interface Meeting", SIGPLAN Notices 8, 9, September, 1973.

5. Schroeder, M. D., "A Brief Report on the SIGPLAN/SIGOPS Interface Meeting", Operating Systems Review 7, 3, July 1973, pp. 4-9.

Theses Completed

1. Clark, D. D., An Input/Output Architecture for Virtual Memory Computer Systems, Ph.D. thesis, M.I.T., Department of Electrical Engineering, August 1973.

2. Gumpertz, R. H., The Design and Fabrication of an ARPA Network Interface, S.B. thesis, M.I.T. Department of Electrical Engineering, July 1973.

3. Rotenberg, L. J., Making Computers Keep Secrets, Ph.D thesis, M.I.T. Department of Electrical Engineering, June 1973.

4. Stern, J. A., Backup and Recovery of On-Line Information in a Computer Utility, E. E. thesis, M.I.T., Department of Electrical Engineering, August 1973.

ARPA Network Committee memberships

1. Padlipsky, M. A.: User Interest Group (USING) and subcommittees.

2. Thomas, E. L.: ARPA Network Graphics Group

3. Pogran, K. T.: ARPA Network Graphics Group, ARPA Network nal System Steering Commitee

TECHNICAL MANUALS

TM-10 Jackson, James N.
Interactive Design Coordination
for the Building Industry
June 1970
AD 708-400

*TM-11 Ward, Philip W.
Description and Flow Chart of the
PDP-7/9 Communications Package
July 1970
AD 711-379

*TM-12 Graham, Robert M.
File Management and Related Topics
(Formerly Programming Linguistics
Group Memo No. 6, June 12, 1970)
September 1970
AD 712-068

*TM-13 Graham, Robert M.
Use of High Level Languages
for Systems Programming
(Formerly Programming Linguistics
Group Memo No. 2, November 20, 1969)
September 1970
AD 711-965

*TM-14 Vogt, Carla M.
Suspension of Processes in a Multi-
processing Computer System
(Based on S.M. Thesis, EE Dept.,
February 1970)
September 1970
AD 713-989

TM's 1-9 were never issued.

Preceding page blank

TM-15 Zilles, Stephen N.
An Expansion of the Data Structuring Capabilities of PAL
(Based on S.M. Thesis, EE Dept., June 1970)
October 1970

AD 720-761

TM-16 Bruere-Dawson, Gerard
Pseudo-Random Sequences
(Based on S.M. Thesis, EE Dept., June 1970)
October 1970

AD 713-852

TM-17 Goodman, Leonard I.
Complexity Measures for Programming Languages (Based on S.M. Thesis, EE Dept., September 1971)
September 1971

AD 729-011

*TM-18 Reprinted as TR-85

*TM-19 Fenichel, Robert R.
A New List-Tracing Algorithm
October 1970

AD 714-522

*TM-20 Jones, Thomas L.
A Computer Model of Simple Forms of Learning (Based on Ph.D. Thesis, EE Dept., September 1970)
January 1971

AD 720-337

*TM-21 Goldstein, Robert, C.
The Substantive Use of Computers for Intellectual Activities
April 1971

AD 721-618

TM-22 Wells, Douglas M.
Transmission of Information Between a Man-Machine Decision System and Its Environment
April 1971
AD 722-837

TM-23 Strnad, Alois J.
The Relational Approach to the Management of Data Bases
April 1971
AD 721-619

TM-24 Goldstein, Robert C. and Alois J. Strnad
The MacAIMS Data Management System
April 1971
AD 721-620

TM-25 Goldstein, Robert C.
Helping People Think
April 1971
AD 721-998

TM-26 Iazeolla, Giuseppe G.
Modeling and Decomposition of Information Systems for Performance Evaluation
June 1971
AD 733-965

TM-27 Bagchi, Amitava
Economy of Descriptions and Minimal Indices
January 1972
AD 736-960

TM-28 Wong, Richard
Construction Heuristics for Geometry and a Vector Algebra Representation of Geometry
June 1972
AD 743-487

TM-29 Hossley, Robert and Charles Rackoff
The Emptiness Problem for Automata on Infinite Trees
Spring 1972

AD 747-250

TM-30 McCray, William A.
SIM360: A S/360 Simulator
(Based on S.B. Thesis, ME Dept., May 1972)
October 1972

AD 749-365

TM-31 Bonneau, Richard J.
A Class of Finite Computation Structures Supporting the Fast Fourier Transform
March 1973

AD 757-787

TM-32 Moll, Robert
An Operator Embedding Theorem for Complexity Classes of Recursive Functions
May 1973

AD 759-999

TM-33 Ferrante, Jeanne and Charles Rackoff
A Decision Procedure for the First Order Theory of Real Addition with Order
May 1973

AD 760-000

TM-34 Bonneau, Richard J.
Polynomial Exponentiation: The Fast Fourier Transform Revisited
June 1973

PB 221-742

TM-35 Bonneau, Richard J.
An Interactive Implementation of the Todd-Coxeter Algorithm
December 1973

AD 770-565

TM-36 Geiger, Steven P.
A User's Guide to the Macro Control Language
December 1973

AD 771-435

TM-37 Schonhage, A.
Real-Time Simulation of Multidimensional Turing Machines by Storage Modification Machines
December 1973

PB 226-103/AS

TM-38 Meyer, Albert R.
Weak Monadic Second Order Theory of Succesor is not Elementary-Recursive
December 1973

PB 226-514/AS

TECHNICAL REPORTS

*TR-1 Bobrow, Daniel G.
Natural Language Input for a Computer Problem Solving System,
Ph.D. Thesis, Math. Dept.
September 1964
AD 604-730

*TR-2 Raphael, Bertram
SIR: A Computer Program for Semantic Information Retrieval,
Ph.D. Thesis, Math. Dept.
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. Feldman
Verbal and Graphical Language for the AED System: A Progress Report
May 1964
AD 604-678

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

TR's 5, 9, 10, 15 were never issued

TR-7 Weizenbaum, Joseph
OPL-1: An Open Ended Programming System within CTSS
April 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 1964
AD 609-296

*TR-13 Greenberger, Martin
A New Methodology for Computer Simulation
October 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
An Analysis of Time-Shared Computer Systems,
Ph.D. Thesis, EE Dept.
June 1965
AD 470-715

TR-19 Russo, Francis John
A Heuristic Approach to Alternate Routing in a Job Shop,
S.B. & S.M. Thesis, Sloan School
June 1965
AD 474-018

TR-20 Wantman, Mayer Elihu
CALCULAID: An On-Line System for
Algebraic Computation and Analysis,
S.M. Thesis, Sloan School
September 1965
AD 474-019

*TR-21 Denning, Peter James
Queueing Models for File Memory Operation,
S.M. Thesis, EE Dept.
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 Multi-programmed
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
Investigation of an Analog Technique to Decrease Pen-Tracking Time in Computer Displays,
S.M. Thesis, EE Dept.
March 1966

AD 631-396

TR-26 Cheek, Thomas Burrell
Design of a Low-Cost Character Generator for Remote Computer Displays,
S.M. Thesis, EE Dept.
March 1966

AD 631-269

TR-27 Edwards, Daniel James
OCAS - On-Line Cryptanalytic Aid System,
S.M. Thesis, EE Dept.
May 1966

AD 633-678

TR-28 Smith, Arthur Anshel
Input/Output in Time-Shared, Segmented, Multiprocessor Systems,
S.M. Thesis, EE Dept.
June 1966

AD 637-215

TR-29 Ivie, Evan Leon
Search Procedures Based on Measures of Relatedness between Documents,
Ph.D. Thesis, EE Dept.
June 1966

AD 636-275

TR-30 Saltzer, Jerome Howard
Traffic Control in a Multiplexed Computer System,
Sc.D. Thesis, EE Dept.
July 1966

AD 635-966

TR-31 Smith, Donald L.
Models and Data Structures for Digital Logic Simulation,
S.M. Thesis, EE Dept.
August 1966

AD 637-192

*TR-32 Teitelman, Warren
PILOT: A Step Toward Man-Computer Symbiosis,
Ph.D. Thesis, Math. Dept.
September 1966

AD 638-446

*TR-33 Norton, Lewis M.
ADEPT - A Heuristic Program for Proving Theorems of Group Theory,
Ph.D. Thesis, Math. Dept.
October 1966

AD 645-660

*TR-34 Van Horn, Earl C., Jr.
Computer Design for Asynchronously Reproducible Multiprocessing,
Ph.D. Thesis, EE Dept.
November 1966

AD 650-407

*TR-35 Fenichel, Robert R.
An On-Line System for Algebraic Manipulation,
Ph.D. Thesis, Appl. Math. (Harvard)
December 1966

AD 657-282

*TR-36 Martin, William A.
Symbolic Mathematical Laboratory,
Ph.D. Thesis, EE Dept.
January 1967

AD 657-283

*TR-37 Guzman-Arenas, Adolfo
Some Aspects of Pattern Recognition by Computer,
S.M. Thesis, EE Dept.
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.
Program Analysis by Digital Computer,
Ph.D. Thesis, EE Dept.
August 1967

AD 662-224

TR-44 Gorry, G. Anthony
A System for Computer-Aided Diagnosis,
Ph.D. Thesis, Sloan School
September 1967

AD 662-665

TR-45 Leal-Cantu, Nestor
On the Simulation of Dynamic Systems
with Lumped Parameters and Time Delays,
S.M. Thesis, ME Dept.
October 1967

AD 663-502

TR-46 Alsop, Joseph W.
A Canonic Translator,
S.B. Thesis, EE Dept.
November 1967

AD 663-503

*TR-47 Moses, Joel
Symbolic Integration,
Ph.D. Thesis, Math. Dept.
December 1967

AD 662-666

TR-48 Jones, Malcolm M.
Incremental Simulation on a Time-
Shared Computer,
Ph.D. Thesis, Sloan School
January 1968

AD 662-225

TR-49 Luconi, Fred L.
Asynchronous Computational Structures,
Ph.D Thesis, EE Dept.
February 1968

AD 667-602

*TR-50 Denning, Peter J.
Resource Allocation in Multiprocess Computer Systems,
Ph.D. Thesis, EE Dept.
May 1968

AD 675-554

*TR-51 Charniak, Eugene
CARPS, A Program which Solves Calculus Word Problems,
S.M. Thesis, EE Dept.
July 1968

AD 673-670

TR-52 Deitel, Harvey M.
Absentee Computations in a Multi-Access Computer System,
S.M. Thesis, EE Dept.
August 1968

AD 684-738

*TR-53 Slutz, Donald R.
The Flow Graph Schemata Model of Parallel Computation,
Ph.D. Thesis, EE Dept.
September 1968

AD 683-393

TR-54 Grochow, Jerrold M.
The Graphic Display as an Aid in the Monitoring of a Time-Shared Computer System,
S.M. Thesis, EE Dept.
October 1968

AD 689-468

TR-55 Rappaport, Robert L.
Implementing Multi-Process Primitives in a Multiplexed Computer System,
S.M. Thesis, EE Dept.
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
Lambda-Calculus Models of Programming Languages,
Ph.D. Thesis, Sloan School
December 1968

AD 683-394

TR-58 Greenbaum, Howard J.
A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System,
S.M. Thesis, EE Dept.
January 1969

AD 686-988

*TR-59 Guzman, Adolfo
Computer Recognition of Three-Dimensional Objects in a Visual Scene,
Ph.D. Thesis, EE Dept.
December 1968

AD 692-200

*TR-60 Ledgard, Henry F.
A Formal System for Defining the Syntax and Semantics of Computer Languages,
Ph.D. Thesis, EE Dept.
April 1969

AD 689-305

TR-61 Baecker, Ronald M.
Interactive Computer-Mediated Animation,
Ph.D. Thesis, EE Dept.
June 1969

AD 690-887

TR-62 Tillman, Coyt C., Jr. (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

TR-63 Brackett, John W., Michael Hammer and Daniel E. Thornhill
Case Study in Interactive Graphics Programming: A Circuit Drawing and Editing Program for Use with a Storage-Tube Display Terminal
October 1969

AD 699-930

*TR-64 Rodriguez, Jorge E. (ESL-R-398)
A Graph Model for Parallel Computations,
Sc.D. Thesis, EE Dept.
September 1969

AD 697-759

*TR-65 DeRemer, Franklin L.
Practical Translators for LR(k) Languages,
Ph.D. Thesis, EE Dept.
October 1969

AD 699-501

*TR-66 Beyer, Wendell T.
Recognition of Topological Invariants by Iterative Arrays,
Ph.D. Thesis, Math. Dept.
October 1969

AD 699-502

*TR-67 Vanderbilt, Dean H.
Controlled Information Sharing in a Computer Utility,
Ph.D. Thesis, EE Dept.
October 1969

AD 699-503

*TR-68 Selwyn, Lee L.
Economies of Scale in Computer Use: Initial Tests and Implications for The Computer Utility,
Ph.D. Thesis, Sloan School
June 1970

AD 710-011

*TR-69 Gertz, Jeffrey L.
Hierarchical Associative Memories for Parallel Computation,
Ph.D. Thesis, EE Dept.
June 1970

AD 711-091

*TR-70 Fillat, Andrew I., and Leslie A. Kraning
Generalized Organization of Large Data-Bases: A Set-Theoretic Approach to Relations,
S.B. & S.M. Thesis, EE Dept.
June 1970

AD 711-060

*TR-71 Fiasconaro, James G.
A Computer-Controlled Graphical Display Processor,
S.M. Thesis, EE Dept.
June 1970

AD 710-479

TR-72 Patil, Suhas S.
Coordination of Asynchronous Events,
Sc. D. Thesis, EE Dept.
June 1970

AD 711-763

*TR-73 Griffith, Arnold K.
Computer Recognition of Prismatic Solids,
Ph.D. Thesis, Math. Dept.
August 1970

AD 712-069

TR-74 Edelberg, Murray
Integral Convex Polyhedra and an Approach to Integralization,
Ph.D. Thesis, EE Dept.
August 1970

AD 712-070

TR-75 Hebalkar, Prakash G.
Deadlock-Free Sharing of Resources in Asynchronous Systems,
Sc.D. Thesis, EE Dept.
September 1970

AD 713-139

*TR-76 Winston, Patrick H.
Learning Structural Descriptions from Examples,
Ph.D. Thesis, EE Dept.
September 1970

AD 713-988

TR-77 Haggerty, Joseph P.
Complexity Measures for Language Recognition by Canonic Systems,
S.M. Thesis, EE Dept.
October 1970

AD 715-134

TR-78 Madnick, Stuart E.
Design Strategies for File Systems,
S.M. Thesis, EE Dept. & Sloan School
October 1970

AD 714-269

TR-79 Horn, Berthold K.
Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View,
Ph.D. Thesis, EE Dept.
November 1970
AD 717-336

TR-80 Clark, David D., Robert M. Graham, Jerome H. Saltzer and Michael D. Schroeder
The Classroom Information and Computing Service
January 1971
AD 717-857

TR-81 Banks, Edwin R.
Information Processing and Transmission in Cellular Automata,
Ph.D. Thesis, ME Dept.
January 1971
AD 717-951

*TR-82 Krakauer, Lawrence J.
Computer Analysis of Visual Properties of Curved Objects,
Ph.D. Thesis, EE Dept.
May 1971
AD 723-647

TR-83 Lewin, Donald E.
In-Process Manufacturing Quality Control,
Ph.D. Thesis, Sloan School
January 1971
AD 720-098

*TR-84 Winograd, Terry
Procedures as a Representation for Data in a Computer Program for Understanding Natural Language,
Ph.D. Thesis, Math. Dept.
February 1971
AD 721-399

TR-85 Miller, Perry L.
Automatic Creation of a Code Generator from a Machine Description,
E.E. Degree, EE Dept.
May 1971

AD 724-730

TR-86 Schell, Roger R.
Dynamic Reconfiguration in a Modular Computer System,
Ph.D. Thesis, EE Dept.
June 1971

AD 725-859

TR-87 Thomas, Robert H.
A Model for Process Representation and Synthesis,
Ph.D. Thesis, EE Dept.
June 1971

AD 726-049

TR-88 Welch, Terry A.
Bounds on Information Retrieval Efficiency in Static File Structures,
Ph.D. Thesis, EE Dept.
June 1971

AD 725-429

TR-89 Owens, Richard C., Jr.
Primary Access Control in Large-Scale Time-Shared Decision Systems,
S.M. Thesis, Sloan School
July 1971

AD 728-036

TR-90 Lester, Bruce P.
Cost Analysis of Debugging Systems,
S.M. & S.B. Thesis, EE Dept.
September 197[illegible]

AD 730-521

*TR-91 Smoliar, Stephen W.
A Parallel Processing Model of
Musical Structures,
Ph.D. Thesis, Math. Dept.
September 1971

AD 731-690

TR-92 Wang, Paul S.
Evaluation of Definite Integrals
by Symbolic Manipulation
Ph.D. Thesis, Math. Dept.
October 1971

AD 732-005

TR-93 Greif, Irene Gloria
Induction in Proofs about Programs,
S.M. Thesis, EE Dept.
February 1972

AD 737-701

TR-94 Hack, Michel Henri Theodore
Analysis of Production Schemata
by Petri Nets,
S.M. Thesis, EE Dept.
February 1972

AD 740-320

TR-95 Fateman, Richard J.
Essays in Algebraic Simplification
(A revision of a Harvard Ph.D. Thesis)
April 1972

AD 740-132

TR-96 Manning, Frank
Autonomous, Synchronous Counters Constructed Only of
J-K Flip-Flops,
S.M. Thesis, EE Dept.
May 1972

AD 744-030

TR-97 Vilfan, Bostjan
The Complexity of Finite Functions
Ph.D. Thesis, EE Dept.
March 1972

AD 739-678

TR-98 Stockmeyer, Larry Joseph
Bounds on Polynomial Evaluation Algorithms
S.M. Thesis, EE Dept.
April 1972

AD 740-328

TR-99 Lynch, Nancy Ann
Relativization of the Theory of Computational Complexity
Ph.D. Thesis, Math. Dept.
June 1972

AD 744-032

TR-100 Mandl, Robert
Further Results on Hierarchies of Canonic Systems
S.M. Thesis, EE Dept.
June 1972

AD 744-206

TR-101 Dennis, Jack B.
On the Design and Specification of a Common Base Language
June 1972

AD 744-207

TR-102 Hossley, Robert F.
Finite Tree Automata and -Automata
S.M. Thesis, EE Dept.
September 1972

AD 749-367

TR-103 Sekino, Akira
Performance Evaluation of Multi-programmed Time-Shared Computer Systems
Ph.D Thesis, EE Dept.
September 1972

AD 749-949

TR-104 Schroeder, Michael D.
Cooperation of Mutually Suspicious Subsystems in a Computer Utility
Ph.D. Thesis, EE Dept.
September 1972
AD 750-173

TR-105 Smith, Burton J.
An Analysis of Sorting Networks
Sc.D. Thesis, EE Dept.
October 1972
AD 751-614

TR-106 Rackoff, Charles W.
The Emptiness and Complementation Problems for Automata on Infinite Trees
S.M. Thesis, EE Dept.
January 1973
AD 756-248

*TR-107 Madnick, Stuart E.
Storage Hierarchy Systems
Ph.D. Thesis, EE Dept.
April 1973
AD 760-001

TR-108 Wand, Mitchell
Mathematical Foundations of Formal Language Theory
Ph.D. Thesis, Math. Dept.
December 1973

TR-109 Johnson, David S.
Near-Optimal Bin Packing Algorithms
Ph.D. Thesis, Math. Dept.
June 1973
PB 222-090

TR-110 Moll, Robert
Complexity Classes of Recursive Functions
Ph.D. Thesis, Math. Dept.
June 1973

AD 767-730

TR-111 Linderman, John P.
Productivity in Parallel Computation Schemata
Ph.D. Thesis, EE Dept.
December 1973

PB 226-159/AS

TR-112 Hawryszkiewycz, Igor T.
Semantics of Data Base Systems
Ph.D. Thesis, EE Dept.
December 1973

PB 226-061/AS

TR-113 Herrmann, Paul P.
On Reducibility Among Combinatorial Problems
S.M. Thesis, Math. Dept.
December 1973

PB 226-157/AS

TR-114 Metcalfe, Robert M.
Packet Communication
Ph.D. Thesis, Applied Math., Harvard University
December 19right
AD 771-430

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

Project MAC Progress Report VI
July 1968-July 1969

AD 705-434

Project MAC Progress Report VII
July 1969-July 1970

AD 732-767

Project MAC Progress Report VIII
July 1970-July 1971

AD 735-148

* Project MAC Progress Report IX
July 1971-July 1972

AD 756-689

Project MAC Progress Report X
July 1972-July 1973

AD 771-428

Copies of all MAC reports listed in Publications may be secured from the National Technical Information Service, Operations Division, Springfield, Virginia, 22151. Prices vary. The AD number must be supplied with the request.

* Out of print, may be obtained from NTIS (see above).