AD-756 689

PROJECT MAC PROGRESS REPORT IX, JULY 1971 TO JULY 1972

Edward Fredkin

Massachusetts Institute of Technology

Prepared for:

Office of Naval Research
Advanced Research Projects Agency
Defense Supply Service Administration
Rome Air Development Center
National Science Foundation

February 1973

DISTRIBUTED BY:

NTIS

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

The work reported herein was carried out within Project MAC, an M.I.T. interdepartmental laboratory. Support was provided by:

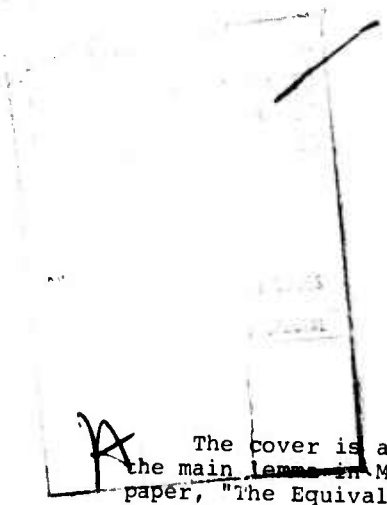
The Advanced Research Projects Agency of the Department of Defense, under Office of Naval Research Contracts: N00014-70-A-0362-0001, and N00014-70-A-0362-0004;

The Office of Naval Research, under contract N00014-69-A-0276-0002;

The National Science Foundation, under contract GJ00432;

The Defense Supply Service Administration, under contract DAHC-15-69-C-0347;

Rome Air Development Center, under contract F30602-72-C-0001.



The cover is an excerpt of the proof of the main lemma in Meyer and Stockmeyer's paper, "The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space".

The equations reveal how certain notations from automata theory (called regular expressions) can describe the behavior of Turing machines, from which it follows that the equivalence problem for regular expressions is inherently difficult to decide, no matter what procedure is used to decide equivalence.

AD 752 693

PROJECT MAC
PROGRESS REPORT IX
JULY 1971 to
JULY 1972

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

TABLE OF CONTENTS

UNCLASSIFIED
Security Classification

AD756689

DOCUMENT CONTROL DATA - R & D

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

ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
MASSACHUSETTS INSTITUTE OF TECHNOLOGY PROJECT MAC	UNCLASSIFIED 2b. GROUP NONE

3. REPORT TITLE

PROGRESS REPORT IX, JULY 1971 to JULY 1972

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

ANNUAL SCIENTIFIC REPORT

5. AUTHOR(S) (First name, middle initial, last name)

Collection of reports from Project MAC participants, edited by Prof. Edward Fredkin, Director of Project MAC

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
FEBRUARY, 1973	137	148

8a. CONTRACT OR GRANT NO.		9a. ORIGINATOR'S REPORT NUMBER(S)
N00014-70-A-0362-0001	(ARPA)	MAC PR IX
N00014-70-A-0362-0004	(ARPA)	
N00014-69-A-0276-0002	(ONR)	
GJ00432	(NSF)	
DAHC-15-69-C-0347	(DSSA)	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
F30602-72-C-0001	(RADC)	NONE

10. DISTRIBUTION STATEMENT

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	ADAVNCED RESEARCH PROJECTS AGENCY; OFFICE OF NAVAL RESEARCH; NATIONAL SCIENCE FOUNDATION; DEFENSE SUPPLY SERVICE ADMINISTRATION; ROME AIR DEVELOPMENT CENTER

13. ABSTRACT

The broad goal of Project MAC is investigation of new ways in which computers can aid people in their individual work.

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

DD FORM 1 NOV 65 1473 (PAGE 1)

S/N 0102-014-6600

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Artificial Intelligence						
Computer Networks						
Computation Structures						
Dynamic Modeling						
Graphics						
Hybrid Circuits						
Information Systems						
Interactive Management						
Machine-Aided Cognition						
Multiple-Access Computers						
On-Line Computers						
Programming Linguistics						
Real-Time Computers						
Theory of Automata						
Time-Sharing						

DD FORM 1 NOV 1473 (BACK)
(PAGE 2)

UNCLASSIFIED
Security Classification

TABLE OF CONTENTS (continued)

PROJECT MAC PERSONNEL

JULY 1971 to JULY 1972

Administration

Prof. E. Fredkin	Director
D. C. Scanlon	Administrative Officer
G. B. Walker	Business Manager
A. E. Egendorf	Director of Information Services
B. H. Kohl	Librarian

Academic Staff

Prof. F. J. Corbato	Prof. J. C. R. Licklider
Prof. J. B. Dennis	Prof. C. L. Liu
Prof. M. L. Dertouzos	Prof. W. A. Martin
Prof. J. J. Donovan	Prof. A. R. Meyer
Prof. R. M. Fano	Prof. J. Moses
Prof. E. Fredkin	Prof. N. P. Negroponte
Prof. G. A. Gorry	Prof. S. S. Patil
Prof. F. C. Hennie	Prof. J. H. Saltzer
Prof. C. Hewitt	Prof. J. Weizenbaum
Prof. M. M. Jones	

Instructors, Research Associates, Research Assistants and Others

J. Aiello	R. G. Bratt	R. Cohen
P. M. Allaman	D. Bricklin	B. K. Daniels
N. Amerasinghe	D. J. Brown	T. L. Davenport
Y. S. Auyang	G. P. Brown	T. M. Demchock
A. Bagchi	K. M. Brown	J. D. DeTreville
H. G. Baker	R. H. Brown	D. R. Dick
R. Barquin	S. S. Brown	A. R. Downing
C. G. Benedict	B. Carlson	A. C. England
P. B. Bishop	D. D. Clark	J. F. Farrell

Preceding page blank

Instructors, Research Associates, Research Assistants and Others (cont.)

R. J. Fateman
J. Ferrante
R. J. Fleischer
D. Folger
H. Forsdick
J. Fosseen
P. J. Fox
R. M. Fox
R. M. Frankston
F. Furtek
T. M. Gearing
M. R. Genesereth
W. Godfrey
L. I. Goodman
B. S. Greenberg
S. Gregory
I. Greif
F. E. Guertin
R. A. Gumpertz
R. M. Haas
M. Hack
M. M. Hammer
R. V. Harrington
G. Harris
I. T. Hawryszkiewycz
D. A. Henderson
R. F. Hossley
P. W. Hughett
D. H. Hunt
P. M. Hutchins
D. L. Isaman
P. Jessel
D. Johnson
J. W. Johnson
R. Johnston
J. Kaplan
P. A. Karger
D. J. Kfoury
P. A. King
M. Knaur
D. Koenig
J. Kok
S. Kruger
S. Kuo
R. S. Lamson
P. D. Lebling
R. Lefkowitz
B. Lester
J. P. Linderman
W. J. Long
J. B. Lotspiech
N. A. Lynch
C. Lynn
S. M. Macrakis
S. E. Madnick
C. Mah
R. Mandl
F. Manning
W. A. McCray
J. A. Meldman
M. S. Miller
P. L. Miller
R. N. Moll
E. J. Montes
D. A. Moon
R. C. Moore
M. L. Morgenstern
S. G. Morton
W. D. Northup
E. F. Okrent
L. S. Perrin
G. L. Peskin
G. F. Pfister
J. Piggins
K. T. Pogran
J. E. Qualitz
C. Rackoff
C. Ramchandani
D. P. Reed
K. G. Rhoads
E. C. Rosen
J. L. Rosenberg
L. J. Rotenberg
J. E. Rumbaugh

Instructors, Research Associates, Research Assistants and Others (cont.)

G. Ruth
J. Sabath
S. E. Saunders
L. J. Scheffler
M. D. Schroeder
J. I. Seiferas
A. Sekino
M. S. Seriff
W. G. Shaw
R. M. Seigel
J. R. Sloan
B. J. Smith
R. J. Steiger
J. A. Stern
R. A. Stern
J. R. Stinger
L. Stockmeyer
S. M. Stoney
J. R. Taggart
S. Tepper
R. C. Thurber
B. M. Trager
E. Tsiang
L. Tsien
S. R. Umarji
B. Vilfan
C. M. Vogt
P. S-H. Wang
S. A. Ward
R. W. Weissberg
J. Wish
R. T. Wong
F. T. Yao
C. Ying
D. Yun
S. Zaborowski
C. Ziering
R. F. Zippel

DSR Staff

M. J. Ablowitz
S. L. Alter
M. C. Amyot
E. R. Banks
A. K. Bhushan
E. H. Black
R. A. Bogen
M. F. Brescia
R. D. Bressler
M. S. Broos
A. L. Brown
B. Byer
H. O. Capps
M. A. Cohen
D. G. Cressey
J. S. D'Aversa
M. S. Draper
A. D. Egendorf
R. J. Feiertag
S. W. Galley
C. C. Garman
M. J. Ginzberg
R. P. Goldberg
J. P. Golden
R. C. Goldstein
P. M. Gunkel
R. J. Harman
J. F. Haverty
J. P. Jarvis
R. K. Kanodia
F. Y. Knight
B. H. Kohl
M. Lenot

DSR Staff (cont.)

M. I. Levin
R. F. Mabee
K. J. Martin
R. M. Metcalfe
E. W. Meyer, Jr.
J. C. Michener
N. I. Morris
J. Nievergelt
S. E. Niles
R. C. Owens
M. A. Padlipsky
L. G. Pantalone
S. G. Peltan
J. Phillips
C. L. Reeve
L. P. Rothschild
D. C. Scanlon
R. Schroeppel
T. Skinner
J. Spall
M. J. Spier
A. J. Strnad
A. Sunguroff
A. Vezza
V. L. Voydock
G. B. Walker
D. C. Watson
M. B. Weaver
S. H. Webber
D. M. Wells
J. L. White

Undergraduate Students

M. H. Alpert
Y. S. Auyang
G. G. Bajoria
G. G. Benedict
R. G. Bratt
D. Bricklin
H. R. Brodie
R. L. Brooks
K. M. Brown
R. H. Brown
S. S. Brown
J. L. Caruso
A. Y. Chan
D. J. Chang
S. S. Cohen
C. C. Conklin
R. G. Curley
S. E. Cutler
T. L. Davenport
T. M. Demchock
D. R. Dick
A. R. Downing
R. Elkin
R. A. Freedman
D. E. Geer
M. R. Genesereth
W. Godfrey
P. A. Green
R. A. Guida
P. H. Guldberg
R. H. Gumpertz
C. A. Hannah
R. V. Harrington
G. Harris
J. H. Harris
M. D. Horowitz
W. F. Hui
E. Kant
P. A. Karger
R. M. Katz
C. A. Kessel
H. J. Kim
R. N. King
N. V. Kohn
J. Kok
D. M. Krackhardt
R. S. Lamson
C. K. Leung

Undergraduate Students (cont.)

S. M. Macrakis
W. S. Mark
D. Misunas
D. A. Moon
W. Y. Ng
B. Niamir
G. Pavel
L. S. Perrin
G. L. Peskin
J. Phillips
R. L. Prakken
D. P. Reed
K. G. Rhoads
E. C. Rosen
L. M. Rubin
N. D. Ryan
S. Sadeq
S. E. Saunders
J. C. Schaffert
R. M. Seigel
H. J. Seigel
S. M. Stoney
R. Swift
J. D. Sybalsky
C. D. Tavares
B. M. Trager
E. Tsiang
M. E. Wolfe
R. E. Zippel

Support Staff

M. E. Baker
S. A. Bankole-Wright
M. A. Bizot
M. S. Broos
G. W. Brown
O. D. Carey
L. S. Cavallaro
M. T. Cheney
S. J. Cohn
M. J. Connell
M. Cummings
S. Daise
J. A. Darcy
B. Doyle
C. P. Doyle
C. T. Falls
L. L. Gammell
A. M. Garrity
J. A. Haley
L. J. Haron
A. J. Hicks
R. F. Hill
D. L. Jones
D. Kontrimus
J. S. Lague
E. Y. Lewis
E. T. Moore
B. A. Morneault
E. F. Nangle
K. W. Pierce
S. Pitkin
N. J. Robinson
E. M. Roderick
A. Rubin
A. C. Simmons
K. K. Simpson
A. H. Speare
A. G. Testa
M. F. Webber
L. E. Yaple

Guests

P. Azema
Prof. J. Berger
Prof. J. I. Elkind
A. Endo
Prof. M. Greenberger
Prof. K. Ikeda
M. Miyazaki
L. Priese

PREFACE

Project MAC was established in 1963 as an interdepartmental laboratory at the Massachusetts Institute of Technology, to do research in Multiple Access Computer Systems and Machine Aided Cognition. This effort resulted in the development of the CTSS and Multics Systems. Project MAC is currently moving toward a major effort in Automatic Programming.

During the year ending June 30, 1972, there were 325 persons associated with Project MAC. They included: 21 faculty members, mainly from the Department of Electrical Engineering, Department of Mathematics and from the Alfred P. Sloan School of Management; 103 staff members, (DSR Staff and Support Staff), 195 students, (both Undergraduate and Graduate), and 8 guests.

This year, through extensive discussions and consultations both within and outside of M.I.T., Project MAC arrived at its new focus on Automatic Programming. One effort in Automatic Programming involves a system, which has embedded in it, extensive knowledge of the subject for which the programming is being automated. The other effort is based on making it easier to build large programs from simplier programs. The first effort is being carried out in the newly formed Automatic Programming Group while the second approach is being carried out in the Dynamic Modeling Group.

The Automatic Programming Group plans to construct Protosystems of increasing complexity to gain experience in embedding knowledge into systems. This task requires new programming techniques and languages. Two new languages called PLANNER and CONNIVER, which have come out of the Artificial Intelligence efforts at M.I.T., seem the most promising languages for embedding knowledge into systems. Development and implementation of these languages is in progress.

The Mathematical Laboratory System, MACSYMA, can now carry out many symbolic manipulations previously considered very difficult. It can, for example, factor polynomials in several variables and do so very efficiently.

As the Multics development effort has tapered off, the Computer Systems Research Group has shifted its attention to security and protection in Computer Systems. In conjunction with Honeywell, Inc., new follow-on hardware for Multics was specified, which is especially tailored to make Multics secure and efficient. This will be in operation early in 1973.

The Computation Structures Group has continued its investigation into base languages, parallel processing schemata and the means for description and realization of digital systems. The Automata Theory Group has now focused its attention toward investigating the complexity of algorithms.

Preceding page blank

During the past year, the basic program of Project MAC was supported by the Information Processing Techniques Directorate of the Advanced Research Projects Agency (ARPA). Individual projects were funded by several other agencies; Dynamic Modeling by the Behavioral Sciences Directorate of ARPA; Programming Generality by the National Science Foundation; and the implementation of PLANNER by the Office of Naval Research.

AUTOMATIC PROGRAMMING

Prof. W. A. Martin

Instructors, Research Associates, Research Assistants and Others

J. D. DeTreville
A. C. England
L. I. Goodman
M. L. Morgenstern
W. D. Northup
J. L. Rosenberg
G. Ruth

Undergraduate Students

W. S. Mark
B. Niamir

DSR Staff

S. L. Alter
E. R. Banks
J. S. D'Aversa
M. J. Ginzberg
J. P. Jarvis
A. Sunguroff

Support Staff

B. Doyle
J. S. Lague

AUTOMATIC PROGRAMMING

The automatic programming group has been formed to investigate the generation of programs from descriptions of the actions the programs are to perform, rather than from a description of how these actions are to be performed. To do this, program generating programs must have a knowledge of possible methods which could be used to perform the actions. They must also know how to select methods appropriate to a given problem.

Current day compilers are examples of program generating programs. They have a knowledge of machine structure and they attempt to find the best series of register operations to perform a desired result such as the multiplication of a series of numbers. To do this requires some search, but primarily it is a matter of incorporating into the compilers good techniques for the situations which are known to occur.

In order to extend the program writing capabilities of programs beyond what current compilers can do, it will be necessary to incorporate into them knowledge about specific problem areas. As with compilers, the problem is to represent this knowledge in the machine in a way that it can be employed with very little search.

The group has chosen to center its initial investigation in the area of management-information systems. This is because a) the automation of programming in this area would be of great practical importance, b) much can be done by solving problems of data and file structures, searching, sorting, and scheduling which are quasi-universal in programming c) the area provides a good spectrum of problems of increasing difficulty. A prototype system is being constructed and should be yielding initial results in a few months. The system contains two major components: the first is for the interactive specification of what logical information processing is to be done, the second is for the automatic realization of this processing on a specific computing system.

Automatic realization of the processing requires the design of a series of data files and programs which do the specified processing at minimum cost. Two methods of measuring cost seem plausible for investigation: the charging scheme supplied with the IBM OS/360 MVT operating system and the evaluation procedure used by the SCERT computer systems simulation package. The IBM scheme is simpler and is being incorporated into the initial prototype system.

Generation of the files and programs has been broken into two phases. The first phase constrains the files and programs to a specific design. The second phase generates PL/I and JCL to realize this design on an IBM/370 computer. Three methods of implementing the first phase are under investigation. The first method is to provide a series of commands so that a user can design the files and programs interactively and then estimate the cost of competing designs. The system checks that all of the user's design decisions are feasible and consistent. The user is also able to ask questions about a partial design. The second method involves the use of search and heuristics to generate a design automatically. The heuristics satisfy the high volume processing requirements first and rely on the

Preceding page blank

assumption that a costly partial design does not often lead to the best total design. The third method is to formulate the problem as an integer programming problem and to seek an optimal solution through sophisticated integer programming techniques. It is possible that optional solutions will take much longer to find than good solutions, without being appreciably better. However, optimal solutions to carefully chosen problems should help us to understand the performance of our heuristic routines.

Our current thoughts are that the interactive design of the processing to be done will proceed in two phases. The first phase will be the discusson with knowledgeable routines of the particular ways certain standard problems such as inventory control are to be handled. In the second phase the system will apply these methods to generate an information system for a particular configuration of physical assets operating with a specified corporate plan. During this phase specific problems may require further interactive design.

In addition to this central thrust, the group is also exploring related questions such as the analysis of programs and sponsoring the development of LISP on MULTICS.

Publications

1. Fateman, R. J., Essays in Algebraic Simplification, Project MAC, M.I.T., MAC-TR-95, April 1972, AD 740-132.

2. Fateman, R. J., "Rationally Simplifying Non-Rational Expressions", SIGSAM Bulletin, No. 23, July 1972, pp. 8-9.

3. Martin, W. A., "Determining the Equivalence of Algebraic Expressions by Hash Coding", Journal of the ACM, Vol. 8, No. 4, October 1971, pp. 549-558.

4. Martin, W. A., "Sorting", Computing Surveys, Vol. 3, No. 4, December 1971, pp. 147-174.

5. Martin, W. A., and D. N. Ness, "Optimizing Binary Trees Grown with a Sorting Algorithm", Communications of the ACM, Vol. 15, No. 2, 1972, pp. 88-93.

6. Moses, J., "Algebraic Simplification: A Guide for the Perplexed", Communications of the ACM, Vol. 14, No. 8, August 1971, pp. 527-537.

7. Moses, J., "Symbolic Integration: The Stormy Decade", Communications of the ACM, Vol. 14, No. 8, August 1971, pp. 548-560.

8. Moses, J., "Toward a General Theory of Special Functions", Communicatiors of the ACM, Vol. 15, No. 7, July 1972, pp. 550-554.

9. Wang, P. S., Evaluation of Definite Integrals by Symbolic Manipulation, Ph.D. Thesis, Department of Mathematics, M.I.T., 1971; also MAC-TR-92, October 1971, AD 732-005.

10. Wang, P. S., "Application of MACSYMA to an Asymptotic Expansion Problem", Proceedings of the ACM 25th Annual Conference, August 1972, pp. 844-850.

Publications In Progress

1. Fateman, R. J., and J. Moses, "Canonical Forms for First Order Exponential Expressions".

2. Fateman, R. J., "Comments on Problem 2".

3. Fateman, R. J., "On the Computation of Powers of Polynomials".

4. Wang, P. S., "Factoring Multivariate Polynomials Over the Integers".

5. Yun, D. Y. Y., "An Application of MACSYMA to Proving the Achievability of the $\lfloor \frac{n+1}{2} \rfloor M$ for Evaluation of

Publications In Progress (continued)

General Non-monic Polynomials of Degree n".

6. Yun, D. Y. Y., "On the Efficiency of the Dijkstra Algorithm".

7. Yun, D. Y. Y., "On Symbolic Solutions of Systems of Algebraic Equations".

Talks

1. Moses, J., "Algebraic Manipulation", a set of lectures given at the Computer Science Seminar, Bonn, Germany, August 1971.

2. Moses, J., "Algebraic Manipulation", given at the Joint Colloquium of the Departments of Mathematics and Computers & Communications, University of Michigan, Ann Arbor, Michigan, November 1971.

3. Wang, P., "Factoring Multivariate Polynomials", given at the Department of Applied Mathematics, Taiwan National Chung Hsing University at Taichung, Taiwan, June 1972.

COMPUTATION STRUCTURES

Prof. J. B. Dennis

Academic Staff

Prof. R. M. Fano

Prof. S. S. Patil

Instructors, Research Associates, Research Assistants and Others

N. Amerasinghe
H. G. Baker
R. Barquin
P. B. Bishop
R. Cohen
J. Fosseen
P. J. Fox
F. Furtek
T. M. Gearing
I. Greif
M. Hack
I. T. Hawryszkiewycz
D. A. Henderson
B. Lester
J. P. Linderman
J. B. Lotspiech
J. A. Meldman
J. E. Qualitz
C. Ramchandani
L. J. Rotenberg
J. E. Rumbaugh
R. J. Steiger
S. R. Umarji

Undergraduate Students

G. G. Bajoria
H. J. Kim
C. K. Leung
D. Misunas
J. Phillips
S. Sadeq
J. C. Schaffert
R. Swift

DSR Staff

J. Nievergelt

Support Staff

B. A. Morneault

A. Rubin

COMPUTATION STRUCTURES

The Computation Structures Group is concerned with the analysis of fundamental issues arising in the design and construction of general-purpose computer systems through the formulation and study of appropriate abstract models. The past year has seen new developments in the theory and application of Petri nets as a model of systems of interacting parts, improved techniques for realizing digital systems with assurance of correct operation, development of the theory of data flow schemata, and contributions to the study of program correctness and programming generality.

A. Petri Nets

Our research relating to Petri nets is concerned with the theory of Petri nets, the relation of nets to logic circuits and asynchronous modular systems, and the use of Petri nets as a model for the behavior of systems of interacting parts, including systems within and outside the domain of computer science.

Timed Petri Nets

Chander Ramchandani is investigating the use of Petri net models in the performance analysis of systems. Petri nets (8, 5) are an attractive model for studies of system performance because the important interactions between system parts are easily represented. Petri nets represent the ordering relationship of events in a system that mark the initiation and termination of activities, but do not represent the timing of events or durations of activities. For performance analysis the Petri-net model of a system must be augmented with timing information.

In a Petri net (Figure 1), the firing of a transition may represent an interval of activity by some system part. If the transition is enabled (at least one token in each of its input places) it means that activity of the system part may begin. We associate initiation of activity with picking up one token from each input place, and termination of activity with adding one token to each output place. This corresponds to considering the transition to be two transitions and a place p as in Figure 2.

Figure 3 shows a timed Petri net obtained by associating time parameters with certain transitions of the net in Figure 1. In a timed net transitions without time parameters represent sequencing constraints on activities as in a conventional Petri net. Action of a timed transition may be explained in terms of Figure 2, where the time parameter $\tau(t)$ is associated with place p. Transition t' may fire immediately when enabled or any time later (providing it remains enabled). Then transition t" becomes enabled and fires exactly $\tau(t)$ time units after the firing of t'. Thus the firing of transitions t' and t" represents initiation and termination of one instance of the activity represented by transition t. It is possible for a transition t in a timed net to be re-enabled before a previously initiated instance of the associated activity has terminated. In fact, many instances of the activity may be in

Preceding page blank

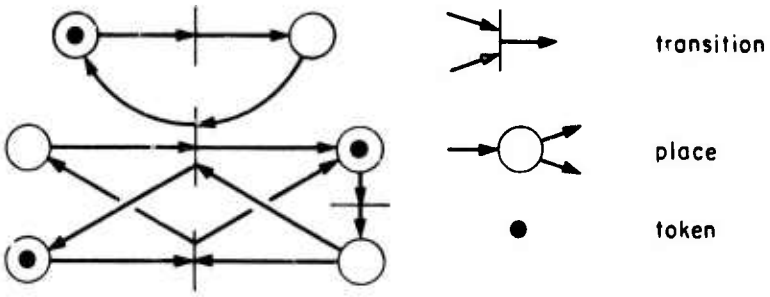


Figure I. A Petri net.

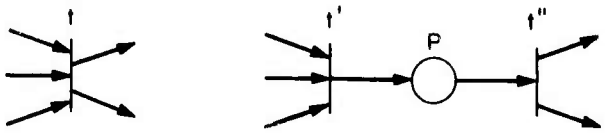


Figure 2. Meaning of a timed transition.

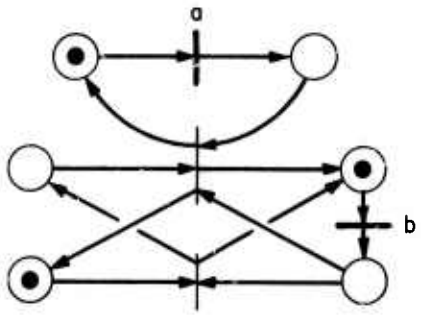
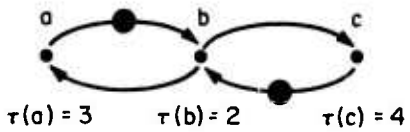


Figure 3. A timed Petri net.

(a) a timed marked graph



(b) periadic schedule

transition a: 2-5, 8-11, 14-17, ...
b: 0-2, 6-8, 12-14, ...
c: 2-6, 8-12, 14-18, ...

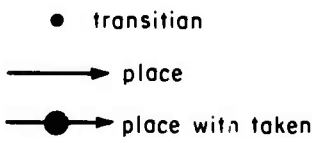


Figure 4. Periadic schedule for a timed marked graph.

progress simultaneously, as we shall see in later examples. The number of tokens in place p is the current number of simultaneous instances of the activity.

A schedule for a timed Petri net is a set of sequences of initiation and termination times for the timed transitions of the net. A schedule is feasible if the timed net can exhibit the behavior specified by the schedule. A schedule is not feasible if it calls for initiation of an activity earlier than allowed by terminations of other activities. A feasible schedule is said to be prompt if each activity always initiates as early as possible. Here are examples of feasible and prompt schedules for the timed net and initial token distribution shown in Figure 3:

(a) a feasible schedule
transition a: 0-5, 8-13, 13-18
b: 2-4, 4-6, 9-11, 11-13, 15-17

(b) a prompt schedule
transition a: 0-5, 5-10, 10-15
b: 0-2, 2-4, 5-7, 7-9, 10-12

Every timed net for which the underlying Petri net is persistent (no transition ceases being enabled except by firing) has a unique prompt schedule.

We have studied the class of Petri nets known as marked graphs. In a marked graph, each place is an input place of at most one transition, and an output place of at most one transition. All transitions of a marked graph fire equally many times in any behavior that returns the net to its original configuration. In consequence, a prompt schedule for a timed graph is periodic in that each timed transition initiates at regular intervals. The example in Figure 4 has a periodic prompt schedule with period six. In this case, the rate of firing is determined by the circuit containing transitions b and c.

Figure 5 illustrates a situation where several instances of an activity represented by transition b may proceed concurrently. Instances of the activity represented by transition a are forced to occur strictly in sequence by the one-token self loop. The prompt schedule shown has a period of eight.

The computation rate of a timed marked graph is the average rate of firing for any transition of the graph in a prompt schedule. For the example in Figure 4 the rate is 1/6; for Figure 5, the rate is 1/4.

There is a simple algorithm for determining the computation rate of a timed marked graph. Let the vertices (transitions) and arcs (places) of a strongly connected marked graph be

$$V = \{v_1, \ldots, v_\ell\}$$

$$A = \{a_1, \ldots, a_p\}$$

where an arc $a_m = (v_i, v_j)$ is directed from transition v_i to v_j, and let τ_i be the time associated with transition v_i ($\tau_i = 0$ if v_i is not a timed transition). For any strongly connected marked graph one can find a set of simple circuits $C_1, \ldots, C_m$ that cover all arcs of the graph (5). Let M_{ij} be the number of tokens on arc (v_i, v_j) in the initial marking of the net. Then the computation rate ρ of the timed marked graph is given by

$$\rho = \min \left\{ \frac{N_k}{T_k} \,\middle|\, k = 1, \ldots, m \right\}$$

where

$$T_k = \sum_{v_i \in C_k} \tau_i$$

is the sum of the times associated with transitions of circuit C_k and

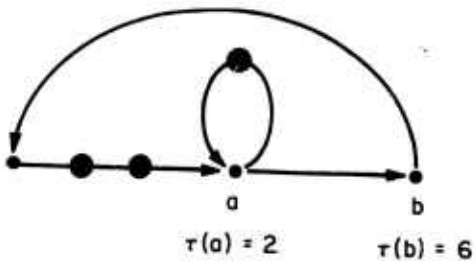
$$N_k = \sum_{(v_i, v_j) \in C_k} M_{ij}$$

is the number of tokens on arcs of circuit C_k.

Figure 6 shows a "PERT" chart with activities a,b,c,d,e and the corresponding timed marked graph. Application of the foregoing procedure shows that the computation rate is 1/8, the reciprocal of the time for the critical path. We may ask what happens to the computation rate if N_P processors are permitted to perform activities concurrently. The corresponding marked graph is shown in Figure 7, where it is assumed that only N_R instances of activity e are permitted at one time, but arbitrarily many instances are possible for the other activities. The figure gives the computation rates for several values of N_P and N_R.

Work is continuing on performance analysis of systems represented by more general classes of Petri nets. Also, the properties of Petri nets having time bounds or statistical

(a)

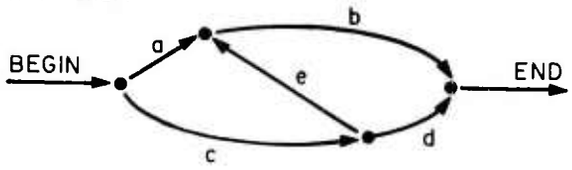


(b)

transition a: 0–2, 2–4, 8–10, 10–12,...
b: 2–8, 4–10, 10–16, 12–18,...

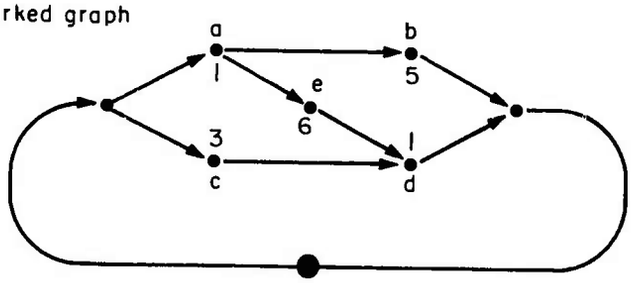
Figure 5. Marked graph with concurrent instances of an activity.

(a) "PERT" chart



$\tau(a) = 1$
$\tau(b) = 5$
$\tau(c) = 3$
$\tau(d) = 1$
$\tau(e) = 6$

(b) marked graph



circuit	N_k	T_k
ab	1	6
cd	1	4
aed	1	8

Figure 6. Computation rate of a timed marked graph

distributions associated with transitions are being studied.

Canonic Forms for Petri Nets

We have begun investigation of notions of equivalence and canonic forms for Petri nets. For the special case of marked graphs, Henry Baker (2) has shown how to reduce any marked graph to a simple form which is the same for all marked graphs equivalent to the given marked graph.

Suppose G is a marked graph and N is some subset of the transitions of G. Then if ω is a firing sequence of G, the corresponding derived firing sequence ω_N is obtained from ω by erasing all elements that are not members of N. Let G and G' be marked graphs and let $N = \{t_1, \ldots, t_n\}$ be a set of n transitions that appear in both G and G'. We say that G and G' are equivalent with respect to N if for each firing sequence ω of G there is a firing sequence ω' of G' such that ω_N and ω'_N are identical, and vice versa. The two marked graphs in Figure 8 are equivalent with respect to N = {a,b} since in each case the set of derived firing sequences is $(ab \cup ba)^*$.

First we give two rules which when applied to any marked graph will give a simpler marked graph equivalent to the original with respect to all of its transitions:

Rule 1: If an arc originates and terminates on the same transition, and has at least one token, it may be deleted.

Rule 2: Let a and b be any two distinct transitions, and let x be an arc from a to b. If the number of tokens on arc x is greater than or equal to the total number of tokens on the arcs of any other simple, directed path from a to b, then arc x may be deleted.

Use of the two rules is illustrated in Figure 9. Rule 1 is used to remove arc 1, and rule 2 is used to delete arcs 2, 3 and 4. For each of the three marked graphs, the firing sequences are all prefixes of the infinite string $(abc)^\infty$. A marked graph for which no applications of the two rules are possible is called a minimal-arc marked graph.

The minimal arc form of a marked graph always has the same set of firing sequences as the original marked graph. Furthermore, any pair of marked graphs that are equivalent with respect to a one-to-one correspondence of their transitions have the same minimal-arc form. Thus the minimal arc form is canonic for these marked graphs.

Now suppose N is a set of n transitions common to two marked graphs G and G'. How can we tell whether G and G' are equivalent with respect to N? It turns out that if G is a live marked graph, it may be reduced to an n-transition marked graph equivalent to G with respect to N. This is done by carrying

out the steps below for each transition t of G that is not a member of N:

Step 1: Delete any arcs that originate and terminate at transition t. If any such arc has no token, the marked graph is not live.

Step 2: Let $X = \{x_1, \ldots, x_m\}$ be the set of input arcs and $Y = \{y_1, \ldots, y_n\}$ the set of output arcs of transition t. Let M_i be the number of tokens on arc x_i and let N_j be the number of tokens on arc y_j.

Step 3: Replace transition t and the arcs in $X \cup Y$ with the arcs

$$\{z_{ij} | i = 1, \ldots, m; \quad j = 1, \ldots, n\}$$

where z_{ij} originates on the same transition as x_i and terminates on the same transition as y_j. Put $M_i + N_j$ tokens on arc z_{ij}.

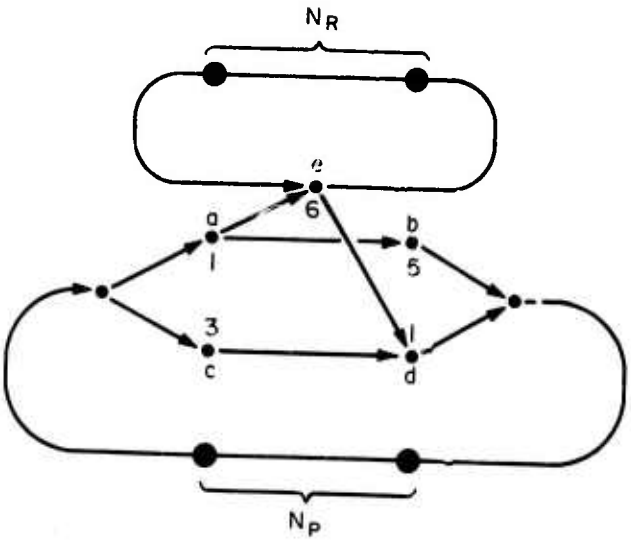
Applying this procedure to either marked graph in Figure 8 gives the canonic form in Figure 10. This example shows that the canonical form for a safe marked graph (5) is not necessarily safe.

B. Arbiters

Arbiters are fundamental units of digital systems that are required whenever two or more asynchronous activities compete for access to a shared unit or resource. A basic form of arbiter known as an elementary arbiter is illustrated in Figure 11. It controls access to a shared resource by two users -- user 1 and user 2. A 0-to-1 transition on either one of the request wires is a signal that the corresponding user desires access to the shared resource. In the absence of a competing request from the other user the arbiter must promptly produce a 0-to-1 transition on the corresponding grant wire. The user signals completion of his use of the resource by a 1-to-0 transition on the request wire, whereupon the arbiter must respond with a 1-to-0 transition on the grant wire. If requests arrive nearly simultaneously from both users, the arbiter must promptly and unambiguously grant either one of the requests and delay granting the second request until the resource is freed. Correct operation of an elementary arbiter must satisfy these conditions:

1. It must never occur that both grant wires are simultaneously at level 1.

2. If both grant wires are at 0 and at least one of the request wires is at 1, the arbiter must grant one of requests.



N_P

N_R	1	2	3	4
1	1/8	1/6	1/6	1/6
2	1/8	1/4	1/3	1/3
3	1/8	1/4	3/8	1/2

Figure 7. Timed marked graph representing several processors and limited throughput of one activity.

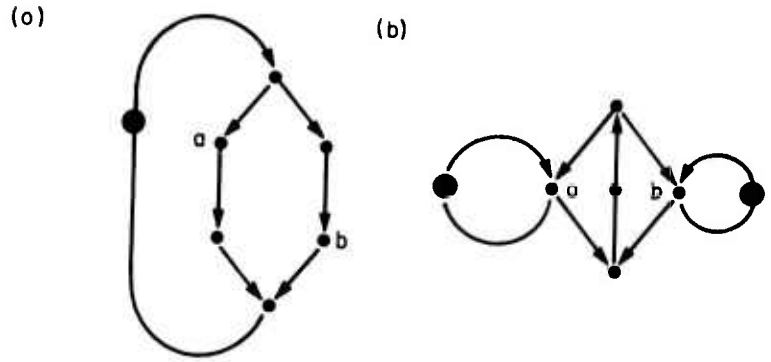
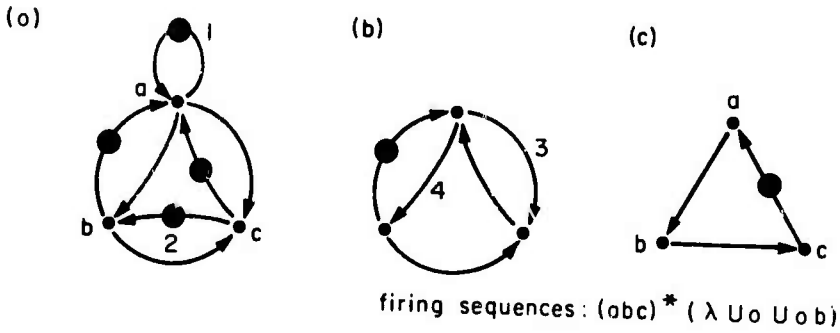


Figure 8. Two equivolent marked grophs.



firing sequences : $(abc)^* (\lambda \cup a \cup a b)$

Figure 9. Simplificotion of a morked groph.

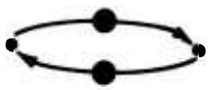


Figure 10. Canonic form for the marked graphs in Figure 8.

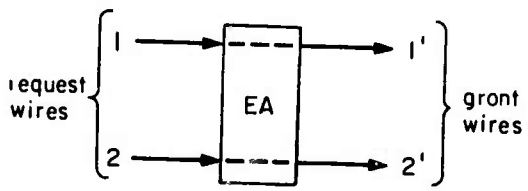


Figure 11. The elementory orbiter.

We have found that any requirement for arbitration in asynchronous digital systems can be met by a modular subsystem using elementary arbiters. For example, an arbiter that oversees sharing of a resource by n users can be built using a binary tree of elementary arbiters (14). The case of n users and m servers has been studied thoroughly by Patil, and he has recently devised an improved solution based on n-user and m-user arbiters (13).

Designing an elementary arbiter that functions correctly and always acts within a specified time interval is a difficult problem. When the two request wires make 0-to-1 transitions nearly simultaneously, the arbiter may make an arbitrary choice, but it must do so without hesitation, and without the appearance of spurious signals on the grant wires.

Suhas Patil has devised an elegant scheme for building an elementary arbiter that will operate correctly in a fixed time with extremely small probability of error. This scheme makes use of a subunit called finite resolution arbiter (FRA) and illustrated in Figure 12. An FRA can fail to operate correctly only if two request signals arrive with a separation of δ time units or less. If an FRA fails, the result is that both grant wires switch to 1.

Now consider a pair of FRA's connected in cascade as in Figure 13a. If two requests arrive at FRA-1 separated by more than δ time units, only one of the request signals will reach FRA-2 and operation will be completed correctly. If requests arrive at FRA-1 with less than δ time units separation, then FRA-1 will transmit both grant signals. Assume for the moment that the two request signals are delayed equally by FRA-1. Then, so long as $\Delta > 2\delta$, the requests arriving at FRA-2 will be separated by more than δ time units and FRA-2 will grant one and only one of the requests.

One of several possible cricuits for a finite resolution arbiter is shown in Figure 14. Each pair of NAND gates forms a set-reset flip flop which is forced into its 1 state by the presence of a request on the associated input wire. The setting of one flip flop prevents the other flip flop from being set, thereby blocking its associated request. If two requests arrive at nearly the same instant, both flip flops will be set since neither will be fast enough to block the other.

The time interval δ is the time separation of request signals such that a request signal and a block signal arrive simultaneously at one of the flip flops. In this circumstance the flip flop may be placed in a metastable state in which it may remain for an arbitrarily long time (with decreasing probability). The existence of metastable states, and the certainty that failures caused by circuits persisting in metastable states have been problematic in computer systems has been nicely explained by Ornstein (4).

It is reasonable to model flip flop behavior for critical input timing as follows: If the set and reset inputs become 1 with time separation less than some small fixed interval ε, the flop flop enters its metastable state, and the probability of commitment to one of the stable states during an interval dt after elapsed time τ is $P(\tau)dt = (1/T)e^{-\tau/T}dt$ where T is a characteristic time of the flip flop. An exponential density function $P(\tau)$ is used because we expect that the probability of commitment during any interval, given that commitment has not occurred earlier, is independent of the elapsed time.

From Figure 15 we see that the cascade of two FRA's is not perfect; it can fail if request 2 and a block signal generated by request 1 occur simultaneously. The probability of failure is very small if $\Delta >> 2\delta$ and decreases exponentially as Δ is made larger. Moreover, the probability of failure may be mas as small as desired by adding further FRA's in cascade, as in Figure 13b.

Note that, while it appears impossible to design a perfect elementary arbiter that always operates within a fixed time, one can modify the FRA circuit so that each flip flop will respond with grant and block signals only when it is committed to a stable state. In this way a perfect elementary arbiter may be constructed which may require an arbitrarily long time (exponentially distributed) to respond.

C. Computation Schemata

Our research in the theory of computation schemata has the goal of reaching a better understanding of good representations for algorithms -- representations in terms of which determinacy of an algorithm may be readily determined or guaranteed; forms suitable for deriving optimum machine code, or for identifying concurrently executable parts; schemes of representation for which the meaning is readily apparent to the programmer.

We have studied two sorts of parallel computation schemata that model programs and systems involving concurrent transformations and tests on unstructured values. On one hand we have developed a refinement of the parallel program schemata of Karp and Miller (9) and have investigated issues raised by the refined model. In this model the flow of data in a program or system is modeled separately from the sequencing or control. On the other hand are models like the program graphs studied by Rodriguez (16) in which the data flow and control specifications are combined in a single graph. Further development of the ideas of Rodriguez has led to the study of data flow schemata. Recent results from these two directions of research on parallel schemata are reviewed below.

Productivity in Parallel Schemata

In a computation schema it may be that certain actions occurring during a computation have no effect on any output value produced by the computation. In this case we say that

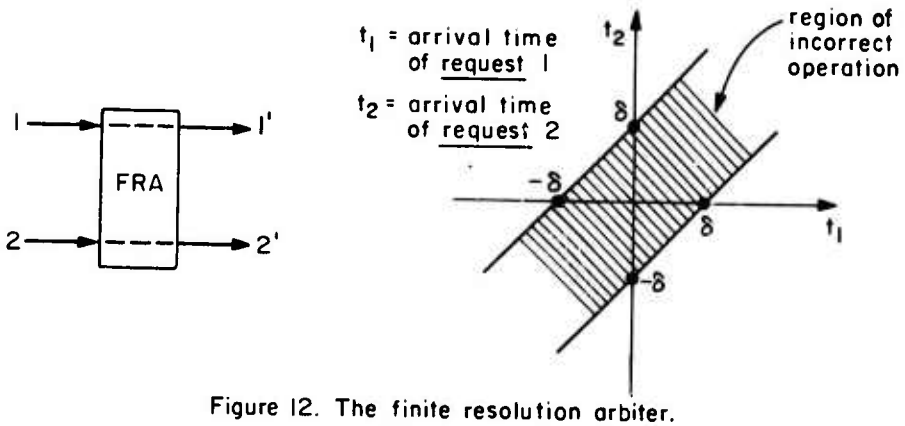
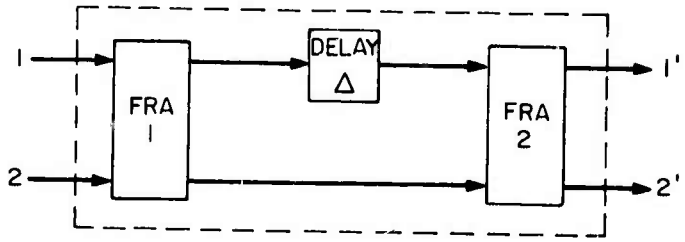


Figure 12. The finite resolution arbiter.

(a)



(b)

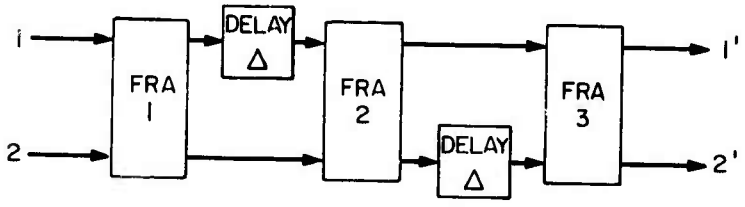


Figure 13. Finite resolution arbiters connected in cascade.

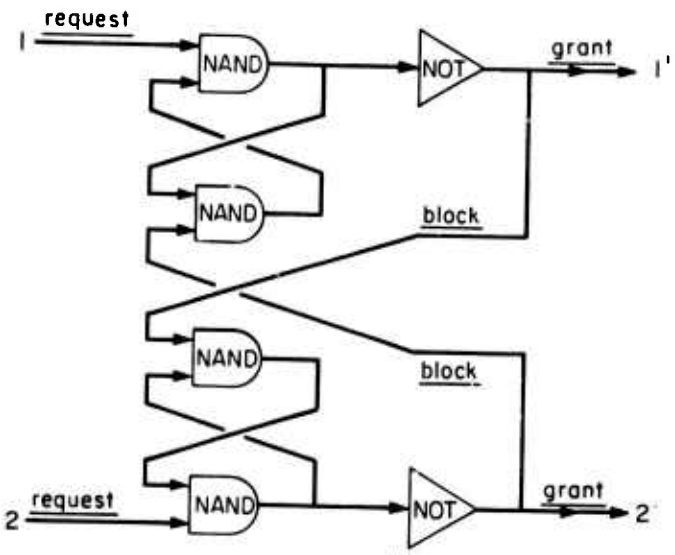


Figure 14. Circuit for finite resolution orbiter.

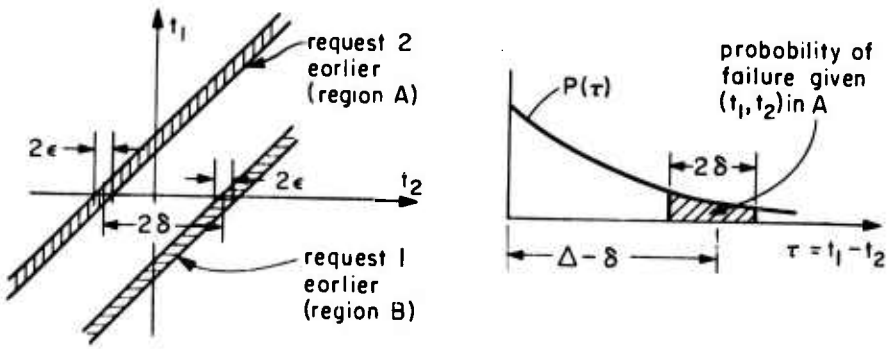


Figure 15. Foilure onolysis for two FRA's in cascode.

these actions are not productive. We have found there is a trade-off in parallel schemata between productivity and degree of concurrency. That is, to achieve maximum parallelism, it is necessary that the possibility of nonproductive actions be introduced. John Linderman has studied this matter for a class of computation schemata closely related to the parallel program schemata of Karp and Miller (9), and the flow-graph schemata of Slutz (17).

These schemata have separate parts to represent the communication paths for data and the sequencing of actions by operators and decision elements. Since the distinction between "transformations" and "tests" is so pervasive in programming, we feel they should be modeled as different fundamental actions in computation schemata. For this reason, our data flow graphs contain both operators, which model elements that transform values, and deciders, which perform tests with true/false outcomes. Associated with each operator is a function letter, and with each decider a predicate letter. Specific functions and predicates are assigned to the function and predicate letters by an interpretation of the schema. In this way, several operators may be required to perform the same transformation -- or several deciders, the same predicate -- in any interpretation of the schema. This departure from the Karp-Miller model permits treatment of determinacy and equivalence for a broader range of programs and systems.

Each operator and decider has associated initiation and termination events. When an operator or decider initiates, values are read from its ordered set of input memory cells and this vector of values is, in effect, entered into a first-in-first-out queue. Thus multiple initiations of an operator or decider may occur without intervening terminations. When an operator terminates, it writes into its output memory cells the values obtained by applying the function denoted by its function letter to the vector of values taken from the head of the queue. For each decider there are two termination events corresponding to the true and false outcomes of applying the predicate denoted by its predicate letter to the vector of values at the head of its queue.

When and if these events can happen is specified by the control of the schema. A variety of explicit mechanisms have been used to represent the control, including finite state machines, precedence graphs, and Petri nets. These mechanisms share the property that they specify which sequences of events are allowed and which are not allowed as possible behaviors of a schema. The allowed sequences of events are called the control sequences of the schema. Study of various control mechanisms has shown that certain properties of control sequences -- persistence, commutativity, conflict freedom, and repetition freedom -- are central to the study of equivalence, determinacy, parallelism and productivity, regardless of the mechanism used to specify the set of control sequences. For this reason we have studied these properties of schemata without regard to the mechanism used to specify the set of control sequences.

Consider the program below in which w and x are input variables and y and z are output variables:

```
begin
     y := f(g(w))
     if p(w,x) then z := g(f(w)) else z := h(f(w))
end
```

Two schemata for this program are shown in Figure 16. To be definitive, the control sets have been specified by Petri nets. Examples of control sequences for S_1 include

$$\bar{a}\ \bar{t}\ \underline{a}\ \bar{c}\ \bar{b}\ t^T\ \underline{b}\ \bar{d}\ \underline{d}\ \underline{c}$$

$$\bar{t}\ \bar{b}\ t^F\ \underline{b}\ \bar{a}\ \underline{b}\ \underline{a}\ \bar{c}\ \bar{e}\ \underline{e}\ \underline{c}$$

in which overbars and underbars indicate initiation and termination, and the superscript T or F refers to the outcome of a decider.

We identify certain memory cells of a schema as an ordered set of input cells and an ordered set of output cells. Then we may discuss equivalence of two schemata in terms of producing the same output values when given identical inputs. In Figure 16, w and x are the input cells and y and z the output cells of both S_1 and S_2. It is easy to see that, in either schema, any allowed sequence will assign the same values to cells y and z as are produced by the program. Hence both schemata are "functionally determinate" and are equivalent with respect to the specified input and output cells.

In these schemata, an issue arises that is not present when every termination event puts a value in some memory cell and all cell histories affect the question of equivalence, as in the Karp-Miller theory. It is now possible for operators and deciders to be invloved in "useless activity." For example, if y were not an output cell of schema S_1 or S_2, operators a and c would not be productive. Similarly if the same sequence of actions followed either outcome of a decider, then that action of the decider would not be productive.

The precise formulation of this notion of productivity requires formalisms we do not wish to develop here, but the central idea is fairly straightforward. A use of an operator in a control sequence is productive if subsequent actions by operators "carry its result" to a schema output cell or to a productive decider. Since an action by a decider does not directly affect contents of memory cells, determining its productivity is not as easy. We consider a use of a decider to be productive if the schema has two control sequences that define inequivalent computations, and are in "disagreement about decider outcomes" only at the given decider use. For example, consider the program

(a) schema S_1 :

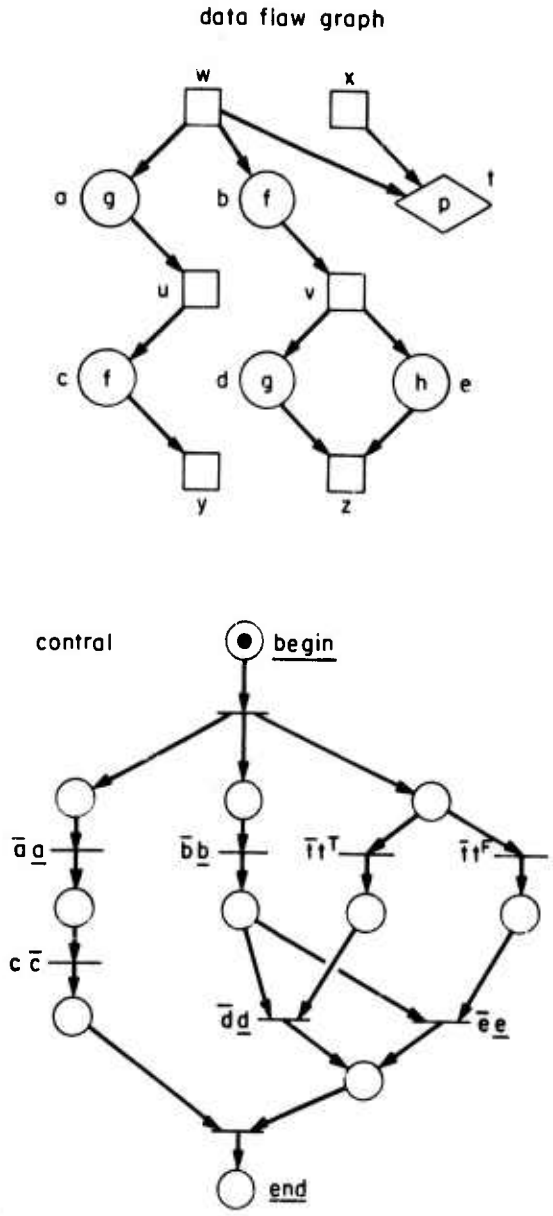


Figure 16. Two equivalent computation schemata.

(b) schema S_2 :

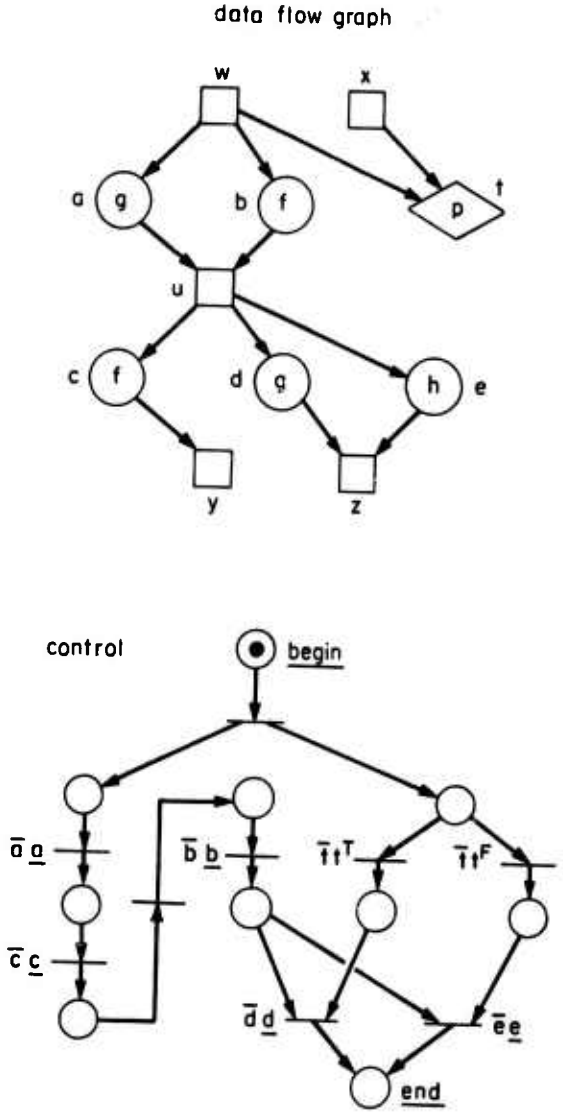


Figure 16. (Continued).

```
begin
    if p(x) then
        if q(x) then y := f(x) else y := g(x)
    else
        if q(x) then y := f(x) else y := g(x)
end
```

Since output y may be set to f(x) if p(x) is false, and g(x) if p(x) is true, one might conclude that this use of p is productive. However, both possibilities exist in either case, the choice being determined by q(x); hence p(x) is not really productive, in agreement with our definition.

Much of this research has been directed toward identifying the most appropriate definitions for "productive control sequences". A seemingly desirable condition is that every use of an operator or decider in a control sequence be productive. Unfortunately, this strong productivity condition limits the degree of parallelism that can be realized. Suppose a sequence must be performed if either of two tests produces true as a result:

```
begin
    if p(x) or q(x) then y := f(x) else y := g(x)
end
```

As soon as either p(x) or q(x) is found to be true, evaluation of the other is unproductive. Thus parallel evaluation of p(x) and q(x) will violate the strong productivity condition. We are studying a weaker form of productivity which does not clash with parallelism.

Data Flow Schemata

An example of a data flow schema is shown in Figure 17. It is a directed graph having two kinds of nodes: actor nodes and link nodes. The arcs of a data flow schema are paths through which data and control values flow from actor nodes to link nodes and from link nodes to actor nodes. Link nodes serve to distribute values to several actor nodes and are of two kinds -- data links drawn as small solid circles for data values, and control links drawn as small open circles for control values. Certain data link nodes are the input nodes of the schema, and certain data link nodes are the output nodes of the schema. Each link node, except the input nodes, has exactly one incident arc, and all but the output nodes have at least one emanating arc.

There are five kinds of actor nodes:

operator	square box with a function letter written inside.
decider	diamond box with a predicate letter written inside.
true gate/false gate	circle with T or F written inside.
merge	ellipse with T and F written inside.
Boolean	square box with one of the symbols $\wedge$, $\vee$, $\neg$ written inside.

Each arc leaving a link node acts like a first-in-first-out queue for values waiting for use by the actor on which the arc terminates. A value arriving at a link node is replicated as required and entered in the queues of the emanating arcs. In most cases, each queue will either be empty or hold one value. However, permitting unbounded queues permits operation of a data flow schemata to achieve a kind of maximum parallelism we shall illustrate by a later example.

Given a data flow schema and an interpretation of its function and predicate letters, computations by the schema are described by sequences of actions by the actor nodes, analogous to the firing sequences of a Petri net. An operator, decider, or Boolean node is enabled to act when at least one value is available from each of its input arcs. When enabled, one of these actors may "fire" by removing one value from each input queue, applying the specified function, predicate or Boolean operator, and sending the results to its output data or control link. A true gate is enabled by the availability of a data value and a control value from its input arcs. The gate fires by removing these values from their queues. Then, if the control value is true the data value is sent to the output data link; if the control value is false no further action takes place. The false gate acts in an analogous manner. A merge node acts by transmitting a value from its F-input arc if the control input value is false, or a value from its T-input arc if the control value is true. The filled-in arrows on certain control links indicate that a false value is entered in their queues in the initial configuration of the schema. This arrangement is needed to initiate action by a portion of a data flow schema that performs an iteration.

According to these rules of behavior, every actor of a data flow schema is persistent: once enabled an actor becomes not enabled only by firing. From this fact and the discipline by which actor and link nodes interact, a result of Patil (12) shows that any data flow schema is a determinate system.

Study of the schema in Figure 17 reveals that it is equivalent to the following "while schema":

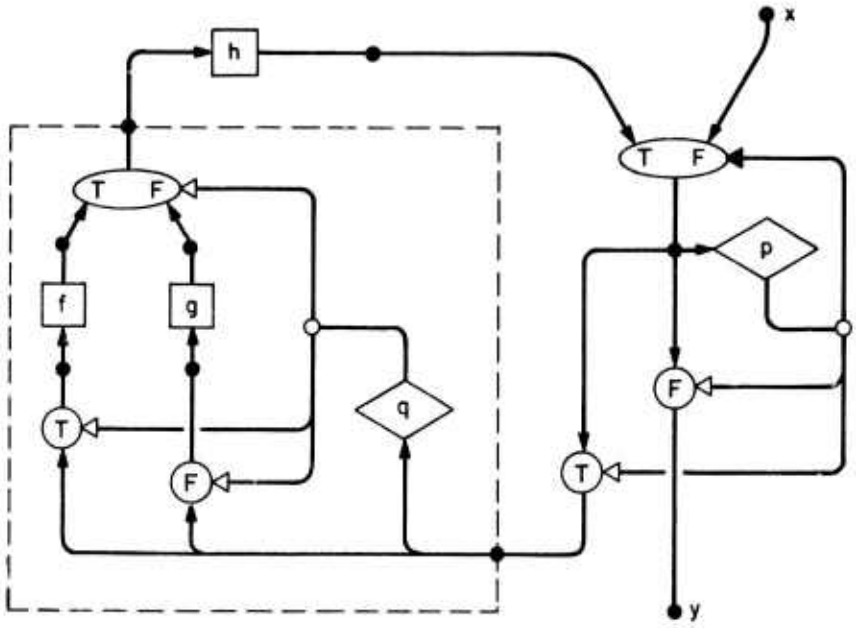


Figure 17. A well-formed data flow schema.

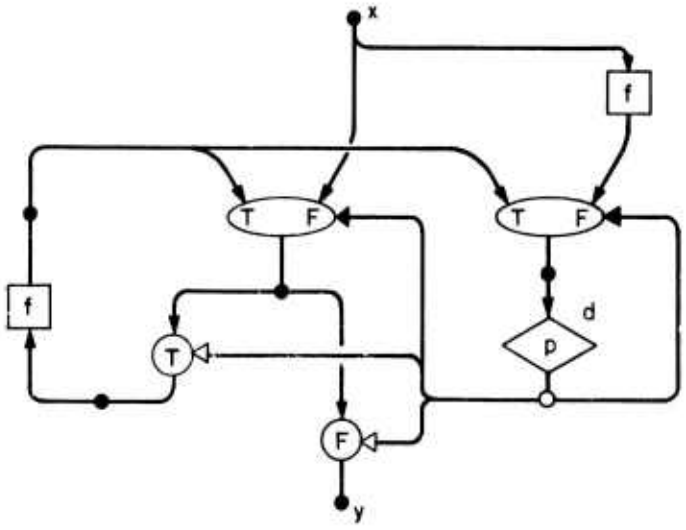


Figure 18. A data flow schema that is not free.

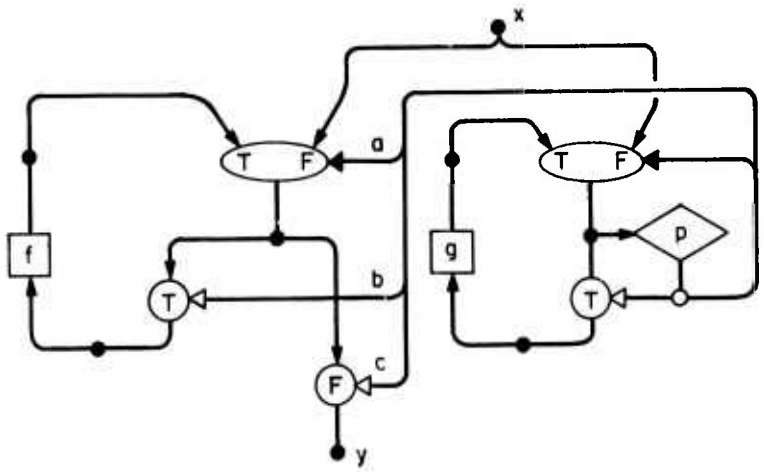


Figure 19. A data flow schema requiring unbounded queues.

```
begin
while p(x) do
     if q(x) then w := f(x) else w := g(x)
     x := h(w)
     end
y := x
end
```

Just as in a while schema, the data flow schema has a nested structure indicated by the dashed lines, and uses specific configurations of gate, merge and decider nodes to form conditional and iteration subschemas. A data flow schema having this structure is said to be well formed. Any well formed data flow schema will generate exactly one value at each output node for each set of values presented at the input nodes. Because it is determinate, any well-formed data flow schema determines a functional dependence of output values on input values. We consider two schemas to be equivalent if both define the same functional dependence of outputs on inputs, and this is true regardless of the interpretation chosen for the function and predicate letters.

On the basis of work by Ashcroft and Manna (1) one can construct a well-formed data flow schema equivalent to any "goto program" or any program schema of the type studied by Paterson (11). Hence the general equivalence problem for data flow schemata is unsolvable.

It has been found that the theory of "free" schemata is more rewarding in terms of positive results than the study of unrestricted schemata. A data-flow schema is said to be free if no two actions by deciders apply the same predicate to the same value. Figure 18 illustrates a schema that is not free because the first two uses of decider d both apply predicate p to the result of applying f to the schema input value. Hence there is no way for the iteration subschema to perform exactly one execution of its body.

John Fosseen (6) has found it possible to transform free data flow schemata in such a way that any pair of data arcs may be tested for equivalence. (Two arcs are equivalent if they pass the same sequence of data values in any computation.) We hope the concepts developed to obtain this result will provide further insight into the equivalence problem for free data flow schemata.

We remarked earlier that treating the input arcs of actors as unbounded queues permits greater concurrency. The data flow schema in Figure 19 illustrates such a case and is based on an example of Keller (10). The right-hand portion of the schema may run arbitrarily ahead of the left-hand portion, a true value being entered in the queues of arcs a, b, and c for each cycle.

The left-hand part may operate as fast as it can until the queues are emptied, whereupon (to be strongly productive) operation must wait for further decisions to be made.

Any data flow schema is inherently maximally parallel in the sense that each operator and decider is at work whenever values are available for some productive use of the operator or decider.

Weakly Productive Computations

In a data flow schema, actions are initiated when the required input values are present and the action (in most cases) is known to be productive. As an interesting exploratory study, we have studied properties of parallel computations in which every operation is initiated as soon as its input values have been computed, so long as some possible continuation of the computation makes productive use of the result. Consider the data flow schema in Figure 20, which represents the following program with input variable x and output variable y:

```
begin
while p(x) do
    if q(x) then x := f(x) else x := g(x)
y := x
end
```

If execution of this schema is performed according to the rules given earlier, then every action by the operators (a and b) and deciders (d and e) is productive. Let us consider what happens if we allow all weakly productive actions to initiate. Suppose termination of the first uses of deciders d and e is arbitrarily delayed. Since the first uses of operators a and b require only the initial value of x, these uses are immediately initiated. Their terminations produce values that are inputs to further weakly productive uses of operators a and b, and so on. These actions define the unbounded tree of values illustrated in Figure 21a; the tree has a node for each value any computation by the schema could generate. As outcomes of decider actions become known, portions of the tree of values become useless and may be deleted, since the operator uses that produce these values become known to be nonproductive. For example, if the first use of decider d yields false, the tree of possibly useful values is as in Figure 21b, and if deciders d and e have successive outcomes F,T and T,T,F, respectively, the tree becomes that in Figure 21c, and represents a completed computation.

Joseph Qualitz (15) has studied the bookeeping requirements for weakly productive computations, and has devised execution structures in terms of which the detailed progress of such computations may be studied. Clearly it is necessary to tag each value produced by a schema operator with the assumptions made

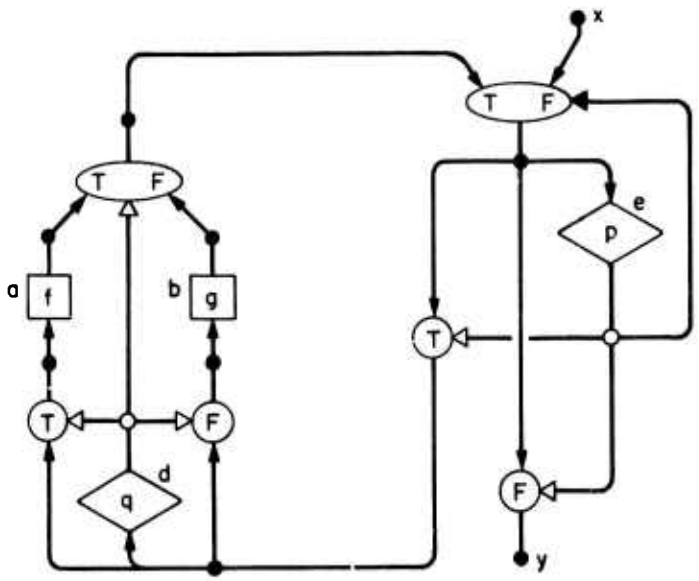


Figure 20. Data flow schema.

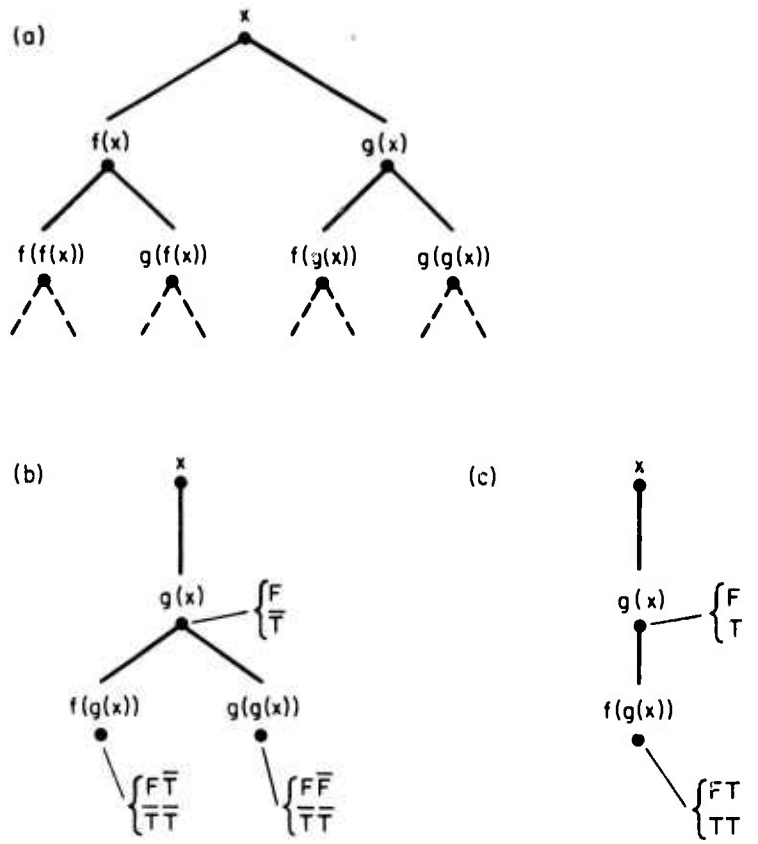


Figure 21. Value trees from a weakly productive computation.

about decider outcomes. We let each value carry a color which is a set of sequences of the symbols $\{T, F, \bar{T}, \bar{F}\}$, one sequence for each decider of the schema. The letters without overbars denote known outcomes, whereas letters with overbars denote assumed outcomes. In Figures 21b, 21c, colors are shown for each value.

At any stage in a weakly productive computation, many values may be associated with certain value nodes of a schema. It is not useful to order these sets of values because, unlike normal execution of a data flow schema, the order in which values arrive is not necessarily the order in which they are used. Instead, each value node is regarded as holding a pool of values, each tagged with the appropriate color, and available for use. Therefore, when an operator or decider has several input value nodes, some means must be provided for identifying the combinations of values to which a function or predicate should be applied. This is done by associating with each value an index that is distinct for each cycle of any loop in the schema. Finally, when a decision is made, certain values become useless and further initiation of actions that use these values must be inhibited.

We have devised rules of execution for weakly productive computations and have shown that these rules correctly simulate the computations of any well-behaved data flow schema.

D. Inductive Proofs of Program Properties

One of the purposes of studying schemata or simplified programming languages is to isolate aspects of programs which must be encompassed by any approach to the construction of formal proofs about the functions computed by programs. Recursion is one such property. To prove equivalence or correctness results about recursive programs, some form of argument by induction must be made. This has been recognized by many people and several of them have formulated induction rules to be used for particular classes of programs. Generally, a program can be viewed as falling in several of these classes. By examining a single program and proofs about it from different viewpoints we have been able to clarify the relationships among these various proof techniques. By means of a simple example we shall illustrate the work of Irene Greif (7) on relating the different ways of interpreting a recursive definition and the corresponding proof techniques.

Consider the following definition of a function f over the nonnegative integers:

$$f(m,n) = \underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m\ \underline{\text{else}}\ f(m + 1, n - 1)$$

(The reader should convince himself that $f(m, n) = m + n$.) The first and most obvious interpretation of the definition is that it describes an algorithm for computing f. The algorithm is to test for $n = 0$; if $n = 0$ then $f(m, n) = m$; otherwise apply the same algorithm in computing $f(m + 1, n - 1)$ to obtain the result. A second interpretation depends on the existence of an ordering on the domain of the function. In this case

the pairs of integers (m,n) can be ordered as follows:

$$(m_1, n_1) \textcircled{<} (m_2, n_2)$$

if and only if

$$n_1 < n_2.$$

Then the definition of f is an inductive definition. The base of the definition is:

For all m $\quad f(m, 0) = m.$

The induction step is:

$$f(m, n) = f(m + 1, n - 1).$$

The third interpretation of f is as the minimal fixpoint of the following functional:

$$\mathcal{C}(X) \equiv \lambda m.\lambda n.\ \underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m\ \underline{\text{else}}\ X(m + 1, n - 1)$$

It can be shown that the minimal fixpoint of $\mathcal{C}$ is $\bigcup_{i=0}^{\infty} \mathcal{C}^i(\Omega)$ where Ω is the function that is everywhere undefined and $^{i}(X)$ means the function produced by i applications of $\mathcal{C}$ to X. Notice that $\mathcal{C}(\Omega) = \lambda m.\lambda n.\ \underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m\ \underline{\text{else}}\ \Omega(m + 1, n - 1)$ is the function which is m for (m, 0) and undefined for all other ordered pairs. $\mathcal{C}^2(\Omega)$ has the value m for the ordered pair (m, 0) and m + 1 for the ordered pair (m, 1) and is otherwise undefined. Proceeding in this manner, the function f which we are expecting will be generated.

The last interpretation is that the function f represents the agreement of its "truncations." These truncations are the partial functions defined as follows:

$$f_i(m, n) = \underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m\ \underline{\text{else}}\ f_{i-1}(m + 1, n - 1).$$

The reader should note that in this case

$$f_i(m, n) = \mathcal{C}^i(\Omega)\ (m, n).$$

Now we will give four different proofs of the following simple fact:

$$f(m + 1, n) = f(m, n) + 1.$$

The first, by recursive induction, corresponds to the notion of definition by algorithm. We show that f(m + 1, n) and f(m, n)+1 can be computed by exactly the same algorithm by showing that they can be expressed in the same form, namely:

$$X(m, n) = \underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m + 1\ \underline{\text{else}}\ X(m + 1, n - 1)$$

1. $g_1(m, n) = f(m + 1, n)$

$= \underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m + 1\ \underline{\text{else}}\ f(m + 2, n - 1)$

$= \underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m + 1\ \underline{\text{else}}\ g_1(m + 1, n - 1)$

2. $g_2(m, n) = f(m, n) + 1$

$= (\underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m\ \underline{\text{else}}\ f(m + 1, n - 1)\ ) + 1$

$= \underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m + 1\ \underline{\text{else}}\ f(m + 1, n - 1) + 1$

$= \underline{\text{if}}\ n = 0\ \underline{\text{then}}\ m + 1\ \underline{\text{else}}\ g_2(m + 1, n - 1)$

This shows that $g_1 = g_2$ on the domain of X. If we are trying to prove $g_1 = g_2$ for the pairs of nonnegative integers, a separate proof about the domain of X will be required.

Another proof can be written, utilizing the partial ordering on the domain of these functions, and the inductive definition. The basis of this proof by <u>structural induction</u> is:

for all m

$f(m + 1, 0) = \underline{\text{if}}\ 0 = 0\ \underline{\text{then}}\ m + 1\ \underline{\text{else}}\ f(m + 2, 0 - 1)$

$= m + 1$

$f(m, 0) + 1 = \underline{\text{if}}\ 0 = 0\ \underline{\text{then}}\ m + 1\ \underline{\text{else}}\ f(m+1, 0-1) + 1$

$= m + 1$

Therefore, for the minimal element in the domain, $f(m + 1, n) = f(m, n) + 1$. The induction step, for (m, n), $n \neq 0$ is:

Assume for (m, n), $n < N$ that $f(m + 1, n) = f(m. n) + 1$

1. $f(m + 1, N) = f(m + 2, N - 1)$

2. $f(m, N) + 1 = f(m + 1, N - 1) + 1$

$= f(m + 2, N - 1)$ by induction since $N-1 < N$.

The initial assumption, based on the means of definition of the function is that f is total on the ordered pairs, partially ordered by $\textcircled{<}$. From this fact and the above proof, we know that $f(m + 1, n) = f(m, n) + 1$ for all pairs of nonnegative integers.

The third proof is actually simple induction on the depth of recursion of a computation. In terms of the definition of the minimal fixpoint

$$f = \bigcup_{i=\infty} \mathcal{C}^i(\Omega),$$

computational (or μ-rule) induction is simple induction on i.

1. for i = 0 we must show

   $$\Omega(m + 1, n) = \Omega(m, n) + 1$$

   Obviously both are totally undefined.

2. Assume $X(m + 1, n) = X(m, n) + 1$

   then prove $\mathcal{C}(X)(m + 1, N) = \mathcal{C}(X)(m, n) + 1$

   $\mathcal{C}(X)(m + 1, n) =$ if n = 0 then m+1 else X(m+2, n-1)

   = if n = 0 then m+1 else X(m+1, n-1) + 1

   (by induction)

   = (if n = 0 then m+1 else X(m+1, n-1))+1

   = $\mathcal{C}(X)(m, n) + 1$

This proves that f(m, n) = f(m, n) + 1 are totally equivalent, i.e., either both are undefined or both are defined and have the same value.

A separate argument can easily be given to show that both functions are defined for all pairs of nonnegative integers.

The last proof technique is very similar to computational induction, being course-of-values induction on the index i of the truncations of a function. This amounts to doing course-of values induction on the depth of recursion. For our particular example, in which equivalence depends only one one step in the computation of the fixpoint, the difference between the two proofs is strictly a matter of formalism. We prove that for i = 0, $f_0(m + 1, n) = f_0(m, n) + 1$.

Then for i ≠ 0:

assume for j < i, $f_j(m + 1, n) = f_j(m, n) + 1$

$f_i(m + 1, n) =$ if n = 0 then m+1 else $f_{i-1}(m+2, n-1)$

= if n = 0 then m+1 else $f_{i-1}(m+1, n-1)+1$

= (if n = 0 then m+1 else $f_{i-1}(m+1, n-1)$)+1

= $f_i(m, n) + 1$

As in the last proof, this shows strong equivalence, this time by truncation induction.

Generally, any method can be used for a proof. If the programmer had one of these interpretations in mind in writing his program, then the corresponding proof technique will probably seem most natural. Ideally an automatic program verifier would be flexible with respect to choice of induction rule. It is unlikely, however, that all of these will be equivalently useful in mechanical proofs, even though there seems to be no real difference in scope of application among them.

E. A Computer for General Data Types

One goal in the design of programming systems is to retain the generality of an algorithm when it is encoded into the language of the programming system. A serious limitation on the generality readily achieved in contemporary computer systems is imposed by the fixed word length and finite size of computer memories.

In preparing a program for execution by a computer system, the programmer first imagines the abstract function the program is to implement. Simple examples might be to implement the scalar product of any two n-component vectors of real numbers, or to obtain the greatest common divisor of two integers. As in these examples, the abstract function almost always has an infinite domain. Then the programmer conceives of an algorithm for the function -- a step-by-step process for obtaining the value of the function through the use of idealized primitive operations such as the arithmetic operations on integers and reals. The next step is to express the algorithm in the language of some practical programming system. Usually the actual data types of the programming system have their idealized counterparts, and, if the language is suited to the needs of the algorithm, the algorithm may be converted into a program with little difficulty. Our problem of generality would be solved if the task of the programmer were completed at this point. However, he must now check whether the word size and finite memory size of the computer system, as reflected in defects of the primitive operations of the programming language, may prevent correct operation of his program. In many cases, the program will operate correctly for a large (but finite) number of points in the domain of the abstract function, and will fail (often without any hint to the user) for the remaining (infinite) set of domain points. In other cases the programmer will find that the number of cases for which the program will work correctly is too small to be of interest and a new approach, using a language less suited to expressing the algorithm, or less efficient in execution, must be adopted.

The ability of a programming system to correctly execute programs expressed in terms of idealized data types is called generality with respect to domain. Most programming systems fail to be general with respect to domain by limiting the amount of storage that may be allocated to one data value to less than the available memory of the computer on which the

programming system runs. For instance, integers are usually limited in value by the number of bits in one memory word, and the maximum range of an array subscript must often be specified at the moment the array is created.

Since the memory of any practical computer system is, in fact, finite we cannot expect any program to obtain the value of the programmer's abstract function for any point in its domain. However, we should expect a programming system to produce the correct result unless the computer system runs out of memory in trying. (If the computer system runs out of memory, should one blame the program for the absence of sufficient memory to compute the function ?) This consideration is the basis for the following definitions:

Definition: A program p, with input variables $\underline{x} = (x_1, \ldots, x_m)$ and output variables $\underline{y} = (y_1, \ldots, y_n)$, computes a function f over domain D if and only if for each point $\underline{x}$ in D either

1. program p produces output $\underline{y}$ from input $\underline{x}$ where $y = f(x)$, or
2. for input x program p fails to complete due to an unsatisfied request for additional storage.

Thus a program that computes a function must obtain the correct result whenever it is given sufficient resources to operate.

Definition: A programming system that implements a language L is general with respect to domain if and only if for any algorithm that defines a function f on domain D, the corresponding program in L computes f on D.

The heart of the problem of implementing programming systems having generality with respect to domain is machine instructions which themselves are programs not general with respect to domain. The basic arithmetic instructions, for example, usually operate on representations that occupy a single register. Since conventional programmed multiple length arithmetic introduces a high cost in time consumed, even for quantities that require only single-length representation, achieving generality for these data types in a conventional computer system is unattractive.

Peter Bishop (3) has designed an abstract computer in which generality with respect to domain is achieved for a large class of data types including integers, floating point numbers, strings and arrays, as well as more elaborate structures. In the abstract computer, each data value is represented by a pointer-linked tree structure having as many elements as necessary to represent the value. The representation of any quantity may expand arbitrarily as required until available memory is exhausted.

References

1. Ashcroft, E., and Manna, Z., The translation of 'go to' programs to 'while' programs. Information Processing 71, Ljubljana, 1971.

2. Baker, H., Petri Nets and Languages. Computation Structures Group Memo 68, Project MAC, M.I.T., May 1972.

3. Bishop, P. B., Data Types for Programming Generality. S.B. and S.M. Thesis, Department of Electrical Engineering, M.I.T., June 1972.

4. Chaney, T. J., Ornstein, S. M. and Littlefield, W. J., Beware the synchronizer. Proceedings of the Sixth Annual IEEE Computer Society International Conference, San Francisco, September 1972, pp 317-319.

5. Commoner, F., Holt, A. W., Even, S., and Pnueli, A. Marked directed graphs. J. of Computer and System Sciences, Vol. 5 (1971), pp 511-523.

6. Fosseen, J. B., Representation of Algorithms by Maximally Parallel Schemata. S.M. Thesis, Department of Electrical Engineering, M.I.T., June 1972.

7. Greif, I. G., Induction in Proofs About Programs. S.M. Thesis, Department of Electrical Engineering, M.I.T., December 1971.

8. Holt, A. W. and Commoner, F. Events and Conditions. (In three parts), Applied Data Research, New York 1970. (Chapters I, II, IV and VI appear in Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York 1970, pp 3-52.)

9. Karp, R. M., and Miller, R. E., Parallel program schemata. J. of Computer and System Sciences, Vol. 3, No. 2 (May 1969), pp 147-195.

10. Keller, R. M., On maximally parallel schemata. IEEE Conference Record. Eleventh Annual Symposium on Switching and Automata Theory, 1970, pp 32-50.

11. Luckham, D. C., Park, D. M. R., and Paterson, M. S. On formalized computer programs. J. of Computer and System Sciences, Vol. 4, No. 3 (June 1970), pp 220-249.

12. Patil, S. S., Closure properties of interconnections of determinate systems. Record of the Project MAC Conference on Concurrent Systems and Parallel Computation, ACM, New York 1970, pp 107-116.

References (continued)

13. Patil, S. S., Forward Acting n x m Arbiter. Computation Structures Group Memo 67, Project MAC, M.I.T., June 1972.

14. Plummer, W. W., Asynchronous Arbiters. IEEE Trans. on Computers, Vol. C-21, No. 1 (January 1972)

15. Qualitz, J. E., Weakly Productive Computation Schemata. S.B. and S.M. Thesis, Department of Electrical Engineering, M.I.T., May 1972.

16. Rodriguez, J. E., A Graph Model for Parallel Computations. Technical Report TR-64, Project MAC, M.I.T., September 1969.

17. Slutz, D. R., The Flow Graph Schemata Model of Parallel Computation. Technical Report TR-53, Project MAC, M.I.T., September 1968.

COMPUTATION STRUCTURES

Publications

1. Baker, H., "Petri Nets and Languages", Computation Structures Group Memo 68, May 1972.

2. Dennis, J. B., "Management of Names in a Computer System", Computation Structures Group Memo 63, November 1971.

3. Dennis, J. B., "Concurrency in Software Systems", Computation Structures Group Memo 65-1, June 1972.

4. Dennis, J. B., "The Design and Construction of Software Systems", Computation Structures Group Memo 69, June 1972.

5. Dennis, J. B., "Modularity", Computation Structures Group Memo 70, June 1972.

6. Dennis, J. B., On the Design and Specification of a Common Base Language, Project MAC, M.I.T., MAC TR-101, June 1972, AD 744-207.

7. Fano, R. M., "On the Number of Bits Required to Implement an Associative Memory", Computation Structures Group Memo 61, August 1971.

8. Flinker, E., "Translation of a Block Structured Language Into the Common Base Language", Computation Structures Group Memo 66, January 1972.

9. Fox, P. J., "A Look at 'The Controlled Execution of Parallel Programs Operating on Structured Data' by Ian Campbell-Grant", Computation Structures Group Memo 62, October 1971.

10. Greif, I. G., Induction in Proofs about Programs, Project MAC, M.I.T., MAC TR-93, February 1972, AD 737-701.

11. Hack, M. H. T., Analysis of Production Schemata by Petri Nets, Project MAC, M.I.T., MAC TR-94, February 1972, AD 740-320.

12. Lester, B. P., Cost Analysis of Debugging Systems, Project MAC, M.I.T., MAC TR-90, September 1971, AD 730-521.

13. Patil, S. S., "Forward Acting n x m Arbiter", Computation Structures Group Memo 67, June 1972.

Talks

1. Dennis, J. B., and S. S. Patil, "Systematic Realization of Asynchronous Systems", IEEE Seminar, Boston, September 8, 1971.

2. Dennis, J. B., "Design of a Common Base Language"; and "A Data Flow Model of Computation", Tutorial Symposium on Semantic Models of Computation, New Mexico State University, Las Cruces, New Mexico, January 3-5, 1972.

3. Dennis, J. B., "The Design and Construction of Software Systems"; "Modularity"; and "Concurrency in Software Systems", Advanced Course on Software Engineering, Technical University of Munich, Munich, Germany, February 21 - March 4, 1972.

4. Dennis, J. B., "On the Design and Specification of a Common Base Language", talk at the General Electric Co., Schenectady, New York, July 15, 1972.

Theses Completed

1. Bishop, P. B., Data Types for Programming Generality, S.B. and S.M. Thesis, Department of Electrical Engineering, M.I.T., June 1972.

2. Fosseen, J. B., Representation of Algorithms by Maximally Parallel Schemata, S.M. Thesis, Department of Electrical Engineering, M.I.T., June 1972.

3. Furtek, F. C., Modular Implementation of Petri Nets, S.M. Thesis, Department of Electrical Engineering, M.I.T., September 1971.

4. Greif, I. G., Induction in Proofs about Programs, S.M. Thesis, Department of Electrical Engineering, M.I.T., February 1972.

5. Hack, M., Analysis of Production Schemata by Petri Nets, S.M. Thesis, Department of Electrical Engineering, M.I.T., February 1972.

6. Qualitz, J. E., Weakly Productive Computation Schemata, S.M. Thesis, Department of Electrical Engineering, M.I.T., May 1972.

Theses in Progress

1. Amerasinghe, S. N., "The Handling of Procedure Variables in a Base Language", S.M. Thesis.

2. Fox, P. J., "Representation of Parallel Computation on Data Structures", S.M. and E.E. Thesis.

3. Hawryszkiewycz, I. T., "The Semantics of Data Base Systems", Ph.D. Thesis.

Theses in Progress (continued)

4. Lester, B. P., "Smooth Interprocess Communication", Ph.D. Thesis.

5. Lotspiech, J., "Reliability of Speed-Independent Asynchronous Systems", S.M. Thesis.

6. Ramchandani, C., "Models for the Analysis of Asynchronous Systems", Ph.D. Thesis.

7. Rotenberg, L. J., "Making Computers Keep Secrets", Ph.D. Thesis.

COMPUTER SYSTEMS RESEARCH

Prof. F. J. Corbató

Academic Staff

Prof. J. H. Saltzer

Instructors, Research Associates, Research Assistants and Others

Y. S. Auyang
R. G. Bratt
D. Bricklin
S. S. Brown
D. D. Clark
T. L. Davenport
T. M. Demchock
D. R. Dick
A. R. Downing
R. M. Frankston
B. S. Greenberg
R. A. Gumpertz
R. M. Haas
G. Harris
D. H. Hunt
P. A. Karger
S. Kuo
R. S. Lamson
D. A. Moon
S. S. Perrin
T. T. Pogran
D. P. Reed
K. G. Rhoads
L. J. Scheffler
M. D. Schroeder
A. Sekino
J. A. Stern
C. M. Vogt

DSR Staff

R. J. Feiertag
C. C. Garman
R. K. Kanodia
R. F. Mabee
K. J. Martin
E. W. Meyer, Jr.
N. I. Morris
M. A. Padlipsky
M. J. Spier
V. L. Voydock
M. B. Weaver
S. H. Webber
D. M. Wells

Support Staff

O. D. Carey
S. Daise
C. P. Doyle
L. J. Haron
D. L. Jones
A. G. Testa
M. F. Webber

Guests

Prof. K. Ikeda
M. Miyazaki

COMPUTER SYSTEMS RESEARCH

A. Progress Report

1. Introduction

The Computer Systems Research Group concentrates upon discovering ways to make engineering of complex information systems more methodical. Its approach is experimental, in contrast to the more theoretical attack followed by the Computation Structures Group. Continuing to use the M.I.T. Multics system as a laboratory, significant work was performed in multi-level memory management, networking, performance modeling, and protection mechanisms. Another focus of attention was preparing to move the system to modern hardware. Routine development, operation, and maintenance of the system were largely handled by the M.I.T. Information Processing Center and Honeywell.

The year has seen less development and more experimentation than previously. As the Multics system came under the operational control of others, members of the Group, especially students, have become able to address themselves to such areas as models of system performance, instrumentation, and the like. In the Computer Networks area as well, concern with making use of the existing systems for study and experimentation has arisen, although implementation is still not completely satisfactory. This trend away from the design and building of systems toward analysis and exploitation is also seen in the Group's participation in support of Automatic Programming. Here, the focus is on the rapid engineering of "hospitable" environments.

Also, during the last year, there was published the long-awaited book on Multics by Prof. E. Organick. The book which describes the internal structure of the Multics system should be of major assistance in deseminating the key design issues. An effort is also underway in Japan to translate and publish the book in Japanese.

2. Multilevel Storage Management

As discussed in some detail last year, a major interest of the Group is the instrumentation of and experimentation with the multilevel memory system of Multics. Several new facilities were introduced into the Multics supervisor to support experimentation. A major step in this direction was taken with the installation of a multilevel paging strategy which, in addition to the usual automatic management of page movements between core and drum, automatically manages the migration of pages between drum and disk. Pages are moved from disk to drum on activation, and an LRU algorithm governs the migration from drum to disk. In effect, the drum becomes a kind of associative memory. The interaction between the two loosely coupled LRU algorithms at different levels is now being studied in detail.

The more effective use of the drum made by the multilevel paging strategy opened the way for a new strategy in drum management which trades space for time. To decrease read latency, multiple copies of pages are written on the drum.

Preceding page blank

Currently, two copies are written, giving rise to the name "2-fold drum manager", but further experiments are possible with the existing software to investigate the characteristics of a 4-fold -- or for that matter, n-fold drum manager. As with multilevel paging, 2-fold drum manager has proven to be a practical success in increasing system effectiveness, in addition to its experimental interest.

Another new strategy of practical as well as experimental interest was proposed and implemented, although not yet installed, in disk management. Referred to as "pre-seeking", this strategy entails having the disk DIM determine whether the seek-portions of pending requests can be overlapped with data transfers. This work evolved from Master's thesis research on the modeling of algorithms for moveable-head storage devices. Although pre-seeking is an old idea in read-write type file systems, we view it as the next step in the exploitation of single strategies matched to demand paging environments.

Neither the n-fold drum manager nor the disk pre-seek strategy is thought to be a unique approach, but in conjunction with multilevel paging strategy they offer a rich environment for further experimentation in the area of multilevel storage management. To offer a standard of comparison for experiments a benchmark load is being developed. Although one might have expected a benchmark load to be among the earlier of projects, in fact it is only now that enough information has been collected about the ways a Multics-type system is used; thus construction of a realistic benchmark can now proceed.

A paper by Prof. J. H. Saltzer (still in draft) has been written on a simple model of demand paging, which is based on the experimental observation that over a wide region, the mean number of memory references between paging exceptions roughly increases linearly with the size of the paging memory. The resulting model is easy to manipulate, and is applicable to such diverse problems as choosing an optimum size for a paging memory, arranging for reproducible memory usage charges and estimating the amount of core memory sharing. The model has already served as the basis for an implementation on Multics of an accounting scheme which allows for charging memory utilization independent of system configuration and system load.

3. ARPA Network

Considerable progress has been made in the Multics ARPA Network implementation. At the beginning of the reporting period, it was possible on a demonstration basis to login to Multics from Multics over the Network. By the end of the reporting period, the Network was a standard feature of Multics. Network logins on a supported, "sampling" project were on the order of half a dozen a day, and there was at least one paying customer of the system on the West Coast and one at the National Bureau of Standards. The originally high cost of Network use of Multics has been coming down drastically

The target is for use of Multics via the Network to be no more expensive than from local teletypewriters. As the basic machinery comes under control, the Group's Network team will pay greater attention to participating in the design of Network Graphics and File Transfer protocols.

4. Performance Model

The doctoral thesis mentioned in last year's report is being completed. In this thesis, a set of hierarchically organized analytical models has been developed to evaluate the performance of multiprogrammed time-shared computer systems using demand paging. These models are capable of predicting the throughput and response time characteristics of such systems, as a function of system hardware/software configuration as well as of the characteristics of terminal users and their programs. These models are then used in optimizing the system performance and deriving the best system configuration. The system performance predicted by these models is compared with that of an actual computer system, i.e., Multics, in order to examine the usability of these models as a design aid.

5. Protection of Programs and Data

With the opportunity for use, it has become apparent that the original Multics access control mechanism present an interface which is too complex and confusing even for expert users. Since confusion leads to misuse, and misuse to lack of the desired control of access, we have here a significant defect.

As a result, a minor interface redesign was undertaken. The changes, when implementation is completed shortly, will simplify and clarify the use of these mechanisms, making their application to user protection problems more apparent to users. The potentially more far-reaching step was the evolution of a detailed plan for allowing all users to define protected subsystems and share them with one another. The proposed facility would allow groups of programs and data segments to be encapsulated and shared, potentially with any system user, in such a way that access of the borrowing user would be restricted to calling "gate" entry points. This scheme seems to utilize the full functional capabilities of the ring protection hardware in the follow-on processor. The plan will be reported on in greater detail when it is implemented.

Finally, a thesis on protection referred to in last year's report is almost completed. This thesis describes practical protection mechanisms that allow mutually suspicious subsystems to cooperate in a single computation, a case which is beyond the functional capability of protection rings. The mechanisms are based on an extension of the domain model of protection. This extension makes explicit provision for communicating arguments between domains when the execution point of a computation passes from one domain to another. The extended domain model serves as the basis for the design of a hardware processor which automatically enforces the complex access constraints associated with general multi-domain computations.

This processor allows a standard subroutine call with arguments to change the domain of execution. Arguments are automatically communicated on cross-domain calls -- even between domains that normally have no access capabilities in common. The processor, when supported by suitable software for specifying the protection relationships to be enforced, allows cooperating, mutually suspicious subsystems to be implemented efficiently and naturally in a computer utility. The thesis points out a direction in which a multi-like system could evolve to support applications where more complex access constraints are required than can be enforced by a mechanism based on concentric rings of protection.

6. Follow-On Hardware

The desire to secure a hardware base for Multics on modern, stable equipment led to the signing of a contract between the M.I.T. Information Processing Center and Honeywell during the reporting period for follow-on Multics hardware which will be based upon the Honeywell 6000 series technology. The follow-on machine will bear essentially the same relation to the Honeywell 6080 as the 645 does to the 635, where the primary differences are the presence of address appending hardware and an associative memory on the Multics machine. (The 6080 is roughly three times faster than the 635.) Although some pieces of new hardware are already on-site, most of the equipment is currently scheduled to be delivered during the Fall of 1972. The intent in both hardware and software is to minimize differences between the new and old systems; however, certain new features are planned in both areas which will carry a high payoff for a relatively small alteration. In the hardware, a modified pointer register structure supports the Multics protection ring mechanism; the high performance drum is replaced by bulk storage; and the number of general registers (long felt to be a constraint on the current machine) will be increased. In the software, new, more efficient call-save-return macros will be introduced in response to the metering observation that the number of subroutine calls on Multics is quite high; the bulk of the software which manages the protection ring mechanism will be removed, in view of the hardware ring support; and the PL/1 compiler will take advantage of the 6080's new Extended Instruction Set to perform character string manipulation more efficiently. Rough estimates indicate that the combined effects of new hardware and software features should result in an overall performance improvement factor of anywhere from three to seven.

Although Honeywell is leading the overall transition effort, the lessons learned in working with the present (essentially prototype) machine were of great value to members of the Group in their participation in the engineering of both software and, to at least an advisory/conceptual extent, hardware. During the year, the strategy for making the transition from the old system to the new was planned in detail and work was begun. Indeed, for around half the period, the current development system has been running not with a 645 GIOC (as the Service system does) but with the I/O controller

and communications computer of the follow-on.

7. Maintenance of Multics

As noted above, normal development, operation, and maintenance of Multics have devolved upon the M.I.T. Information Processing Center and Honeywell. It is significant that a new Multics site was brought into existence during the year by Honeywell personnel in Paris, France. Compaigne Honeywell Bull is using this system as a 'software laboratory", in conjunction with a similar facility in Waltham, Massachusetts. (The fourth 645-based Multics site is at Rome Air Development Center, New York.) Although occasional bursts of help from CSR personnel are required on very hard problems in certain areas, the successful day-to-day operation of four separate Multics sites speaks well of the successful transfer of expertise as well as responsibility to Honeywell.

8. MPM Upgrade

Experience with the ARPA Network has strongly underscored the importance of user documentation. Both our experience with attempting to learn to use other systems and reports by Network participants from other sites on their attempts to learn to use Multics suggest that the problem of teaching-at-a-distance is far from solved. The necessity of furnishing a solid foundation for learning in the form of documentation appears to be inescapable. A concerted effort to both upgrade the quality and complete the contents of the Multics Programmers' Manual was, then, probably the most important contribution of the Group to making the system easy to use during the reporting period. As the MPM is the primary source for disseminating knowledge about the system, improvements in its quality serve a useful purpose for both users and students of the system. Particularly noteworthy from the user's point of view are the additions of a rather extensive "Beginners' Guide" and an index. Several new Reference Data Sections have been produced, as well as many new command and subroutine writeups.

9. Programming Languages

Significant work on several programming languages was also performed. Although only initial steps were taken during the reporting period, an upgrade of the LISP implementation is of particular interest because of its impact on the implementation of the Automatic Programming project. (See Future Plans for more information on this project.) The effort is being performed jointly with Automatic Programming personnel, and is the first step taken in the Group's collaboration in that area.

An experimental version of APL was developed by Honeywell with assistance from a Computer Systems Research Group student. Although it is only an initial implementation, it has created considerable interest in the ARPA Network community as it is the only APL known to be available on the Network. Its interface to the user has been constructed to be exactly the same

as the IBM APL -- a construction which most APL enthusiasts consider essential. A plan for improving its performance has been evolved, which it is hoped will be implemented this summer.

Work at Honeywell is nearly complete on "Version II" PL/I and FORTRAN. The PL/I is ANSI standard, and offers superior performance as well as ability to make use of the capabilities of the follow-on hardware (which contains an extended instruction set to facilitate character manipulation). It is interesting to note that the Fortran compiler shares the code generator of the new PL/I compiler.

An XPL compiler implementation was completed as part of a Bachelor's thesis. This language is of interest since it is the vehicle for SPL, a PL/I subset language which is used in the M.I.T. introductory programming course. The languages are based on the work of McKeeman, Horning and Wortman at Stanford University and the University of California at Santa Cruz.

B. Future Plans

In the next twelve months, the following major projects are planned:

- Continued support of Honeywell's conversion of Multics to the 645F hardware system. This activity will reach a peak in Fall, 1972, at the time of delivery of the 645F processor and memory, and continue through Spring of 1973, when the conversion is scheduled for completion. Project MAC is currently providing about 1/4 of the manpower for the conversion activity. Development, with the Automatic Programming group, of a LISP interpreter and compiler system on Multics. This work has two goals:

  1. A language essentially identical to the LISP used on the MAC and Artificial Intelligence Laboratory's PDP-10's.

  2. A compiler which produces the highest quality object code possible for the 645 follow-on processor. The combination of an instruction execution rate in excess of 10^6 instructions/second, a 384K word core memory, and a virtual memory, should make this compiler a very potent tool.

- Completion of the Multics Programmers' Manual (MPM) and the System Programmers' Supplement (SPS) to the MPM. These two books are both about 3/4 complete, and a concerted effort to finish them is underway. Arrangements have been made with Honeywell's publications office to take over editing, updating, and publication of these two manuals as soon as they are substantially complete. Current schedules call for the MPM to be available from Honeywell in first quarter of 1973. The SPS will come later.

Publications

1. Bensoussan, A. C., C. T. Clingen, and R. C. Daley, "The Multics Virtual Memory: Concepts and Design", Communications of the ACM, Vol. 15, No. 5, pp. 308-318.

2. Corbató, F. J., C. T. Clingen, and J. H. Saltzer, "Multics: The First Seven Years", AFIPS Conference Proceedings, Vol. 40, 1972 Spring Joint Computer Conference, pp. 571-583.

3. Feiertag, R. J., and E. I. Organick, "The Multics Input/Output System", Third Symposium on Operating Systems Principles, October 18-20, 1971, pp. 35-41.

4. Multics Programmers' Manual, Revision 10, M.I.T. Information Processing Center Publications Office, May 4, 1972.

5. Organick, E. I., The Multics System: An Examination of its Structure, M.I.T. Press, Cambridge, Massachusetts, and London, England, 1972.

6. Saltzer, J. H., "Some Observations about Decentralization of File Systems", Proceedings of the 1971 IEEE International Computer Society Conference, September 22-24, 1971, pp. 163-164.

7. Schroeder, M. D., and J. H. Saltzer, "A Hardware Architecture for Implementing Protection Rings", Communications of the ACM, Vol. 15, No. 3, pp. 157-170.

8. Sekino, A., "Response Time Distribution of Multiprogrammed Time-Shared Computer Systems", Proceedings of the Sixth Annual Princeton Conference on Information Sciences and Systems, March 23-24, 1972, pp. 613-619.

9. Sekino, A., "Discussion of Section VI", in Statistical Computer Performance Evaluation, Walter Freiberger, Editor, Academic Press, New York, 1972, pp. 461-462.

Talks

1. Corbató, F. J., "The Multics System: Steps Toward A Computer Utility", presented at the conference on Time-Sharing and Multi-Access Computing in the 70's by ONR/SDC, November 9, 1971, Santa Monica, California.

2. Padlipsky, M. A., "ARPA Network Users Experience", panel at SHARE, New York, August, 1971.

3. Saltzer, J. H., "Current Research in Computer Operating Systems at M.I.T.", talk given at Stanford University, October 20, 1971; Princeton University, November 18, 1971; and University of Pittsburgh, February 18, 1972.

Talks (continued)

4. Saltzer, J. H., "Large Scale Computing Systems in the U.S. Today", talk at Kansai Institute of Information Systems, January 24, 1972.

5. Saltzer, J. H., "System Resources required for Supporting Interactive Terminal Use", given at USIB Symposium on Terminals, Washington, D. C., May 23, 1972.

6. Schroeder, M. D., "Cooperating Mutually Suspicious Subsystems", given at University of California at Berkeley, Stanford University and Washington State University, January, 1972.

7. Schroeder, M. D., "A Hardware Architecture for Implementing Independent Domains", Palo Alto Research Center, Xerox Corporation, February, 1972.

8. Sekino, A., Discussant for a session on Systems Management at the Conference on Statistical Methods for the Evaluation of Computer Systems Performance, Brown University, November 22-23, 1971.

Thesis Completed

1. Karger, Paul, An Implementation of XPL for Multics, S.B. Thesis, Department of Electrical Engineering, M.I.T., 1972.

Doctoral Theses in Progress

1. Clark, D. D., "The Input/Output in a Virtual Memory Computer System".

2. Schroeder, M. D., "Allowing Cooperation of Mutually Suspicious Subsystems in a Computer Utility".

3. Sekino, A., "Performance Evaluation of Multiprogrammed Time-Shared Computer Systems".

DYNAMIC MODELING, COMPUTER GRAPHICS, AND COMPUTER NETWORKS

Prof. J. C. R. Licklider

Academic Staff

Prof. J. J. Donovan

Prof. N. P. Negroponte

Instructors, Research Associates, Research Assistants and Others

J. Aiello
P. M. Allaman
B. K. Daniels
J. F. Farrell
R. J. Fleischer
D. Folger
H. Forsdick
R. M. Fox
W. Godfrey
L. I. Goodman
S. Gregory
F. E. Guertin
P. W. Hughett
J. W. Johnson
M. Knaur
D. Koenig
S. Kruger
P. D. Lebling
W. J. Long
S. E. Madnick
W. A. McCray
M. S. Miller
S. G. Morton
H. F. Okrent
G. F. Pfister
J. Piggins
J. Sabath
M. S. Seriff
W. G. Shaw
R. A. Stern
J. R. Taggart
S. Tepper
R. W. Weissberg
S. Zaborowski
C. Ziering

Undergraduate Students

M. H. Alpert
H. R. Brodie
J. L. Caruso
A. Y. Chan
C. C. Conklin
R. G. Curley
S. E. Cutler
R. A. Freedman
D. E. Geer
R. A. Guida
J. H. Harris
W. F. Hui
E. Kant
R. M. Katz
C. A. Kessel
R. N. King
N. V. Kohn
C. K. Leung
G. Pavel
R. L. Prakken
L. M. Rubin
N. D. Ryan
J. D. Sybalsky
M. E. Wolfe

DSR Staff

A. K. Bhushan
E. H. Black
M. F. Brescia
R. D. Bressler
A. L. Brown
M. A. Cohen
D. G. Cressey
M. S. Draper
S. W. Galley
R. P. Goldberg
J. F. Haverty
F. Y. Knight
R. M. Metcalfe
J. C. Michener
L. G. Pantalone
S. G. Peltan
C. L. Reeve
A. Vezza

Support Staff

M. A. Bizot
M. S. Broos
M. T. Cheney
M. Cummings
C. T. Falls
J. A. Haley
A. J. Hicks
R. F. Hill
E. F. Nangle
S. Pitkin

DYNAMIC MODELING, COMPUTER GRAPHICS, AND COMPUTER NETWORKS

A. INTRODUCTION

This year has brought almost to culmination the joint computer-system research and development effort of the Dynamic Modeling Group, the Computer Graphics Group, and the part of the Computer Networks Group based on the PDP-10 computer. The effort has brought into being a rapidly responsive, highly interactive time-sharing system with sophisticated graphic display, good connection to the ARPA Network, and unusually well developed facilities for using and augmenting through use a library of sharable procedures and data sets. This summer, the two-and-a-half groups are beginning a transition from computer system building to research in automatic programming that will exploit the computer system. Final integration of several parts of the system and the shift into automatic-programming research will continue through the fall, and most of the reporting of ideas tested and experience gained in the system-building effort will be done during the fall and winter. It is possible now, nevertheless, to present a description of the computer system and to relate several aspects of its design to the aims and aspirations that motivated the project.

The system has been called the "Dynamic Modeling System" or "DynaMod" despite efforts to promote use of names that reflect the participation of Computer Graphics and Computer Networks as well as Dynamic Modeling. The name "Dynamic Modeling System" will be used in this report.

B. MOTIVATION

The motivation behind the Dynamic Modeling System had (and has) several components. In a general way, of course, they all stemmed from the drive for machine-aided cognition that motivated so much of computer research and development during the 1960's. When the Dynamic Modeling System was undertaken, however, there was a widespread feeling that the thrust of the 1960's had fallen short in several ways. Most time-sharing systems were slow to respond -- not "highly interactive". None (we know or know of) had good facilities for general-purpose graphical interaction. The most responsive systems and the ones most convenient to use were either one-language systems (e.g., APL) or systems with two or three essentially noncommunicating language subsystems. And, most importantly, nowhere was there a time-sharing system that had amassed or seemed on the threshold of amassing a large, sharable, consistent, coherent collection of procedures and sets of data that could be retrieved on-line by users and conveniently incorporated into their own programs. The concept of a large and coherent run-time library had been about for several years as a gleam or a dream, but it had not been approached at all closely in actuality.

Many of the features that we considered desiderata for the Dynamic Modeling System had been implemented in one time-sharing system or another by the time the Dynamic Modeling

Preceding page blank

System was undertaken. (Work on the Dynamic Modeling System began, effectively, in October, 1969, when a used PDP-6 computer with 32K words of memory was delivered to Project MAC.) The designers of Multics had solved most of the problems that were important to us in the software-technical area of controlled sharing of procedures and data. The Artificial Intelligence Laboratory's PDP-10/ITS system was providing fast and convenient interaction (mainly alphanumeric) and the myriad benefits of the LISP language with its options of interpretation and compilation. IBM's APL was demonstrating most of the advantages (and some of the disadvantages) of working interpretively in a neatly designed language and compact notation. The Lincoln Laboratory's TX-2/Apex Time-Sharing System was operating with time-shared graphics and was testing out an approach (the Reckoner) to coherence in programming and systematic sharing of programs. In the Augmentation Research Center of Stanford Research Institute, Engelbart's On-Line System was showing how effectively a well organized multi-access file system can correlate and integrate the activities of the members of a design team. And there were, indeed, several other time-shared computer systems that demonstrated effectively one or more of the features that we considered essential, but none demonstrated all the features. The aspiration for the Dynamic Modeling System was to put all the desiderata together in one reasonably efficient system. Again, the most important design desiderata were:

1. Fast response
2. Convenient interaction
3. Well developed graphic display
4. Where and when needed, the flexibility provided by interpretation; where and when needed, the efficiency provided by compilation or assembly
5. Large, coherent run-time library of sharable procedures and data -- augmented by application-oriented users as well as system programmers

C. OVER-ALL PLAN OF THE DYNAMIC MODELING SYSTEM

The general plan of the Dynamic Modeling System is shown in Fig. 1. The figure shows only the largest features of the landscape. They consist, of course, of hardware and software -- of consoles, central software, central hardware, and interconnections.

CONSOLES

The consoles are Imlac PDS-1's, Adage Advanced Remote Display Stations (ARDSs), and Adage-built vector generators and displays that are part of the Evans and Sutherland LDS-1 Line Drawing System. For graphical input there are "mice", "joysticks", and one stylus tablet. (More stylus tablets are on order.) The only software in the consoles is in the Imlac consoles, which include minicomputers. Most of the software that operates in the Imlacs was provided by the manufacturer, but we have augmented it and, on a larger scale, prepared PDP-10 software that takes advantage of the Imlac's impressive

capabilities in editing and graphic display.

CENTRAL SOFTWARE

The great preponderance of our effort has been devoted, as shown in Fig. 1, to work at levels above the operating system. We have tried to develop a system of facilities extending upward and outward from the operating system toward and into the realm of user applications. It may help to distinguish our focus from those of the primary developers of Multics if we say that we adopted an already developed operating system and modified it only insofar as absolutely necessary for our purpose, whereas they created an operating system de novo devoting most of their resources to it, and left in large part to various subsystem developers in the Multics user community the task of creating the extensions and adaptations that have primarily concerned us. In this connection, we acknowledge a great debt to members of the Artificial Intelligence Laboratory who developed the ITS operating system and were extremely helpful in transferring it to us and tutoring us in its maintenance and use.

The upper tier of central software (Fig. 1) is divided into two main parts and an interface. The main parts are called "MUDDLE" and "CALICO". They are separate because two quite distinct and largely antagonistic philosophies have influenced our work. They are interfaced, and we are doing our best to meld the two into a functional unity, because users need -- and we want -- the best of both worlds. But we may defer the philosophies. Let us first complete our examination of the over-all plan.

MUDDLE is the part of the Dynamic Modeling System that is most helpful to a person who wants to formulate and explore ideas and to test them out quickly and conveniently. It is intended to become in addition, for many applications, the door into all the rest of the Dynamic Modeling System. Whereas LISP is limited to the single data structure LIST, MUDDLE offers several data structures. MUDDLE has, indeed, several other areas of advantage over LISP and appears likely to serve as a follow-on to LISP and as a basis for higher-level languages such as PLANNER and CONNIVER.

The design and implementation of MUDDLE have been (and are being) carried out as a joint undertaking of the M.I.T. Artificial Intelligence Laboratory and Project MAC.

"The CALICO World" is a system of programs intended to provide users with convenient ways of carrying out most of the basic information-processing tasks that arise frequently in interactive computing and, at the same time, with proven and documented building blocks for use in constructing application programs. "CALICO" is derived from the names of three of its main subsystems: the call-and-return mediator CARE, the library LIB, and the command interpreter COIN. Most of CALICO has been prepared in the assembly language MIDAS. It is consistent with a subset of PL/1. It emphasizes a small set of data types and structures compatible with those of MUDDLE

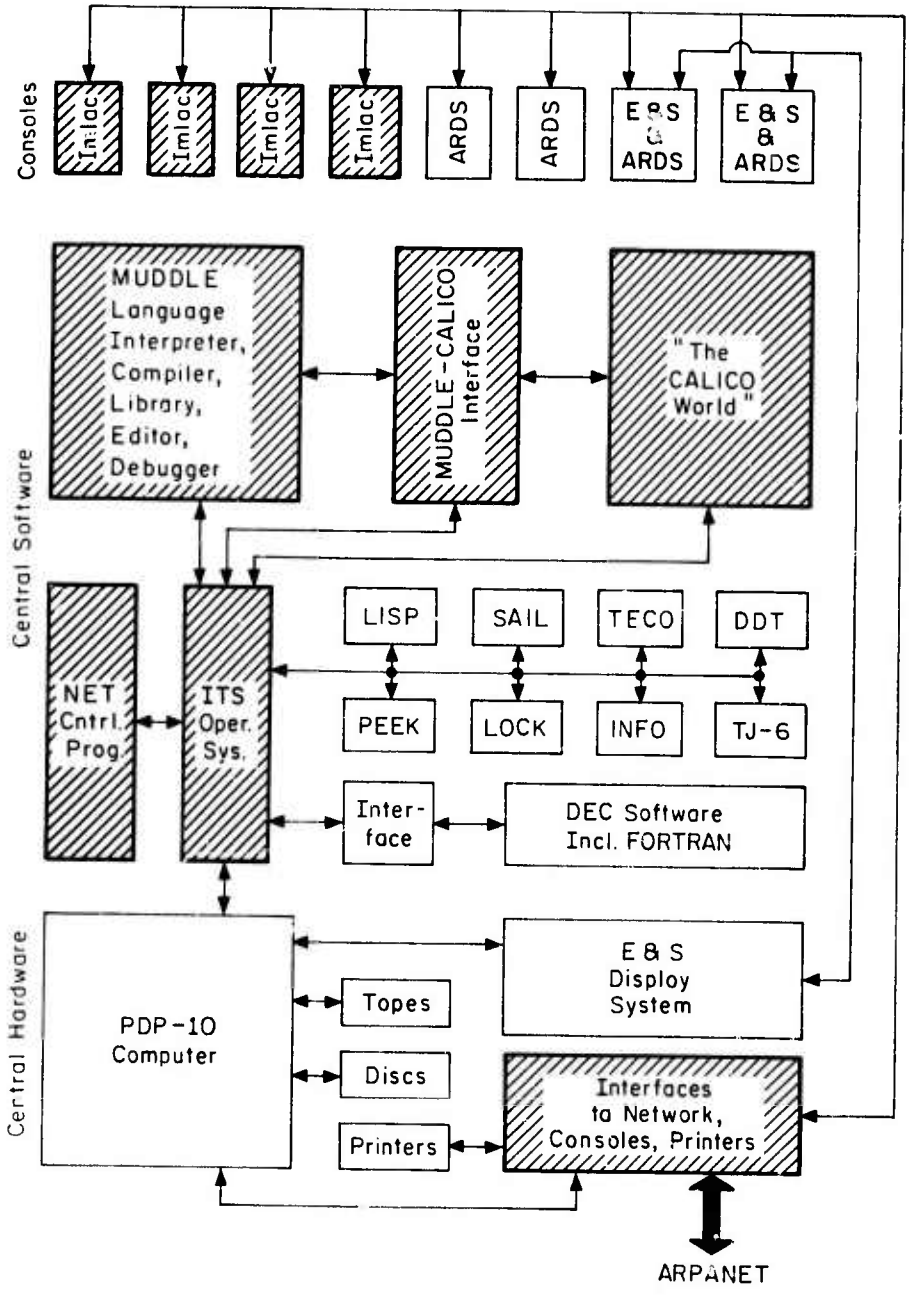


Fig. 1

General Plan of the DM System:

The heavy lines and cross-hatching indicate the main investments of effort in the system on the parts of the Dynamic Modeling, Computer Graphics, and Computer Networks Groups. See text and other figures for explication.

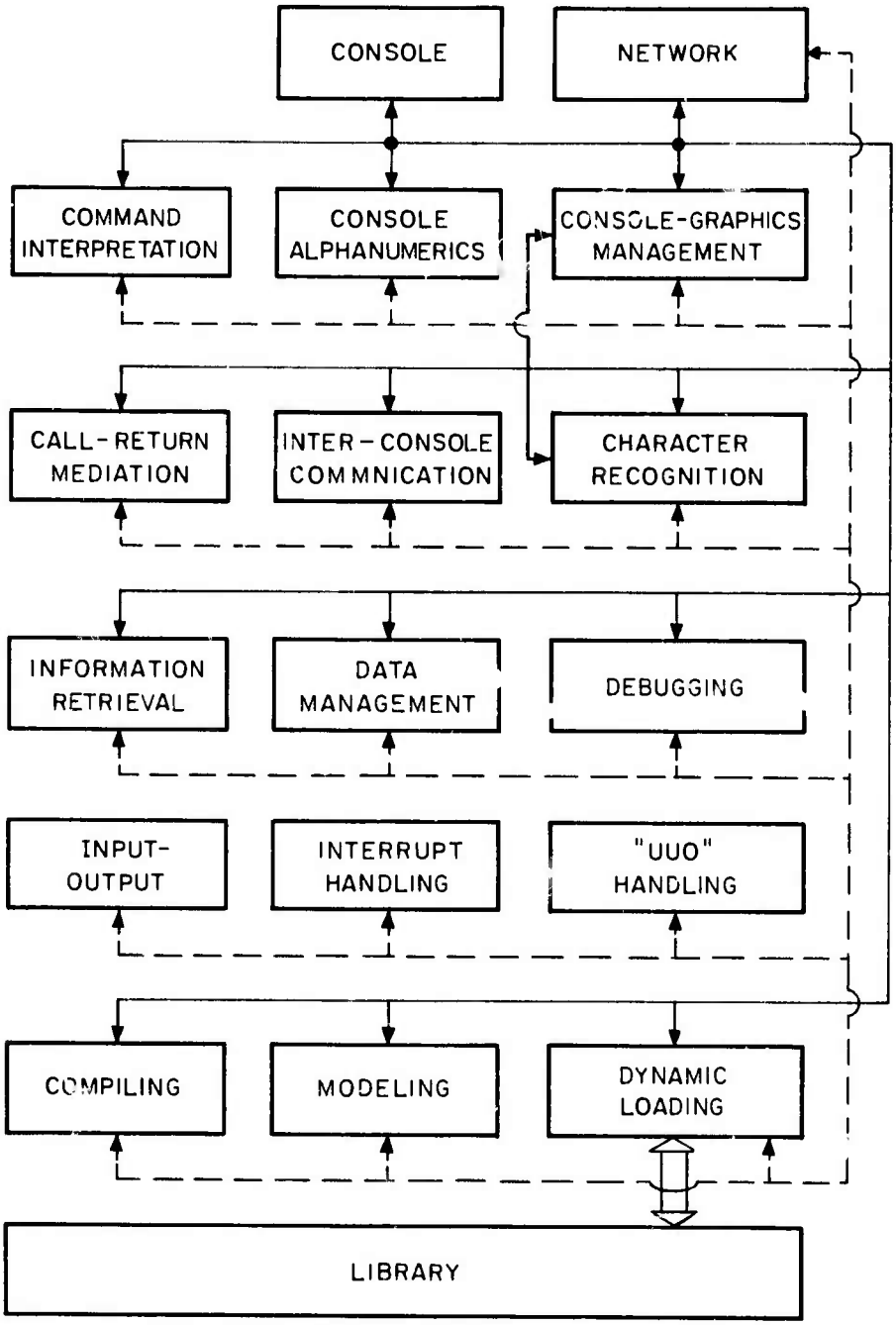


Fig. 2
"The CALICO World" and its major subsystems

but admits a wider range, laying down conventions and procedures for registering data types and structures. It defines several classes of subroutines and a preferred calling-and-returning sequence format, but it is hospitable to subroutines written in any language if they have been translated into loadable, callable, and executable PDP-10 code.

CALICO subsumes a set of facility programs and a library of modular subprograms. The facility programs handle the functions shown in Fig. 2. The library consists of about 500 subroutines (callable entries), each with documentation available (but vide infra) on-line through the consoles as well as in print-out form in files near the consoles. Most of the effort of the two-and-a-half groups has gone into CALICO, and it must be said at the present time to be the main part of the Dynamic Modeling System created by the effort.

HYDRA: As shown in Fig. 1, MUDDLE and CALICO are interconnected. MUDDLE functions can call subroutines from CALICO's library and have them operate on data that have been processed by MUDDLE, and -- although this is less useful -- CALICO subroutines can call MUDDLE functions and (if the data types and structures are among those dealt with by MUDDLE) have them operate on data that have been processed by CALICO. The interaction between MUDDLE and CALICO is at present rather inefficient and constrained, but it is rapidly being made more efficient and less constrained. We are hopeful that, within a few months, we shall have in operation a true realization of what three years ago we envisioned in a nebulous way but did not know quite how to achieve: a computer system providing both (a) the convenience and flexibility of interpretive execution in the upper echelons of program hierarchies and in any other areas in which procedures need to be readily modifiable and (b) the power and efficiency of assembly or compilation in all those parts of the programs in which off-the-shelf (i.e., out-of-the-library) subprograms can be used.

ITS: The operating system ITS, developed over a period of years by members of the Artificial Intelligence Laboratory, has proven to be excellent for our purposes. We have modified it only insofar as necessary to accommodate our hardware, to permit shared use of pure procedures and data, and to support operation of the Dynamic Modeling System's PDP-10 as part of the ARPA Network. It is important to note that ITS permits each user to have several processes, organized in a hierarchy, working for him concurrently. That is often a great convenience: the one-thing-at-a-time work style forced by single-process operating systems is not natural to most people, and there is great advantage (as in debugging a faulty program with a debugging aid such as DDT) in having programs able to communicate with each other yet not able to destroy each other.

Network Control Program: Partly under this heading and partly under "CALICO" there are several related programs that connect the Dynamic Modeling System into the ARPA Network. The Network Control Program proper, the Logger, the

implementation of the TELNET protocol, and an implementation of the new Data and File Transfer protocols have now been brought into line with the conventions of CALICO, and they consist mainly of library subprograms. The Network Daemon (independent ITS process), which earlier had to remain active continuously, now "wakes up" whenever it is called and "goes back to sleep" when it is finished.

Independent Modules: Not integrated with CALICO or MUDDLE but nevertheless of great value to users of the Dynamic Modeling System are the text editor TECO, the debugging aid DDT, and such utility programs as PEEK, LOCK, INFO, and TJ-6 borrowed with appreciation from the M.I.T. Artificial Intelligence Laboratory. Not integrated, and of less value to users because of it, are the language and compiler LISP, borrowed with appreciation from the M.I.T. Artificial Intelligence Laboratory, and the language and compiler SAIL, borrowed with appreciation from the Stanford Artificial Intelligence Laboratory. It will be very difficult to merge either LISP or SAIL into the MUDDLE-CALICO complex. We do not see much to be gained from merging LISP, but the idea of merging SAIL refuses to die because SAIL would bring with it access to a host of published ALGOL algorithms.

DEC Software: Last year, a joint MAC-AI effort developed a program that makes the ITS system look to the Digital Equipment Corporation's software like the DEC PDP-10 operating system. That program makes available to us the whole collection of DEC software, including FORTRAN, but does not, of course, make it coherent with MUDDLE or CALICO.

CENTRAL HARDWARE

The hardware base of the Dynamic Modeling System was described in the Annual Report of 1970-71 and will, therefore, be described only briefly here. The hardware base includes:

1	DEC PDP-10 main processor
1	E&S LDS-1 display processor
8	Blocks of core memory (208K words)
3	DEC RPØ2 disk drives (15M words)
1	DEC TU2Ø magnetic-tape unit (9-track)
8	DEC 555 microtape units
2	Bright BI1215 line printers
1	Interface to ARPANET IMP
1	Interface to consoles and printers

together with the necessary ports, channels, and interconnections. Most of the hardware was purchased from the Digital Equipment Corporation. Some was purchased from other manufacturers. The ports for about half the core memory and the interfaces to the IMP, the consoles, and the printers were constructed in the laboratory.

D. OPPOSITION AND CONFLUENCE OF TWO PHILOSOPHIES

In the preceding section, we mentioned that there are

two partly antagonistic philosophies in the world into which we are bringing the Dynamic Modeling System. It is to a large extent the product of their attractions and counter attractions, their pullings and haulings.

The one philosophy, embodied in various degrees in LISP, APL, BASIC, and now MUDDLE*, prizes the power and convenience that one can achieve by using a single, internally consistent language that provides the basic operators for working with a restricted set of basic data types and a way of composing higher-level operators -- procedures, functions, or subroutines -- and accumulating them within the system. Associated with this philosophy, but not an integral part of it, is the idea that each programmer-user should or will create his own, personal system of procedures, data and techniques, and that there is more to be gained through facilitation of such individual efforts than through attempting to organize one over-all system replete with standards, conventions, and a central software library, for use by, and to be contributed to by, an entire community of users.

The other philosophy, which we think has never yet been fully and successfully embodied in any actual computer system, focuses upon an image (which may be quixotic, yet must surely contain the germ of the fundamental plan of distant-future computer systems) in which most of the procedures that users need have already been written, tested, debugged, documented, and cataloged -- and in which users engage themselves more often, and more productively, in retrieving and using procedures than in creating (or recreating) them.** This philosophy was strong among developers and users of the Compatible Time-Sharing System, the files of which at one point contained one to two million words of public programs and 20 to 30 million words of private but to some extent sharable programs. However, no way of tightly organizing the CTSS user community was ever worked out, and although there were many cross-linkages among CTSS files, it must be said that CTSS only defined and clarified the philosophy and did not fully realize it. The philosophy of the community-wide coherent system has been strong in Multics, also, but -- as already noted -- most of the resources for Multics development went into providing an operational basis for sharing rather than into such things as or line facilities for retrieving library software. At present, the main thrust in the Multics world toward community-wide coherence is that of the Cambridge Project, which is developing a Consistent System

* * * * *

*It is dangerous to put these names into such proximity. The four languages/systems are different from one another in many ways, and proponents emphasize differences. Nevertheless, there is a strong common theme. Perhaps it is embodied more in the proponents than in the languages themselves.

**In the distant future, of course, users may describe the programs they want to a program-writing program rather than to a program-retrieving program. Automatic programming is to program retrieval as the calculation of functions of arguments is to table look-up.

for research and applications in the behavioral and social sciences.

Any effort to achieve community-wide coherence is sure to bring more than one language into the picture. It is more difficult to achieve coherence across languages than within one language. It is even more difficult to convert a BASIC fan to APL (or vice versa) or to get an old-time FORTRAN user out of FORTRAN and into the more modern world of PL/1 (or even ALGOL). Thus there are partly antagonistic technical and social forces at work as well as partly antagonistic philosophies. We have tried, as mentioned, to achieve the best of both worlds, but of course we have had to compromise -- for system engineering is almost always as much compromise as breakthrough.

To keep the problem within reasonable bounds, we defined a community of programmer-users much smaller than the Multics user community -- considerably smaller than the Cambridge Project's behavioral and social-science community within Multics. Sixty users have permanent files in the Dynamic Modeling System. Although the group includes representatives from physics, microbiology, neurophysiology, political science, urban planning, and nuclear engineering, most of the members are primarily computer scientists and speak the same languages. Almost all the participants have desks or offices in the same laboratory. Intercommunication is easy and there has been much of it. For an example, this spring there were 12 Special Interest Groups -- actually, planning-design-implementation task forces -- and one over-all Central Coordinating Committee. For another example, the working documents of the Dynamic Modeling System come to approximately 2,000 typewritten pages.

Even with the problem of achieving coherence thus scaled down, the difficulties have seemed very great. From the very beginning, the "managers" preached coherence, generality, documentation, and modularity-in-the-small while the creative drives of the "programmers" and the joyous positive reinforcement achieved through successful erection of ever-larger-and-more-elegant software castles swept the project in the opposite direction. (We put "managers" and "programmers" in quotes because they are roles, not people. A group leader alone at a console is a programmer and creates his own castle. A software specialist on the library "SIG" is a manager -- until he gets back to a console.)

The coherence and systematization that we have achieved was won at the cost of great effort, mainly group rather than individual effort, and always through several iterations that discarded much or most or all of the initial schema in order to adjust to new ideas or to concurrent interacting developments. The process has been hard on several proponents of initial schemata, and it has given the Dynamic Modeling System some of the nature of a patchwork quilt. Nevertheless, the process has brought the system close enough to the initial "dream" to let the dreamers feel that it is what they envisioned, and has come close enough to melding the partly

antagonistic philosophies that their antagonism is no longer a disruptive factor.

E. MUDDLE

As mentioned in the introductory comments in the OVER-ALL PLAN OF THE DYNAMIC MODELING SYSTEM, MUDDLE provides a very powerful and convenient environment for trying out ideas. One can create a fairly complex program, see what is right and what is wrong with it, and modify it and retest it -- or even throw it away and begin over -- relatively quickly. For example, a person with almost zero previous experience with MUDDLE (and LISP, to which MUDDLE is most closely related) prepared and debugged a descriptor-based information-retrieval system in 10 hours at the console plus 2 hours at his desk. In 10 more hours at the console he reprogrammed it in two quite different ways. The best version is now being combined with another MUDDLE retrieval system (somewhat more complex, programmed by a more experienced programmer in one afternoon), and the whole will then be redone in assembly or compiler code to operate efficiently under CALICO.

At present, MUDDLE is used almost exclusively as an interpreter, and at present it is too slow in operation and too demanding of memory space to handle large-scale tasks in a practical way. Even under those circumstances, however, MUDDLE is the preferred environment for trying out and testing ideas.

The people who have contributed intensicely to the design and development of MUDDLE are Gerry Sussman, Carl Hewitt, Jeff Hill, and Drew McDermott of the Artificial Intelligence Laboratory and Reeve, Daniels, Cressey, and Fleischer*. Daniels is developing a MUDDLE compiler, which will greatly increase efficiency of execution. It exists now in rudimentary form, and it functions, but it will not be ready for widespread use until late fall. MUDDLE's voracious consumption of memory space will become tolerable in December when the projected paging subsystem of the PDP-10 computer is installed.

The present MUDDLE library consists of 150 basic MUDDLE subroutines and 300 MUDDLE functions. The latter are simply the confluence of the personal MUDDLE files of Reeve, Daniels, Pfister, Broos, Metcalfe, Stern, Black, Cressey, Ryan, Rubin and Licklider, and the collection constitutes a mere beginning. It is a mere beginning because MUDDLE came into the planning of the Dynamic Modeling System rather late and because the ease with which one can program in MUDDLE tends to de-emphasize dependence upon a library. MUDDLE and its library are regarded, nevertheless, as very important parts of the Dynamic Modeling System.

With MUDDLE is associated an editor and debugging aid, MEDDLE (Pfister), that greatly facilitates use of MUDDLE. There is, also, a dynamic loader (Pfister, Cressey) of MUDDLE functions that retrieves them from disk files when they are

* * * * *

*For full names, see lists of members of Dynamic Modeling, Computer Graphics, and Computer Networks Groups.

called by other MUDDLE functions. Graphics capabilities are being incorporated into MUDDLE in two ways: one exploits the Evans and Sutherland processor and a graphics subsystem called "Polyvision" (Daniels, Michener, Black, Broos); the other uses display programs that operate in the Imlac consoles (Ryan, Rubin and Brodie).

F. CALICO

As shown in Fig. 2, CALICO includes subsystems concerned with almost all the various aspects on interactive computing. All the subsystems are built out of library subroutines, and the subsystems share subroutines insofar as possible. The library subroutines are either entirely pure procedures or are separated into pure and impure parts. Only one copy of a pure procedure (or pure part) is held in core memory, but there is, of course, one copy of each impure part for each user (using process) thereof.

MEDIATION OF INTERACTION

Interaction between the user-programmer at the console and the rest of the system is mediated mainly by the command interpreter (Seriff), the console alphanumeric subsystem (Cohen, Brescia, Peltan), and several console-graphics subsystems. The latter, more complex than shown in Fig. 2, include a display-management subsystem for the Evans and Sutherland display processor (Michener, Black, Geer and Curley), a subsystem for stylus-tablet input (Hui, Guertin, Broos, Niles), a subsystem for composing two-dimensional and three-dimensional graphs and charts (Black, Pangaro, Cutler, Siegel), and a subsystem for Imlac and ARDS graphics (Peltan, Conklin, Rubin, Ryan).

SUBROUTINE MEDIATION OF CALLING AND RETURNING

There are no "main routines" in CALICO -- only subroutines. Either a MUDDLE function or the mediator's initializer serves as the top-level routine. The subroutines of CALICO are of several kinds. On one dimension of variation are "fully mediated", "lightly mediated", and "unmediated". On a second dimension are "location-sensitive" and "location-insensitive", and on a third are "internal" and "external". The mediator CARE (Reeve, Harris, Hughett, Brodie, Licklider) handles the calling and returning of fully and lightly mediated subroutines, assuming control at times of calling and returning, performing several housekeeping functions on behalf of the calling, called, returning, and receiving subroutines, and affording the user-programmer an opportunity to intervene. Only low-level, inner-loop subroutines are unmediated, and only such routines (and only a few of them) are internal -- i.e., built into larger, higher-level routines. The governing philosophy calls for extreme modularity, for independence of each subroutine from all others except through calling and returning sequences, and for purity and location-insensitivity insofar as possible. Every fully mediated subroutine and every data set has a header that permits other software entities to discover its pertinent properties. The mediator is set up to

check the types of arguments at times of call and results at times of return; type checking is not yet in effect, but it will go into effect (on an optional basis) shortly.

COMMUNICATION

Interconsole communication is based on capabilities inherent in the ITS system and only slightly augmented for our purposes (Cohen, Brescia, Peltan). "System-mail" announcements and user-to-user messages are displayed at the time of logging in. If the intended recipient's on-line communication function is enabled, one user can type messages directly onto another user's display. An administrative file area and directory are used as a communication medium. And there is a users' directory and information-retrieval system (Guida) that contains names, addresses, telephone numbers, interests, project description and the like.

INFORMATION RETRIEVAL

At present, the main subsystem for retrieving information about programs and data sets (Bryan, Martin) operates in Multics and is used either through the ARPA Network or through a Multics console. (It was developed before the PDP-10 system had disk files.) It has turned out not to be very practical, because of log-in delays and processing costs, to use that subsystem as an aid in on-line programming. We are, therefore, reprogramming the system on the PDP-10. Two preliminary versions (Broos, Licklider) are now operating in MUDDLE. The design of the command interpreter and call-return mediator make it possible to stop at any point in the execution of a program, explore the library, and substitute a newly selected subprogram for one that is in line to be called -- or to substitute a newly selected data set for one that is in line to be processed. We hope to realize that capability in operation this fall.

DATA MANAGEMENT

The data-management subsystem consists at present of the "Keyed Data Manager" (Haverty, Lebling) and "Lexicontext" (Haverty). The former provides a means for associating retrieval labels (keys) with arbitrary units of data and then filing and retrieving the data in a uniform and efficient way. The latter provides a means for working with text in terms of basic lexical units (usually English words) and hierarchically structured concatenations (phrases, sentences, paragraphs, chapters) of them. It is much more straightforward and efficient to process Lexicontext's uniform representatives than it is to process strings of characters. Because the Dynamic Modeling System was conceived of as a general-purpose system, we have not developed a content-oriented data-management system. Even as our interests now are focusing on the field of automatic programming and debugging, it seems likely that we shall want to keep the data-management subsystem general, rather than field-specific, and rely for content sensitivity upon processing programs written in such languages as PLANNER

and CONNIVER.

DEBUGGING

The principal debugging aid is still DDT, which operates as a process separate from (superior to) the one that is being debugged. DDT is not a part of CALICO. We have augmented it somewhat (Cutler), but we have not tried to modularize and integrate it.

In an effort to aid the programmer in visualizing the execution of his programs and in seeing their flaws as the flaws are encountered, we have developed a PDP-10 simulator (Galley, Hughett) and a graphical debugging aid (Galley, Liu). Together, they display the flow of control through selected parts of a procedure and, at the same time, the flow and transformation of the data operated upon by the procedure.

A second graphical debugging aid is nearing completion (Hughett). It records the execution history of a program and then organizes and traces "trees of influence", working backwards, into the past, to check each operation that contributed to the shaping of an incorrect value.

Work is under way on other debugging aids that check the values of selected variables during the course of execution (Stern). Plans call for the integration of all these debugging tools (except DDT) and for their correlation with the mediator, which will make it possible for the programmer to turn them on and off at call and retrieve points during the execution of a program.

INPUT/OUTPUT, INTERRUPT HANDLING, AND UUO HANDLING

The input/output, interrupt-handling, and UUO handling subsytems include sets of CALICO macros and subroutines that are invoked by application programs and that, in turn, communicate with ITS. They include, also, the code required to handle certain common interrupt situations and to accept new UUOs. ("UUO" stands for "Unimplemented User Operations", which are codes that resemble PDP-10 instruction codes but are left for users to define.) These subsystems allocate resources dynamically, insulate application routines from such absolute qualities as channel numbers, and generally facilitate applications programming. They were designed and implemented by Haverty, Broos, Reeve, Seriff, Long, Lebling, and Hughett.

COMPILING

The only compiler that we have thus far made consistent with CALICO is the PL/1 subset compiler mentioned earlier (Okrent, Sybalski). It translates statements from the subset of PL/1 into MIDAS source language and accepts all the macros and respects all the conventions of CALICO.

SPECIAL TOOLS FOR MODELING

In the philosophy of modeling that we have tried to

implement, it is not held desirable to restrict oneself a priori to the context or "world view" of any single modeling or simulation language such as SIMSCRIPT, GPSS, SIMULA, or DYNAMO. Ideally, we would like to have the good features of many such languages brought together within one consistent framework (such as MUDDLE or PL/1). In fact, we have devoted our energies mainly to erection of the framework, and we have not completed a kit of tools of the kind (e.g., event schedulers, queue managers) that are features of the simulation languages mentioned. Work on several such tools is being done by Okrent, Fox, and Weissberg. A major aim in this work is to facilitate the construction of models that include both continuous-process and discrete-event components.

DYNAMIC LOADING

In CALICO, the top-level subroutine is called through the mediator either by a MUDDLE function or by the user at his console. If it is already in core memory, the subroutine is executed; if it is not already in core memory, it is loaded by DYNAL (Reeve, Brodie) and then executed. Thereafter, each called subroutine that is not in core is loaded dynamically and then executed. The called routines are sought by DYNAL first in the user's files, then in a recent-accessions file, and finally in the main library.

THE LIBRARY

The part of the Dynamic Modeling System into which the greatest amount of effort has been put -- and the part on which the success or failure of the "community" philosophy rides -- is the library. At present, there are about 500 subroutines in the library. They range from simple, low-level subroutines that "get" and "put" characters or symbols to complex, high-level subroutines that translate programs or display three-dimensional graphs of hand-printed functions. Corresponding to each executable library subroutine there is a source-language listing that includes an abstract that can be retrieved on-line and contains all the information required by a programmer-user to use the subroutine successfully.

G. DOCUMENTATION

An over- all view of the Dynamic Modeling System and information about how to get started are provided by Galley's "A Novice's Introduction to the System". It deals with CALICO, ITS, the utilities, and the hardware rather fully but leaves to another document a comparable introduction to MUDDLE. That document is Pfister's "A MUDDLE Primer", which permits anyone with basic knowledge of programming to move very quickly into MUDDLE programming.

Essential technical information about the PDP-10 computer is collected in the manufacturer's System Reference Manual. Full descriptions of the ITS operating system, the MIDAS assembler, and several utility programs are contained in reports of the M.I.T. Artificial Intelligence Laboratory. The DEC and AI Laboratory sources cover the foundations of the Dynamic

Modeling System. The superstructure is described in a rather voluminous set of documents prepared by members of the Dynamic Modeling, Computer Graphics, and Computer Networks Groups. At present, a Dynamic Modeling System Manual is being prepared on the basis of these documents.

The set of documents is subdivided into several classes:

GA	General and Administrative
SYS	Basic System
SR	Subroutines
MCR	Macros
UUO	"Unimplemented User Operations"
DTCS	Data Types, Classes, and Sets
FA	Form Abstracts
LA	Listing Abstracts

Convention II:

Sixteen of the 23 General and Administrative documents deal with a system of standards, protocols, and conventions called "Convention II". It sets forth rules intended to foster uniformity and coherence in those areas of software system development in which many minor decisions must be made among alternatives that differ only slightly in utility. Decisions about format are often of that kind: it doesn't matter much which format is adopted for bibliographic references, for example, but it is considerate of the reader to make things consistent. We note, however, that the subject of software conventions is one in which people have deep ego-involvements, the depth of involvement being related inversely to the importance of the issue.

It took many hours of argumentation to define Convention II, and it would have taken many more hours of policing to enforce it if it were not for one fact: the documentation is on-line, and uniformity of format makes it possible for computer programs to deal with documents. There is now quite a collection of subroutines and macros that extract the titles, one-line descriptions, descriptor sets, and so on from listing abstracts. Broos' TECO macros run through the library of source programs and update the Listing Abstract Book at regular intervals. Lebling's macros count each user's files and measure the disk space they occupy, and the posted results apply strong moral pressure on "disk hogs" to delete the least active items. And so on. It is obvious now that it is very important for programs to be able to know a lot about programs and we want to move toward having programs understand documentation. Format is of course only one of many facets of software that programs should know about, but it holds some of the keys to learning about the other facets.

Subroutines: For most low-level subroutines, the listing abstract -- usually about two typewritten pages in length -- provides adequate documentation. For high-level subroutines, however, the problem is different because each high-level subroutine organizes the work of several or many low-level

subroutines and, in effect, is a complex subsystem. Protocol calls for a full program write-up for each such subsystem.

Among the principle subsystem of the type under discussion are:

Name	Function	Author	Identification
PARSE	General parser	K. Brown	SR.12.Ø1
OBDIS	Object display handler	Hui	SR.15.Ø1
CHAREG	Character recognizer	Hui	SR.15.Ø5
GRAPH	Cartesian graphs, 2D & 3D	Black	SR.15.Ø6
ESKBD	Character strings on display	Broos	SR.15.Ø9
PENGET	Stylus-tablet input	Michener	SR.15.13
MATTER	Display of 2D & 3D graphs corresponding to hand-printed functions	Guertin	SR.15.14
PREORD	Binary trees	Okrent	SR.16.Ø1
ATTACH	Compression of machine-language files	Daniels	SR.18.Ø1
DATPGE	Data paging for disk files	Haverty	SR.18.Ø1
KDM	File manager	Haverty	SR.19.Ø2
OLDS	On-Line document system	Bryan and Martin	SR.21.ØØ - SR.21.12

Other Categories: Macros, UUOs, and other categories of software are documented in approximately the same way as subroutines, but the effort to get them and keep them under bibliographic control has not been as strenuous. Documents describing macro and UUO subsystems are listed in a periodically updated Table of Contents of the Dynamic Modeling Documentation Series.

Abstracts: The most important item in the documentation system has turned out to be the listing abstract. It is part of the source-language listing of each subroutine or data set, and no subroutine or data set can get into the library without a listing abstract. There are also other abstracts, called "form abstracts", that a programmer prepares on printed forms when he begins to work on a program. Because they are available on line (as well as off), listing abstracts have become the mainstay.

H. MODELING

Although the main effort of the past three years has been to build a system to facilitate modeling, we have engaged (and are engaging) with colleagues outside Project MAC in several substantive modeling efforts. These include a model of a neuronal network (Bingham, Jarvis, Reeve, Balin), an air-

traffic-control simulation (McMillan), a simulation of airport air traffic (Dosett), an urban development simulation (Professor Aaron Fleischer, Allaman), a political science simulation (Lebling), a model of a neural time-to-place transformation (Professor Jerome Lettvin, Paul Pangaro), and a model of the protein-building process that involves RNA (Professor Alex Roch, Siemeus). The last-mentioned effort, just getting under way, involves an IBM 360-91 computer at UCLA as well as the Dynamic Modeling System; the 91 will handle the "number crunching" and the Dynamic Modeling System the interactive part of the work, which will exploit quasi-3D graphic display.

MODELING, GRAPHICS, NETWORKS

Publications

1. Brescia, M. F., Time-Sharing a Display Processor for a Time-Shared Computer System, S.B. Thesis, Department of Electrical Engineering, M.I.T., February 1972.

2. DeTreville, J. D., Design and Implementation of an Example Operating System, S.M. Thesis, Department of Electrical Engineering, M.I.T., 1972.

3. Donovan, J. J., Systems Programming, McGraw-Hill, New York, 1972.

4. Donovan, J. J., Teacher's Manual to Accompany Systems Programming, McGraw-Hill, New York, 1972.

5. Fleischer, R. J., Computer-Aided Visual Analysis of Statistical Data, S.M. and S.B. Thesis, Department of Electrical Engineering, M.I.T., August 1971.

6. Galley, S. W., A Novice's Introduction to the System, DM/CG/CN Group Memo, SYS.00.00, Project MAC, M.I.T., March 1972.

7. Haverty, J. F., Lexicontext A Dictionary-Based Text Processing System, S.M. and S.B. Thesis, Department of Electrical Engineering, M.I.T., August 1971.

8. Hui, W. F., An Algorithm for Detecting the Intersection of Arbitrary Computer-Displayed Figures, S.B. Thesis, Department of Electrical Engineering, M.I.T., June 1972.

9. Konig, D., Evaluation of PL/I as an Interactive Language, S.M. Thesis, Department of Electrical Engineering, M.I.T., 1972.

10. Licklider, J. C. R., "Criteria for Design and Evaluation of Information Networks", SIGIR, ACM, Las Vegas, Nevada, November 1972.

11. Madnick, S. E., "Program Parallelism Based upon Computation Schemata", Sixth International Congress on Cybernetics, 1972, pp. 480-494.

12. Madnick, S. E., Storage Hierarchy Systems, Ph.D. Thesis, Department of Electrical Engineering, M.I.T., 1972.

13. Metcalfe, R. M., Anderson, et al., "The Data Reconfiguration Service -- An Experiment in Process/Process Communication", Proceedings of the Second Symposium on Problems in the Optimization of Data Communications Systems, Stanford University, October 1971.

14. Metcalfe, R. M., S. Crocker, et al., "Function-Oriented Protocols for the ARPA Network", SJCC, 1972.

Publications (continued)

15. Metcalfe, R. M., "Strategies for Interprocess Communication in Distributed Computing Systems", Conference on Computer Communication and Teletraffic, April 1972.

16. McCray, W. A., SIM360: A S/360 Simulator, S.B. Thesis, Department of Electrical Engineering, M.I.T., 1972; also MAC TM-30, May 1972, AD 749-365.

17. Okrent, H. F., Design of the Dynamic Modeling Manager, S.M. Thesis, Department of Electrical Engineering, M.I.T., June 1972.

18. Pfister, G., A MEDDLE Manual, DM/CG/CN Group Memo, SYS.11.02, Project MAC, M.I.T., January 1972.

19. Pfister, G., A MUDDLE Primer, DM/CG/CN Group Memo, SYS.11.01, Project MAC, M.I.T., July 1972.

20. Seiferas, J. I., Observational Complexity of Character Strings, S.B. Thesis, Department of Electrical Engineering, M.I.T., 1972.

21. Zaborowski, S., Monitoring Student Programs -- A Case Study in Software Reliability, S.B. Thesis, Department of Electrical Engineering, M.I.T., 1972.

Talks

1. Donovan, J. J., "Memory Management", presented as part of the Yale Fall Lectures in Computer Science, 1971.

2. Donovan, J. J., a series of 8 lectures presented at the University of Pittsburgh, Spring 1972.

3. Licklider, J. C. R., "The Concept of an Interactive Technical Journal", Annual Symposium on Information Retrieval, May 1972.

4. Licklider, J. C. R., "ADP Technology", Department of Defense Computer Institute, June 1972.

5. Licklider, J. C. R., "Future of Library and Information Systems", National Academy of Sciences, June 1972.

6. Metcalfe, R. M., "The ARPANET', ACM/SIGREAL, Cambridge, Mass., March 1972.

7. Metcalfe, R. M., "The ARPANET", Stanford Digital Systems Seminar, May 1972.

MODELING, GRAPHICS, NETWORKS

Publications In Progress

1. Donovan, J. J., Research & Pedagogical Aids for Programming, M.I.T. Press, Cambridge, Massachusetts, publication planned for 1972.

2. Donovan, J. J. and S. E. Madnick, Operating Systems, McGraw-Hill, New York, publication planned for 1973.

3. Donovan, J. J. and S. E. Madnick, "Models of Operating Systems".

4. Donovan, J. J. and S. E. Madnick, "Thrashing in an Operating System".

5. Donovan, J. J. and S. E. Madnick, "System Performance and Evaluation".

EDUCATIONAL COMPUTING SYSTEMS

Prof. J. Weizenbaum

Academic Staff

Prof. M. L. Dertouzos

Instructors, Research Associates, Research Assistants and Others

R. H. Brown
D. L. Isaman
P. Jessel
P. A. King
C. Lynn
C. Mah
J. R. Stinger
L. Tsien
S. A. Ward

Support Staff

M. J. Connell
A. M. Garrity
N. J. Robinson
L. E. Yaple

Guest

Prof. J. Berger

EDUCATIONAL COMPUTER SYSTEMS

A. Present Work

During the period July 1971 to May 1972, the group continued several research activities which were initiated before this period, and are discussed below. Halfway through this period, Prof. M. L. Dertouzos and his students came from the Electronic Systems Laboratory to Project MAC and joined the group. During that time, relatively few new activities were initiated partly because of extensive involvement of the faculty in the development of 6.252 and partly because, with the departure of Prof. J. Weizenbaum for a two-year period starting in the Fall of 1972, the group intends to re-orient its objectives and scope of research.

Continuing research entailed the following:

1. Study of means for making possible the animation of dynamic processes for educational purposes. In particular, several approaches were investigated for the interconnection of a simulated process (a simple two address machine) to a graphical process (e.g., representations of registers and a processor stack). It is intended that this interconnection be flexible enough to allow on-line editing of the graphical representations independently of the simulated machine. Supervision of this work was initially under Prof. J. Weizenbaum and has shifted to Prof. M. L. Dertouzos a few months ago.

2. An associative memory higher-level interpreter. This work involves exploration of the use of an associative memory with a higher-level language hardware interpreter. The objective is to match the hardware memory and processor to the environment and interpreter structures of a higher-level language such as LISP. Such a match which appears to be economically feasible with current technology, eliminates the need for the typical machine language and numerical memory location issues which are quite distinct from the user's language. This, in turn, means the absence of assembly, compilation, relocation, storage allocation, and address computation tasks in their traditional sense. The elimination or reduction of these problems, in turn, frees us to study more fundamental notions concerning procedure semantics and computing environments. This work is supervised by Prof. M. L. Dertouzos.

3. Two Ph.D theses being completed June 1972 and September 1972 are carry-overs from the research of Prof. M. L. Dertouzos in Computer-Aided Design. One thesis involves the study of exact-inexact machines for the solution of large systems of equations. These machines consist of a digital relatively accurate exact part and an analog or coarser digital inexact part. Programming the machines entails decomposing the problem into a part, which checks a proposed solution with a computing time that grows relatively slowly with problem size -- this is handled by the exact part of the machine; and another part which proposes a new solution on the basis of the above check -- this part which would normally require a large amount of computing time is done rapidly by virtue of the inexactness of the subsystem that handles it. The overall solution is as exact as the exact part of the machine.

This research has resulted in many interesting results which

Preceding page blank

are documented by Jim Stinger's Ph.D. Thesis. The work is supervised by Prof. M. L. Dertouzos.

4. The other Ph.D. thesis (P. Jessel) involves the development of a "language" which describes existing and hopefully new circuit analysis programs. Expressing a circuit analysis program in this language leads to the ability of measuring with acceptable accuracy the power of that program, e.g., the computing time it will require for various classes and various sizes of circuits, the amount of storage used and so on. These measurements, in turn, make possible a comparison of the (many) existing and proposed circuit analysis programs in a systematic way. Several interesting results have been obtained to date and more are expected during the summer. This work is supervised by Prof. M. L. Dertouzos.

5. A thesis completed under the supervision of Prof. J. Weizenbaum involves the development of simple primitive functions for natural language. These are based on a lay-word transformational grammar distinguishing English function words and a canonical representation of English sentences. The functions provide the general capability for saving information for answering questions, for executing commands, and for doing elementary deductions. They are designed to facilitate the construction of special-purpose computer natural language systems, with the particular application to a program for generating computer graphic displays for plane geometry.

B. Future Work

1. The 6.252 Computer System

Academic activities will continue to occupy a substantial part of the time spent by several members of the group. These activities involve continuation of the development of 6.252 and the development of a 6.252 time-shared system. The latter, based on a PDP-11/45 CPU, is a limited-objective 8-user system (no swapping, one file per user) which will offer to students a subset of the PDP-11 assembly language and several LISP interpreters with progressively more complex environment and control structures. This system, which was purchased by the Electrical Engineering Department, will be used exclusively by 6.252 students who will be expected to use it for an average of 2 (console) hours per week.

It is also expected that the system will be used to implement the on-going research work on graphics (1 above), for the purpose of demonstrating to students either on-line or through movies, several dynamic processes, e.g., (i) stack operations during execution of recursive procedures, (ii) implementation and consequences of global labels and free variables, and (iii) garbage collection algorithms.

2. Continuation of Present Work

Of the present work, it is expected that items A.1. and A.2. above will be continued.

3. Exploration of New Areas

As part of next year's activities, the group wishes to explore the area of "engineering-robotics". By that term we mean the use of computers in connection with several sensors and actuators for the execution of engineering-oriented tasks. As a prime example, consider the current (and past) area of automatic control where the principal "processor" is an electronic circuit -- the so-called servo. We believe that in the foreseeable future, the majority of servos will be implemented by programmable chips, and the task of control engineers will be to write appropriate procedures. An example of work that has to be done in this area involves the computer-physical world interface, e.g., the development of an approach (a language and a hardware organization) for handling the demands imposed by the interaction of the machine and the outside world. Specifically, the simultaneous handling of synchronously and asynchronously serviced sensors and actuators cannot be done conveniently with the I-O techniques developed in the past (e.g., interrupts), which were motivated and developed in the context of a man-machine interface.

In addition to exploring this problem, the group would like to explore other possible applications areas, where engineering robotics can be studied and developed. Our intent is to use some of the excellent results obtained to date by the Artificial Intelligence people, but with an engineering philosophy, i.e., to some extent, subordination of "first principles" to the expediency of achieving specific tasks. As a result of this exploration, we would like to arrive at one or more applications areas that deserve further work, or to the conclusion that we should not continue work in this direction.

EDUCATIONAL COMPUTER SYSTEMS

Publications

1. Dertouzos, M. L., "Time Bounds on Space Complexity", Proceedings of the Switching and Automata Theory Symposium, October 13-15, 1971, pp. 182-187.

2. Dertouzos, M. L., et al., "Insight vs. Algorithms", IEEE Transactions on Education, Vol. E-14, No. 4, pp. 164-169.

3. Dertouzos, M. L., "Elements, Systems, and Computation: A Five Year Experiment in Combining Networks, Digital Systems and Numberical Techniques in the First Course", IEEE Transactions on Education, Vol. E-14, No. 4, pp. 197-201.

4. Dertouzos, M. L., G. P. Jessel and J. R. Stinger, "Circal-2: General-Purpose On-Line Circuit Design", Proceedings of the IEEE, Vol. 60, No. 1, pp. 39-48.

5. Dertouzos, M. L., M. Athans, R. N. Spann and S. Mason, Systems, Networks, and Computation, McGraw-Hill, New York, May 1972.

MATHLAB

Prof. W. A. Martin

Prof. J. Moses

Instructors, Research Associates, Research Assistants and Others

R. J. Fateman
M. R. Genesereth
J. Kok
S. M. Macrakis
E. C. Rosen
S. E. Saunders
R. M. Seigel
B. M. Trager
E. Tsiang
P. S-H Wang
D. Yun
R. E. Zippel

DSR Staff

M. J. Ablowitz
R. A. Bogen
H. O. Capps
J. P. Golden
L. P. Rothschild
R. Schroeppel
D. C. Watson
J. L. White

The past year has been a year devoted mostly to consolidating our previous work. The "Mathlab" PDP-10 arrived during the year and became operational in February 1972. Scores of casual sessions with the MACSYMA system have been held. Users were increasingly satisfied with MACSYMA as more capabilities were integrated into it and as the system, now approaching 150K words, became better debugged.

A. The Hardware and Time-Sharing System

The basic PDP-10 processor and one half its memory (i.e. 128K) arrived in October 1971. Richard Greenblatt and Thomas Knight made an heroic effort to bring up an ITS time-sharing system compatible with that on the Artificial Intelligence Group's machine. The new file system they created was given to the Dynamic Modeling group and tripled their effective disk utilization.

The remainder of the memory arrived in December. In January Systems Concepts installed a pager compatible with the one on the Artificial Intelligence machine. As a result, the Mathlab machine is able to use the latest versions of the Artificial Intelligence time-sharing systems. Some thrashing originally encountered when three or more MACSYMA's were being run on the Mathlab machine caused a redesign of the scheduler. As a result of the compatibility of the software, the new scheduler has been used on the Artificial Intelligence machine as well.

B. Consolidation of MACSYMA Software

The past year was spent mostly in debugging and consolidating existing code into the MACSYMA system. The presence of the new machine meant that more users were available to report on bugs, and two or more versions of MACSYMA could be run and debugged at the same time.

While MACSYMA was on the Artificial Intelligence machine, we were restricted to a system of about 90K words. This meant that several subsystems of MACSYMA were left on the disk,

The latest versions of MACSYMA contain almost all of the code and a little larger working storage. These versions approach 150K in size. Over 60% of the system is pure and is shareable by several users. Recently we have run five simultaneous versions of MACSYMA.

We decided to avoid having separate user and experimental versions for a while longer. The current version of the system has a great number of debugging aids, in particular the ability to trace any function in the system. The effect of the ease of debugging is a slowdown of the system of a factor of about four. We have started to create a first "release" version of the system which would not have the debugging aids. The timing studies attempted thus far for the release version have uncovered a number of surprises, as seems to be common in systems of this size. One common style of LISP programming is to call a function indirectly through its name which is kept on the property list of an atom. This technique seems to slow

Preceding page blank

down a function call by a factor of at least 50. With the present maturity of the system the modularity of the indirect call is not required, and we expect an increase in speed in subsystems which rely on this technique. Another effect of the shift in coding style will be that the amount of impure data in the system will be decreased noticeably.

C. New Subsystems

Paul Wang completed his doctoral research on definite integration last summer. This subsystem together with the limit routines comprise 15% of all of our code. It is being integrated into MACSYMA at the present time.

During the past year, we completed work on a polynomial factorization program based on Berlekamp's method for factorization modulo a prime. Drs. Linda Rothschild and Paul Wang did the bulk of the coding with some help from Dr. Richard Fateman. The present program is the only one which can factor polynomials in several variables efficiently. Special case techniques, such as for polynomials of the form $X^n \pm 1$, have been implemented in the program.

Richard Zippel, a sophomore in mathematics, has almost completed a subsystem for the manipulation of power series. The representation of power series is different from that of polynomials in that the coefficients are rational functions rather than polynomials. The subsystem allows the user to declare cutoffs such as $X^9 \rightarrow 0$ so that powers of a variable higher than a given degree are neglected. This is a desirable capability because it keeps the intermediate expression swell down in certain applications.

D. The New LISP Compiler

Eric Rosen, with the help of Jeffrey Golden and Jon White, completed a new version of the LISP compiler which compiles purely arithmetic code better than the current DEC PDP-10 FORTRAN compiler. A new version of LISP which handles arithmetic function efficiently was designed by William Martin, Jon White and Eric Rosen. MACSYMA will begin using the new LISP system and the new compiler in the coming year.

E. CONNIVER

A new language, called CONNIVER, has been implemented in LISP by Gerald Sussman and Drew McDermott of the Artificial Intelligence Laboratory. The central features of this language were suggested by Joel Moses. This language is intended for use in Artificial Intelligence applications and also in the Mathlab group's automatic programming efforts. The language has the capability of handling local data bases which are erased after function returns. The language has some of the features of PLANNER such as pattern matching, but lacks the automatic backup which PLANNER utilizes. We believe it to be superior to PLANNER for most applications.

F. The MACSYMA (maximus?, maximum?) Experience

Successful use of MACSYMA has increased markedly during the past three months. While many bugs still exist and the MACSYMA version is liable to change from moment to moment, the frequency of complaints about bugs has decreased to the point that some console sessions do not encounter bugs in several hours of use.

A console connected to the Mathlab machine is operating in the Mathematics department. Richard Fateman has used the console to solve a number of problems for the Applied Mathematics faculty. In particular, he solved certain integration problems, systems of linear equations with parameters and nonlinear differential equations. We believe that at the present time successful use of MACSYMA is best achieved by a "hand-holding" operation in which someone familiar with MACSYMA solves the problem on the computer. Should the system be sufficiently attractive to the Mathematics faculty, then they will want to invest the time to learn how to use it themselves. It should be noted that much of the faculty does not know how to use a computer at all. One has already familiarized himself with MACSYMA.
A group at the Jet Propulsion Laboratory has been using MACSYMA by making calls through the Federal Telecommunications System. The line quality through FTS is quite poor. This group is, however, too impatient to wait for the ARPA connection, which should occur in the summer. The group has a contract to generate integrals of hypergeometric functions which they hope to do with MACSYMA.

A doctoral student of Prof. G. C. Rota has been using MACSYMA to investigate simplification properties of linear operators. Because of the large size of his expressions, he has used a tailor-made 200K version of the system.

Many members of Project MAC and the Artificial Intelligence Laboratory have made casual use of MACSYMA for solving homework problems and the like. At present, we do not keep track of successful usage, but we tend to be quickly informed of the need to fix bugs and improve certain aspects of the system.

G. Future Directions

Future efforts of the group will be concentrated in three directions.

1. Improvements to the Current System

The current system was written in less than 3 years by essentially 4 people (Fateman, Martin, Moses, Wang). This has led to a system in which a number of useful features were ignored in order to concentrate on others which were deemed more essential to the total effort. The list of improvements includes:

a) A better representation of matrices -

Three representations exist at present. Paul Wang is trying to consolidate them.

b) A more efficient input parser -

The current parser is an LR(1) parser. An operator precedence parser is being designed by Steve Saunders.

c) Better storage mechanisms for intermediate results -

Several schemes for storing intermediate results of a long session have been attempted in the past without much success. Jeffrey Golden is working on this project.

We are also interested in integrating into MACSYMA significant capabilities available in other algebraic manipulation systems (e.g. the physics package in REDUCE-2 and Laplace transform package in MATHLAB).

It is clear that the large number of users of MACSYMA which we expect in the coming year will suggest a variety of new projects for the group.

2. Subsystems for Important Areas of Application

In trying to create a general-purpose system, we have largely ignored special capabilities desirable in only one or two areas. Areas which we have been examining are:

a) Numberical Analysis -

One important use of an algebraic manipulation system is as a front end to numerical analysis programs. David Yun has used MACSYMA to set up an efficient numerical solution to partial differential equations for members of the Civil Engineering and Mathematics departments.

b) Optimization and Control

David Yun has also explored the use of MACSYMA in problems in optimization and control which involve parameters and hence cannot be easily solved by numerical techniques.

c) Perturbation Theory

Jeffrey Golden has been examining the possibility of implementing techniques from perturbation theory in MACSYMA. With Dr. Mark Ablowitz he studied the stability of a classical problem in applied mathematics last summer.

d) Astronomy

The most spectacular achievements of algebraic manipulation have occurred in astronomy. Dr. Andre Deprit recently received the only award in computer science given by the National Academy of Science for a recalculation of the moon's orbit. This calculation originally

took 20 years in the 19th century. Deprit found one mistake in it. While Deprit used a tailor-made system, we believe a more general subsystem of MACSYMA could be written for doing such a calculation and many others with reasonable efficiency.

e) Number Theory and Combinatorics

Although MACSYMA was originally designed to be of use to applied mathematicians, we have had a significant number of users interested in pure mathematics, in particular number theory and combinatorics. We would like to extend our capabilities in these areas.

3. Research on New Algorithms

Algebraic manipulation systems have up to now relied on the user to provide a method for solving his problem. Only in rare cases, such as integration, have these systems used an approach far different from the traditional "paper and pencil" approach of people. We believe that it may be possible to obtain a new set of algorithms of great power and generality which will solve the traditional problems of applied mathematicians.

The approach we have in mind is to start with the original model of a system in terms of differential or difference equations and then ask the questions directly of this model. The traditional approach has been to try to solve the system in closed form in terms of a class of special functions (e.g. exponentials, Bessel funtions) and then ask the question (e.g. stability) of the solution. As is well known, few systems possess closed form solutions. Another popular approach has been to get an approximation to the solution in terms of special functions, and then an answer based on the approximation. This method frequently fails when the system is nonlinear.

We are very optimistic about the possibility for a theory or theories about important properties of differential or difference equations. Recent results in algebraic geometry and combinatorics have shed great light on properties such as integration and identities (i.e. simplification rules). There is some hope that results on stability are forthcoming as well. The current situation is described in Joel Moses' forthcoming paper in the 25th Anniversary Issue of CACM.

H. SIGSAM

In June 1971 Joel Moses and William A. Martin were elected Chairman and Treasurer, respectively, of the Special Interest Group on Sympolic and Algebraic Manipulation of ACM. They will serve in these posts until July 1973.

MATHLAB

Publications

1. Fateman, R. J., Essays in Algebraic Simplification, Project MAC, M.I.T., MAC-TR-95, April 1972, AD 740-132.

2. Fateman, R. J., "Rationally Simplifying Non-Rational Expressions", SIGSAM Bulletin, No. 23, July 1972, pp. 8-9.

3. Martin, W. A., "Determining the Equivalence of Algebraic Expressions by Hash Coding", Journal of the ACM, Vol. 8, No. 4, October 1971, pp. 549-558.

4. Moses, J., "Algebraic Simplification: A Guide for the Perplexed", Communications of the ACM, Vol. 14, No. 8, August 1971, pp. 527-537.

5. Moses, J., "Symbolic Integration: The Stormy Decade", Communications of the ACM, Vol. 14, No. 8, August 1971, pp. 548-560.

6. Moses, J., "Toward a General Theory of Special Functions", Communications of the ACM, Vol. 15, No. 7, July 1972, pp. 550-554.

7. Wang, P. S., Evaluation of Definite Integrals by Symbolic Manipulation, Ph.D. Thesis, Department of Mathematics, M.I.T., 1971; also MAC-TR-92, October 1971, AD 732-005.

8. Wang, P. S., "Application of MACSYMA to an Asymptotic Expansion Problem", Proceedings of the ACM 25th Annual Conference, August 1972, pp. 844-850.

Publications In Progress

1. Fateman, R. J., and J. Moses, "Canonical Forms for First Order Exponential Expressions".

2. Fateman, R. J., "Comments on Problem 2".

3. Fateman, R. J., "On the Computation of Powers of Polynomials".

4. Wang, P. S., "Factoring Multivariate Polynomials Over the Integers".

5. Yun, D. Y. Y., "An Application of MACSYMA to Proving the Achievability of the $\lceil \frac{n+1}{2} \rceil M$ for Evaluation of General Non-monic Polynomials of Degree n".

6. Yun, D. Y. Y., "On the Efficiency of the Dijkstra Algorithm".

7. Yun, D. Y. Y., "On Symbolic Solutions of Systems of Algebraic Equations".

Talks

1. Moses, J., "Algebraic Manipulation", a set of lectures given at the Computer Science Seminar, Bonn, Germany, August 1971.

2. Moses, J., "Algebraic Manipulation", given at the Joint Colloquium of the Departments of Mathematics and Computers & Communications, University of Michigan, Ann Arbor, Michigan, November 1971.

3. Wang, P., "Factoring Multivariate Polynomials", given at the Department of Applied Mathematics, Taiwan National Chung Hsing University at Taichung, Taiwan, June 1972.

PLANNER

Prof. C. Hewitt

Instructors, Research Associates, Research Assistants and Others

C. G. Benedict
P. B. Bishop
G. L. Peskin

Preceding page blank

PLANNER

The PLANNER project is implementing a high-level goal-oriented language on Multics. The aim is to have a clean implementation of a procedural foundation for problem solving. The foundation attempts to be a matrix in which real world problem-solving knowledge can be efficiently and naturally embedded. As a first step, we are implementing the basics of the control structure and data structure.

The fundamental data structure of PLANNER is the directed graph with named links and nodes. Any data element can be used as the name of a link. Furthermore, any data element can have links associated with it. The actual linkage can be mechanized by hash coding, indirect addressing, indexing, or by a procedure call.

Processes are the active elements in the model. Instantaneously, a process is a data structure in its own right consisting of a pseudo-stack of activations of procedures. The actions of processes are caused by executing a general kind of CALL statement. There are two actions for each CALL: control leaving a procedure and control returning to the procedure. Control can leave a procedure of a process P by calling out to any of the following:

1. an ordinary function.

2. a label function L which is a function that returns to the block in which L is defined instead of to the point at which L is called.

3. a port which is a communication channel between processes.

4. some combination of the above.

Control can be resumed in the process P by one of the following:

1. an ordinary function return.

2. receiving a container of values on a port in which the process P is waiting.

3. some combination of the above such as simultaneously waiting on several ports or waiting for any of a number of ports.

Our goal is to procedurally embed problem-solving knowledge in the data and control structure so that it can be effectively used. The overhead of a mechanism should scale in proportion to the use of the mechanism. In terms of control structure we are implementing some ideas of Landau, Bobrow, and Weqbreit for a pseudo-stack for processes. The efficiency of the pseudo-stack falls off linearly in proportion to the degree that stack discipline is violated. We have implemented a hash coding scheme which scales properly with respect to large data bases.

Preceding page blank

A. Adding Knowledge

In constructing models we need the ability to embed more knowledge in the model without having to totally rewrite it. Certain kinds of additions can be easily encompassed by declarative formalisms such as the quantificational calculus by simply adding more axioms. However, declarative calculi do not allow new deductive procedures to be added. We are implementing mechanisms that allow a great deal of flexibility in adding new procedural knowledge. The data structures of PLANNER can be bound to the control structure through a variety of mechanisms. The mechanisms provide the following abilities:

> They provide the means by which knowledge can easily and naturally be embedded.
>
> They enable new knowledge to be added without rewriting everything.
>
> They make it possible to reorganize the connections between the pieces of knowledge.

PLANNER must provide interfaces so that the bindings can be controlled by knowledge of the domain of the problem. The right kind of interface promotes modularity because the procedures on the other side of the interface are not affected so long as the conventions of the interface are not changed. These interfaces aid in debugging since traps and checkpoints are conveniently placed there. More generally, formal conditions can be stated for the interfaces and verified once and for all.

B. Monitors

The mechanism of monitoring allows the attachment of an arbitrary procedure (called a monitor) to a location so that whenever the contents of the location are read, changed, or executed the monitor will be invoked. Monitors allow processes to be dynamically bound to read, write, and execute operations on particular structures. In general, whenever data is examined or modified, there must be a mechanism for a process to mediate the action.

C. Ports

Ports act as communication channels between processes. They allow the output of some processes to be fed to others without the processes having to know each others names. Thus the connections can be dynamically reconfigured without affecting the processes. The simplest kind of port consists of two queues. The first queue consists of containers of values waiting in the port to be given to processes. The second queue consists of processes waiting for containers of values. Typically either or both queues are empty.

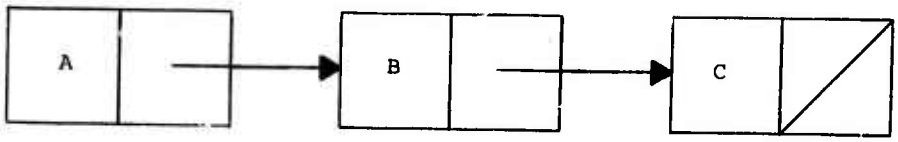
D. Data Structure Definitions

A data structure definition binds the operational properties

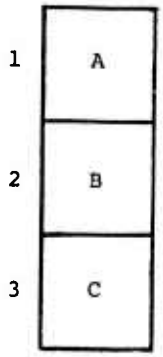
of a data type to a concrete physical representation. For example, sequences can be physically realized in a variety of ways.

Consider the following methods for realizing the sequence: A, B, C.

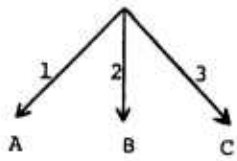
LIST



VECTOR



NODE



E. Generator

A generator is a function of no arguments which can be called repeatedly to produce the elements of a sequence. Thus the first call to a generator for the sequence would return A, the second call B, and the third C. Generators work from side-effects since they are always entered from the top with no arguments.

F. Process

A process (co-routine) has the capability of being resumed from the place where it produced its last value. Thus, unlike a generator, it does not always have to start over from the top.

G. Pattern Matching

Pattern matching is a natural way to conditionally recognize and bind the substructures of a particular data structure. Pattern matching procedures preserve visual fidelity since they "look like" the object they match. Pattern matching exhibits controlled binding by allowing a whole class of data structures to be specified.

H. Pattern Directed Retrieval

Pattern-directed retrieval enables data structures to be retrieved by a pattern of what they look like. For example, the pattern (AT NAUTILUS ?PLACE) can be used to bind the identifier PLACE to the location of the NAUTILUS if it appears in the data base.

I. Pattern Directed Invocation

Pattern-directed invocation enables processes to be invoked by triggering patterns. It is a convenient way to bind the processes that propagate the implications of updates to a model. Recommendations can be used to further control the binding of which processes will actually be invoked. Pattern-directed invocation is often used as a kind of data base monitor or "demon". Demons are more powerful (but less efficient) than ordinary monitors since they watch for changes to a whole class of data instead of just a single location.

J. Simultaneous Goals

Often we need to be able to achieve more than one goal at once. For example, we might want to have Joe, Fred, and Martha at the airport at three o'clock. We might be tempted to express the problem as follows:

```
<achieve (at Joe airport 3:00)>
<achieve (at Fred airport 3:00)>
<achieve (at Martha airport 3:00)>
```

However because of complicated travel arrangements, we might not know beforehand the order in which to try to get the people to the airport. For example, some of the people might have to chauffer others. If this is the case, then we don't want to bind the order in which a problem solver attempts the goals. We would prefer to write:

```
<simultaneous
      (at Joe airport 3:00)
      (at Fred airport 3:00)
      (at Martha airport 3:00)>
```

K. The Fire in the Warehouse

In this section we present an example to illustrate the operation of PLANNER's tree-structured data bases. PLANNER allows facts and procedures to be stored in data bases where they can be conveniently retrieved. The data bases are indexed using hash coding so that the time to retrieve an element is almost independent of the size of the data base. Suppose that warehousel has $5000 worth of plywood, $1000 of bricks and $4000 of tractors. Tractors and plywocd are flammable. We can express this more formally as:

```
<assert (in warehousel plywood $5000)>
<assert (in warehousel  bricks $1000)>
<assert (in warehousel tractors $4000)>
<assert (flammable plywood)>
<assert (flammable tractors)>
```

If the warehouse burns down, then everything inside of it which is flammable will be burned up.

```
<define burn
        ;"define burn to be the following procedure"
    <if-asserted
        (burn-down :PLACE)
        ;"if (burn-down :PLACE) is
                asserted then bind PLACE
                to the actual location and
                execute the following"
         for "current" (in .PLACE :THING <?> )
                ;"for each thing that is
                currently in the place
                execute the following"
            <if <current? (flammable .THING)>
                ;"if THING is flammable then"
                <erase (in .PLACE .THING <?> )>
                ;"erase the fact that THING is in PLACE">>>>
```

Now let us suppose that warehousel burns down.

```
<new-world ((WORLD2 <world>))
        ;"we go down inside a new-world (which we name
                WORLD2) to construct our experiment"
        ;"this new world starts out identical
                to the world from which we have just
                come but changes made in here will
                not affect the outside"
        <assert (burn-down warehousel)>
        ;"the assertion will trigger the burn 'demon' which
                will erase
                (in warehousel plywood $5000) and
                (in warehousel tractors $4000)">
```

Now we can compare and contrast the initial world with WORLD2 in which the fire took place. For example, we can ask the value of the contents of the warehouse destroyed by the fire.

```
<set LOSS 0>  ;"initialize the LOSS to 0"
<for "current" (in warehouse1 :THING :VALUE)
        ;"execute the following for each THING that was
                originally in warehouse1"
        <if
                <not  <current? (in .warehouse1 .THING
                        .VALUE) .WORLD2>>
                ;"if THING does not exist in WORLD2 then"
                <set LOSS <+ .LOSS .VALUE>>
                ;"add the VALUE OF THING into LOSS">>
```

After the above is executed, the value of LOSS will be $9000 which is the value of the plywood and tractors. Our initial world and WORLD2 are both available for inspection and modification. When we have finished our experiments, we can either save the resulting worlds or discard them having abstracted the pertinent information.

Publications

1. Hewitt, C., "Procedural Embedding of Knowledge in PLANNER", Second International Joint Conference on Artificial Intelligence, September 1-3, 1971, pp. 167-182.

2. Hewitt, C., Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot, Ph.D. Thesis, Department of Mathematics, M.I.T., 1971; also AI-TR-258, April 1972, AD 744-620.

Publications In Progress

1. Hewitt, C., "Procedural Semantics: Models of Procedures", in Some Current Views on Language, Randall Rustin, Editor, New York University Press, New York.

Talks

1. Hewitt, C., "Automatic Generation of Programs from Examples", presented at Stanford University, Palo Alto, California, June 1972.

2. Hewitt, C., "Current State of Planner", presented to the Stanford Research Institute, Palo Alto, California, June 1972.

3. Hewitt, C., "Procedural Embedding", presented at the University of Michigan, Ann Arbor, Michigan, June 1972.

4. Hewitt, C., "Procedural Semantics: Models of Procedures and the Teaching of Procedures", presented at the Courant Institute of Mathematical Sciences, New York University, New York, December 12, 1972.

SIMPL

Prof. M. M. Jones

Instructors, Research Associates, Research Assistants and Others

R. V. Harrington
S. M. Stoney
R. C. Thurber

Preceding page blank

A. Summary

The Simpl group has completed its implementation of the Simpl interactive simulation language on Multics, and has completed the documentation for the system. The completed system is being submitted to the Multics Author-Maintained Library for general use on Multics.

B. Review of the Past Year

By July, 197[illegible], an efficient version of the Simpl translator and run-time system (known as System III) had been implemented. Since then, the system has undergone five additional major evolutions, described below:

System IV — The Simpl statistical features (tables, histograms, time-plots, and queueing statistics) and tracing facilities were added.

System V — The Simpl Monitor, which enables on-line interaction with a Simpl model, was completed.

System VI — External activities and connector variables were implemented, and a central error handler was installed for the translator and run-time system.

System VII — The system was modified to handle storage allocation more efficiently, and the files in the process directory were reorganized.

System VIII — Reactivation labels (in the translated program) and internal static label variables (in the translator and run-time system) were eliminated.

We had hoped to recompile the entire system -- translator, run-time package and monitor -- with the new version II PL/1 compiler. However, even by late in May it was still not sufficiently bug-free to allow us to make this final step.

During the last few months, all documentation of the Simpl system was completed. The "Simpl Primer" and "Simpl Reference Manual" were updated to be consistent with System VIII, and the "Simpl Implementation Manual" was written, which describes the internal workings of the entire system. All three of these documents will be published as MAC TR's.

There are no future plans for the Simpl Group.

Preceding page blank

THEORY OF AUTOMATA

Prof. F. C. Hennie

Academic Staff

Prof. C. L. Liu

Prof. A. R. Meyer

Instructors, Research Associates, Research Assistants and Others

A. Bagchi
D. J. Brown
K. M. Brown
J. Ferrante
M. M. Hammer
R. F. Hossley
D. Johnson
R. Johnston
D. J. Kfoury
N. A. Lynch
R. Mandl
R. N. Moll
C. Rackoff
J. I. Seiferas
B. J. Smith
L. Stockmeyer
B. Vilfan

Undergraduate Students

M. D. Horowitz

Support Staff

M. E. Baker

S. J. Cohn

Preceding page blank

THEORY OF AUTOMATA

The Automata Theory Group is concerned with the structure and complexity of computational models and algorithms. In recent years, major interest within the group has centered on the study of algorithms and decision procedures. This study has as its goals: i) The analysis and design of optimal algorithms for basic computational tasks, such as graph manipulation, sorting, scheduling, and various arithmetic calculations; and ii) The development of methods for proving that certain computational tasks are inherently difficult to perform. In addition to this emphasis on algorithms, the group also retains an interest in some of the more traditional problems of automata and complexity theory, such as the relationships between the structure and efficiency of various computational models, and the abstract theory of the complexity of recursive functions.

The major accomplishments of the last year are outlined in four sections below. The first section, titled Inherently Complex Decision Problems, describes some new results that provide, for the first time, examples of "natural" computation problems that can be shown to be inherently difficult. The second section, Combinatorial Algorithms, describes progress made in the investigation of various packing, scheduling, and sorting problems. The last two sections present results in several more "classical" areas. That labelled Computation by Automata includes work on grammatical transformations, fault detection, machine decomposition, complexity of Boolean functions, and the theory of program schemas. The final section describes continuing work in abstract complexity theory.

A. Inherently Complex Decision Problems

Gödel's celebrated theorems reveal that no computer program can verify all the true statements of elementary arithmetic. Nevertheless, decision procedures have been developed for a number of more restricted classes of mathematical statements. Four years ago Rabin succeeded in proving that there is an algorithmic procedure for determining the truth or falsity of an arbitrary sentence of what is called the monadic second order theory of two successors (S2S). This discovery is remarkable because S2S is one of the few decidable theories in which it is possible to phrase interesting mathematical statements.

Unfortunately, Rabin's decision procedure for S2S is impossibly slow, as is Buchi's decision procedure for the weak monadic second order theory of one successor (WS1S), a precursor of Rabin's work. Attempts have been made to reduce the enormous computations involved in the known decision procedures for WS1S, but as a consequence of a new result of Prof. A. R. Meyer, we now know that any decision procedures for WS1S, S2S, and several related theories must involve impractically long computations.

Meyer's results apply equally well to certain first order logical theories. As a specific example, consider the first order theory of natural number addition together with the two-place predicate P, where P(x,y) ≡ [x is a power of 2 that divides y]. The formulas of this theory are obtained in the

Preceding page blank

usual way from atomic formulas of the form $P(x,y)$ and $x+y = z$ by means of the logical connectives and quantifiers. Thus

$$\forall x \exists y (P(y,y) \wedge \exists z (x+z = y))$$

is a (true) sentence which asserts that there are arbitrarily large powers of two.

Let $\mathcal{A}$ denote the set of all sentences of this theory. Then there exists an algorithmic procedure for determining whether an arbitrarily specified member of $\mathcal{A}$ represents a true assertion about the natural numbers. The idea behind this procedure is that for any formula $F(x)$ in the theory, it is possible to construct a finite automaton that accepts precisely the binary representations of those natural numbers n for which $F(n)$ is true. Carrying out this construction for a given sentence in $\mathcal{A}$ therefore yields an automaton that accepts every string of 0's and 1's if the sentence is true and accepts no string if the sentence is false. Since it is relatively easy to determine whether an automaton accepts all input strings, one can effectively decide whether a given sentence in $\mathcal{A}$ is true.

The difficulty with this approach is that sentences of length n yield automata having on the order of $\theta(n)$ states, where

$$\theta(n) = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \quad \text{(a tower of } n \text{ twos)}$$

The number of steps needed to determine whether such an automaton accepts every input string is therefore of the same order. Thus although it is in theory possible to determine the true sentences of $\mathcal{A}$, the procedure outlined above is totally impractical. Meyer's result shows that in fact every procedure which determines the true sentences of $\mathcal{A}$ must use the same exorbitant number of steps. Specifically:

> There exists a number $\epsilon > 0$ such that for any Turing machine that decides the truth of sentences in $\mathcal{A}$, there is an integer n_0 such that for all $n \geq n_0$ there is a sentence of length n in $\mathcal{A}$ on which the given machine requires at least $\theta(\epsilon n)$ steps.

Although this result is phrased in terms of Turing machine computations, essentially the same result holds for more realistic computational models, since even the most powerful model can be simulated by a Turing machine in an amount of time that is very small compared to $\theta(n)$.

This does not necessarily mean that no progress can be made in the development of decision procedures for the theories mentioned above. It might be that there are fast ad hoc

procedures for all sentences of length less than (say) 10^6 letters, or that most sentences of a given length are easy to decide, or that all "interesting" sentences are easy to decide. Nevertheless, our results strongly suggest that there is little hope for practical application of many decision procedures in logic.

Another interesting consequence of these results is that we now have examples of procedures that exhibit the Blum speed-up property. Earlier work in abstract complexity theory, some of which was carried out in this group, established the existence of computable functions with speed-up, but it did not appear that any natural mathematical problem had this property. We now know that given any decision procedure for A, one can effectively construct another procedure requiring less than the logarithm of the number of steps of the given procedure for infinitely many sufficiently long sentences. Thus there is no optimal decision procedure for A, and no exact computational complexity that can be assigned to the problem of deciding A.

Meyer's results are derived in part from the work of several other members of the group. One of the key ideas in Meyer's proof comes from Larry Stockmeyer's study of the complexity of a problem in automata theory. Stockmeyer has shown that the problem of deciding whether two nondeterministic finite automata are equivalent requires an amount of time that grows faster than any polynomial function of the size of the automata. Robert Hossley and Charles Rackoff have significantly improved a portion of the proof that S2S is decidable. Hossley, Rackoff, and Jeanne Ferrante also obtained corollaries establishing the inherent computational complexity of various theories related to WS1S. Efforts are now being made to extend our techniques to the first order theory of addition, and to the subcase of existential statements about addition. Success in the latter subcase would yield lower bounds on the complexity of integer programming and related combinatorial problems of widespread interest.

B. Combinatorial Algorithms

The area of combinatorial algorithms, which at first sight seems to consist of many unrelated problems, is beginning to show some coherence. Within the last year, Cook and Karp have proved that the following problems are computationally equivalent:

i) Solving a zero-one valued travelling salesman problem
ii) Finding a maximal size clique in an undirected graph
iii) Finding a minimal covering of a family of sets
iv) Determining whether a Boolean function is identically equal to 1.

These problems are "equivalent" in the sense that a fast

algorithm for solving any one of them can be used as the basis for constructing a fast algorithm for solving any one of the others. (Here a "fast algorithm" is one whose computation time is bounded by at most a polynomial function of the size of the inputs.) At present, all known algorithms for solving these problems require an amount of time that grows exponentially with the size of the input. Thus these four problems, as well as many other familiar combinatorial problems, are hard for the same reason--assuming that they really are hard, which remains to be shown. In particular, a proof that any one of them is inherently time consuming to solve would automatically establish that they all are.

Several members of the group have been investigating extensions of the Cook and Karp classification. As already noted, Larry Stockmeyer's efforts to classify the equivalence problem for nondeterministic finite automata led to Meyer's results on decision problems. And David Johnson, working with Professors Meyer and Fischer, has shown the following problem to be equivalent to the Cook-Karp problems: Given a sequence of rational numbers $r_1,\ldots,r_n$, where $0 < r_i \leq 1$, find the smallest number of unit-sized "bins" into which the given rational numbers can be "packed" subject to the constraints that each r_i is assigned to exactly one bin and that the sum of the numbers in any bin is at most one.

This packing problem differs from the problems of Cook and Karp in one important respect. Although finding an optimal packing appears to require an amount of time that grows exponentially in n, there are very efficient packing algorithms that give nearly optimal results. Extending earlier work of Garey, Graham and Ullman, David Johnson has discovered a packing algorithm that is guaranteed to operate in time proportional to n and that never uses more than 11/9 the optimal number of bins. Moreover, it has been shown that this bound of 11/9 is the best possible one for the class of fast algorithms under consideration. Johnson is now trying to extend his techniques to various covering problems, where it also appears that efficient methods yielding near optimal results can be found.

In a related area, Prof. C. L. Liu has been investigating the efficiency of various algorithms for scheduling jobs in a multi-processor computing system. Typical of the problems studied is the following. A set of tasks $\{T_1,T_2,\ldots,T_r\}$ are to be scheduled on a two-processor system, where the execution time of each task is known. The order in which the tasks may be performed is constrained by a specified partial order $\prec$ so that if $T_i \prec T_j$ the execution of T_j must not begin until the execution of T_i has been completed. How is the total elapsed time for the execution of all the tasks affected by the nature of the algorithm used to schedule the tasks?

One of the major results obtained is a quantification of the relative efficiencies of preemptive and non-preemptive scheduling algorithms. (A preemptive schedule is one in which the execution of one task may be interrupted to begin another task.) Let ω denote the total elapsed time required for the execution of a given set of tasks when an optimal non-preemptive schedule is used, and let ω' denote the corresponding time when an

optimal preemptive schedule is used. Clearly,

$$\omega \geq \omega'$$

However, Liu has shown that

$$\omega' \geq \frac{3}{4}\,\omega$$

for any set of tasks and any specified partial order $\prec$. This result can be extended to the case of n processors, where

$$\omega' \geq \frac{n+1}{2n}\,\omega$$

Moreoever, these bounds have been shown to be the tightest possible ones.

Liu's results may be interpreted as follows. The introduction of a high-speed drum so as to make possible the use of preemptive schedules can increase the speed of a two-processor computing system by at most 25%, and that of an n-processor system by at most 50%.

Burton Smith and Prof. C. L. Liu have been investigating the behavior of sorting networks made up of two-input, two-output comparators. Although general methods for designing n-input, n-output sorting networks are known, the problem of designing networks that use as few comparators as possible is very difficult. One of the best networks of modest size known to date, due to M. W. Green, uses 60 comparators to sort 16 inputs. Liu has recently found a 61-comparator, 16-input network, as well as a new four-way merge technique that yields a 4^n input sorting network using $(n^2 - \frac{7}{12^n} + \frac{10}{9})4^n - \frac{10}{9}$ comparators for any n.

The difficulty of designing efficient sorting networks is due in part to the difficulty of determining whether or not a given network of comparators actually sorts its inputs. One criterion for making this determination is Knuth's zero-one principle: A comparator network sorts if it correctly sorts all possible zero-one valued input patterns. Smith has shown that this criterion is sharp, in that for any given zero-one input pattern (other than the all-zero or all-one patterns) there exists a network that correctly sorts every zero-one input pattern except the one in question.

Another criterion for deciding whether a network sorts is based on the fact that many sorting networks are formed by combining several sorting networks with fewer inputs. Indeed, every sorting network can be viewed as a cascade combination of two-input sorting networks. Liu has developed a procedure for analyzing certain composite networks of this type which is useful both in designing sorting networks and in proving that a given network operates correctly. Smith has extended Liu's work to apply to arbitrary networks of comparators, and has developed several techniques for succinctly characterizing the patterns that can appear at the output of a given comparator network. Such characterizations are useful in designing networks as cascades of simpler ones.

In some cases, the set of possible outputs of a comparator network consists of precisely those patterns of values that are consistent with a particular partial order on the output terminals. Smith has shown that if this is true when the network inputs are restricted to patterns of distinct values, it must also be true when the inputs are zero-one valued. In the case of arbitrary real-valued inputs, the set of possible outputs will be those consistent with a single partial order if and only if the outputs form a convex set in Euclidean n-space. This case is interesting because it results in a very compact description of the output behavior of a network. However, if an n-input sorting network is designed so this condition obtains at every point within the network, the network will necessarily require $(\frac{n}{2})$ comparators and hence will be very inefficient.

C. Computation by Automata

Automata and formal linguistic models have many applications in the study of computers and computation, ranging from logical design to compiler construction to complexity theory. During the past year, several members of the group have explored computational issues arising from such models. Their results are outlined below.

1. Grammatical Transformations

One of the most fruitful developments in automata theory in recent years has been the application of automata theoretic ideas to problems in the theory of languages and compilers. Substantial progress in this area has recently been made by Michael Hammer, who has been investigating grammatical transformations that are capable of converting given programming language grammars into equivalent but more useful grammars. In particular, he has found a class of transformations yielding grammars whose languages can be parsed top-down.

While top-down parsing has a number of advantages over bottom-up parsing, the grammars that admit top-down parses (the so-called LL grammars) constitute a small subset of those that admit bottom-up parses (the LR grammmars). Hammer has been working with a subclass of the LR grammars known as minimally predictive (MP) grammars. These grammars can be parsed in a hybrid fashion that is basically a bottom-up parse with occasional judicious predictions. A procedure has been found for transforming any MP(k) grammar into an LL(k) grammar that generates the same language. Moreover, this transformation preserves the semantic capabilities of the original grammar, in the sense that any translations supported by the original grammar are also supported by the new one.

The transformation itself is based on the structure of the finite-state machine that directs the progress of a bottom-up parse for the given LR grammar. The first step is to split the states of this machine in such a way as to break certain kinds of loops in the machine's state diagram. Once this has been done, the new grammar can be "read off" directly from the altered machine. Transformed grammars obtained in this way can be shown to have a number of interesting properties:

they are frequently LR(0), they support a wide range of translations, and they can be parsed quickly. The same kind of transformation can also be applied to certain non-LR grammars so as to yield equivalent deterministic grammars.

Results obtained so far have provided new insight into the nature of LL and LR grammars and the relationship between top-down and bottom-up parsing. Moreover, this work promises to have applications to the derivation of optimal grammars for use in practical compilers.

2. Fault Detection

It is of considerable practical importance to be able to determine experimentally whether a given piece of hardware is operating correctly. Although a variety of fault detection and diagnosis techniques are available for use in practical situations, the essential complexity of the detection and diagnosis problems is still poorly understood. In particular, realistic upper and lower bounds on the lengths of general fault-detection experiments for sequential circuits have yet to be established. In the case of lower bounds, for example, it is easy to see that a valid fault detection experiment must cause the correctly operating circuit to traverse each of its state transitions at least once, an observation that leads to a lower bound of mn for the length of a fault detection experiment for an m-input, n-state circuit. However, no circuit admits a detection experiment of this short a length.

Some progress in this area has been made by Ken Brown in work done for his SB thesis. He has shown that every fault detection experiment for a two-input circuit must cause the circuit to traverse each transition at least once and half of the transitions at least twice. This result provides a lower bound of 3n for the length of detection experiments for n-state, two-input machines. There exist circuits for which this bound is attainable, and there exists a class of circuits for which the argument leading to the lower bound provides a simple set of necessary and sufficient conditions for an input sequence to be a fault-checking sequence. Thus we now have examples of circuits that are easy to test and an accurate assessment of the complexity of the experiments needed to test them. We expect that this work can be extended so as to yield more accurate bounds for more general classes of circuits.

3. Linear Machines

In the course of investigating some fault-detection questions for linear sequential machines, Robert Johnston discovered a new approach to the decomposition of a broad class of linear systems.

The usual notion of a linear machine as one whose input and state spaces are vector spaces over some field was extended to that of a machine whose input and state spaces are modules over an arbitrary ring. The development of this more general class of machines parallels that described in Kalman, Falb, and Arbib. Johnston next derived certain conditions that are

sufficient to ensure that the state-space module is finitely generated. It turned out that these conditions are also sufficient to ensure that the homomorphisms used to define the machine's next-state and output functions can be described by matrices of a convenient (companion) form.

The basic decomposition result was obtained by requiring the input module and ring involved in the definition of a machine to satisfy an ascending chain condition. Specifically, any sequence of submodules (or left ideals) in which each submodule (ideal) is properly contained in the last is required to be of finite length. Johnston has shown that any machine satisfying this condition can be simulated by a cascade combination of simpler machines, most of which are reset machines (machines whose next state does not depend on the current state).

Work in this area is now directed towards (i) investigating the relationship between the decomposition described here and the Zeiger decomposition for finite-state machines; (ii) exploring other forms of decomposition; and (iii) studying the control theoretic applications of the approach described above. In view of the relevance to control theory of many of the questions raised by this work, we expect that further investigations will be carried out jointly with the Electronic Systems Laboratory.

One of the classic problems associated with linear sequential machines is that of determining whether a specified terminal behavior is realizable by a linear machine. In particular, one can ask whether a given regular expression on the alphabet $\{0,1\}$ is recognized by a linear machine over GF(2). The usual technique for answering this question involves constructing the finite-state machine associated with the given expression and then determining whether in fact that machine is linear. In his bachelor's thesis, Mark Horowitz sets forth a method that more fully exploits the property of linearity. The first step is to determine the necessary impulse response from the given regular expression and to construct the linear machine having that impulse response. It then remains only to decide whether the machine actually recognizes the given regular set. Horowitz plans to try to extend his technique to deal with multiple inputs, multiple outputs, and arbitrary finite fields.

4. Schemas for Programs

A schema is essentially a program in which some or all of the relations and operations are left undefined. One of the motivations for the study of schemas is to determine how much of the computing power of a program is inherent in its flow structure, rather than in the semantics of its instructions. From a practical point of view, this corresponds to determining how much of the compilation process can be achieved by considering only the flow structure of the given program. In a recent doctoral thesis, Denis Kfoury has applied some of the techniques of first-order logic and model theory to investigate program schemas. His work was motivated by the following kind of question: When restrictions are placed on the possible interpretations of the relations and operations of programs,

what must be required of the flow-structure of programs in order to achieve a certain level of computational power?

Different classes of schemas have different expressive powers. The first objective of Kfoury's thesis is to find a "universal" class S of schemas which, when interpreted in any algebraic structure, yield the set of all effective procedures over the domain of that structure -- effective relative to the underlying relations and operations of the structure. Given an arbitrary class S' of schemas, Kfoury investigates conditions on an algebraic structure $\mathcal{A}$ that will ensure the equivalence of S' to the universal class S, in which case S' will capture the notion of effectiveness in $\mathcal{A}$. In particular, he considers the class of flow-chart schemas, and shows that they are sufficient to capture the notion of effectiveness in many of the algebraic structures encountered in practice, including rings, fields, and ordered fields.

A second objective of the thesis is to compare algebraic structures in terms of their "algorithmic behavior" -- i.e., in terms of the properties of schemas interpreted in those structures. Kfoury gives special attention to conditions under which this algorithmic behavior is the same for related structures, such as two groups or two fields. As a by-product of this investigation, he has shown that over some familiar structures, such as the complex number field, an effective procedure is total only if it is equivalent to a loop-free procedure.

5. Complexity of Boolean Functions

In order to explore the role of information-theoretic arguments concerning the complexity of functions, Bostjan Vilfan has been studying the sizes of expressions needed to represent certain Boolean functions. In particular, he has considered expressions built up from variables and symbols for the Boolean operations of and, or, not and exclusive or, for such functions as the n-variable function whose value is 1 if and only if the number of arguments equal to 1 is divisible by 3. Using some rather deep combinatorial arguments, Vilfan has shown that expressions for this "divisible by three" function of n variables must grow nonlinearly with n no matter what finite set of basic operations are allowed in expressions. This lower bound is close to being the best possible, since for any $\epsilon > 0$ there is a finite set of operations in terms of which the "divisible by three" functions have expressions of length at most $n^{1+\epsilon}$.

Abstract Complexity Theory

As noted in our last progress report, the development of abstract complexity theory no longer constitutes a dominant part of the group's activity. Nevertheless, the results on the inherent complexity of decision problems cited above illustrate the relationship between theoretical work in complexity and computational problems of more practical interest. We expect, therefore, to maintain a small research effort in the area of abstract complexity theory.

During the last year, Amitava Bagchi, Nancy Lynch, and Robert Moll have all made valuable contributions to complexity theory. In these contributions, some of the highly developed methods of recursive function theory have been brought to bear on questions of complexity. In particular, several of Lynch's and Moll's results make use of the priority-injury arguments of recursion theory.

The main theorem of Lynch's thesis illustrates the nature of these results. This theorem is motivated by the Cook-Karp result noted earlier in which a class of familiar problems are shown to be computationally equivalent. Computationally equivalent problems have solutions that are equally hard to calculate. Is the converse true? That is, if two decision problems are known to require the same (large) amount of time to solve, does it follow that the ability to solve one problem in no time at all (as by means of an instantaneous oracle) would provide a way of solving the other problem quickly? Lynch has shown that for any decision problem with a known lower bound on the time required for solution, there are arbitrarily complex decision problems that do not help in the solution of the given problem. This result gives mathematical meaning to the intuitive assertion that while the Cook-Karp problems are hard for the same reason, other equally hard problems are hard for different reasons.

References

1. Büchi, R., Weak Second-Order Arithmetic and Finite Automata, Zeit. f. Math. Logik und Grund. d. Math., 6, 1960, 66-92.

2. Cook, S.A., The Complexity of Theorem-Proving Procedures, 3rd ACM Symp. on Theory of Computing, 1971, 151-158.

3. Garey, M.R., R.L. Graham and J.D. Ullman, Worst-Case Analysis of Memory Algorithms, 4th ACM Symp. on Theory of Computing, 1972.

4. Kalman, R.E., Falb, P.L., and M.A. Arbib, Topics in Mathematical Systems Theory, McGraw-Hill, 1969.

5. Karp, R.M., Reducibility Among Combinatorial Problems, Tech. Report 3, Department of Computer Science, University of California, Berkeley, 1972.

6. Knuth, D.E., The Art of Computer Programming, Vol. 3, Addison-Wesley, (To be published 1972)

7. Rabin, M.O., Decidability of Second-Order Theories and Automata on Infinite Trees, Trans. Amer. Math. Soc., 141, July 1969, 1-35.

Publications

1. Bagchi, A., Economy of Descriptions and Minimal Indices, Ph.D. Thesis, Department of Electrical Engineering, M.I.T., 1972; also MAC-TM-27, January 1972, AD 736-960.

2. Brown, K., Lower Bounds for Finite State Machine Checking Experiments, S.B. Thesis, Department of Electrical Engineering, M.I.T., 1972.

3. Fischer, M. J. and A. R. Meyer, "Boolean Matrix Multiplication and Transitive Closure", Conference Record 1971 12th Annual Symposium on Switching and Automata Theory, pp. 129-131.

4. Horowitz, M. D., Linear Finite Automata and Their Regular Expressions, S.B. Thesis, Department of Electrical Engineering, M.I.T., 1972.

5. Hossley, R., Finite Tree Automata and ω-Automata, S.M. Thesis, Department of Electrical Engineering, M.I.T., 1972; also MAC-TR-102, September 1972, AD 749-367.

6. Kfoury, D., Effective Procedures in Arbitrary Structures, Ph.D. Thesis, Department of Electrical Engineering, M.I.T., 1972.

7. Liu, C. L., "Construction of Sorting Plan", Theory of Machines and Computations, Academic Press, New York, 1971, pp. 87-98.

8. Liu, C. L., "Analysis of Sorting Algorithms", Conference Record 1971 12th Annual Symposium on Switching and Automata Theory, pp. 207-215.

9. Lynch, N. A., Relativization of the Theory of Computational Complexity, Ph.D. Thesis, Department of Mathematics, M.I.T., 1972; also MAC-TR-99, June 1972, AD 744-032.

10. Meyer, A. R. and A. Bagchi, "Program Size and Economy of Description", Conference Record of 4th ACM Symposium on Theory of Computing, 1972, pp. 183-187.

11. Meyer, A. R. and E. M. McCreight, "Computationally Complex and Pseudo-Random Zero-One Valued Functions", Theory of Machines and Computations, Academic Press, New York, 1971, pp. 19-42.

12. Paterson, M. and L. Stockmeyer, "Bounds on Evaluation Time for Rational Polynomials", Conference Record 1971 12th Annual Symposium on Switching and Automata Theory, pp. 132-139.

Publications (continued)

13. Stockmeyer, L., Bounds on Polynomial Evaluation Algorithms, S.M. Thesis, Department of Electrical Engineering, M.I.T., 1972; also MAC-TR-98, April 1972, AD 740-328.

14. Vilfan, B., The Complexity of Finite Functions, Ph.D. Thesis, Department of Electrical Engineering, M.I.T., 1972; also MAC-TR-87, June 1971, AD 726-049.

PROJECT MAC PUBLICATIONS

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 Corbató, 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-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

Preceding page blank

PUBLICATIONS

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

PUBLICATIONS

* 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 677-602

* TR-50 Denning, Peter J.
Resource Allocation in Multiprocess Computer Systems, Ph.D. Thesis, EE Dept.
May 1968 AD 675-5[illegible]

* 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

PUBLICATIONS

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, Ph.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, Sc.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

PUBLICATIONS

* 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, Elec. 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 1971 — 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

PUBLICATIONS

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

TECHNICAL MEMORANDA

* 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-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

PUBLICATIONS

* 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's 1-9 were never issued

PUBLICATIONS

*	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	

****

Copies of all MAC reports listed in Publications may be secured from the National Technical Information Service, Operations Division, Springfield, Virginia, 22151. The prices from NTIS are: microfilm $0.95; hard copies: reports more than two years old $6.00, all others are $3.00 except TR-83 which is also $6.00. The AD number must be supplied with the request.

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