

AD-A061 246

MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
LABORATORY FOR COMPUTER SCIENCE (FORMERLY PROJECT MAC) PROGRESS--ETC(U)
AUG 78 M L DERTOUZOS

N00014-75-C-0661

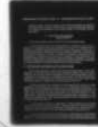
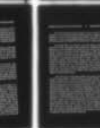
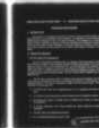
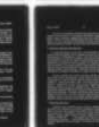
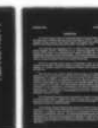
NL

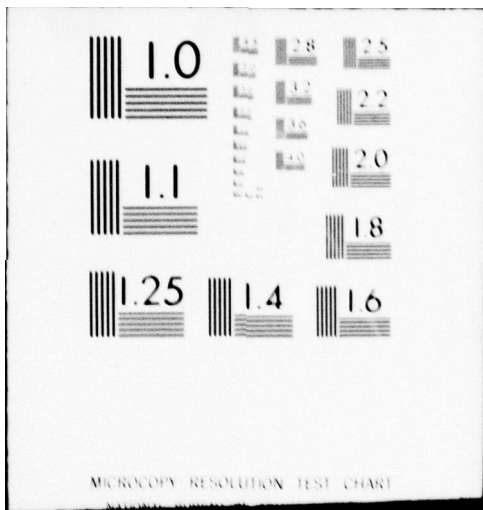
UNCLASSIFIED

LCS-PR-13

1 of 2

AD
A061 246





DDC FILE COPY
ADA061246

LABORATORY FOR
COMPUTER SCIENCE

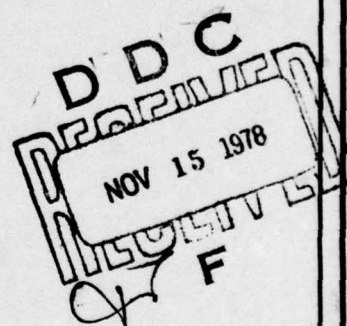


MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

12 NW
LEVEL II

Progress Report XIII

JANUARY - DECEMBER 1975



This document has been approved
for public release and sale; its
distribution is unlimited.

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

78 11 06 042

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Progress Report XIII (13)	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Laboratory for Computer Science (formerly Project MAC) Progress Report XIII January-December 1975,	5. TYPE OF REPORT & PERIOD COVERED Arpa-DOD Progress Report 1/75-12/75	6. PERFORMING ORG. REPORT NUMBER LCS-PR- 111 13
7. AUTHOR(s) Laboratory for Computer Science Participants M. L. Dertouzos Director	8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-0661 Work Order No. 2095	
9. PERFORMING ORGANIZATION NAME AND ADDRESS LABORATORY FOR COMPUTER SCIENCE (formerly project MAC) Massachusetts Institute of Technology 545 Technology Square, Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency Department of Defense 1400 Wilson Blvd. Arlington, Va., 22209	12. REPORT DATE Aug 1978	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Department of the Navy Information Systems Program Arlington, VA 22217	13. NUMBER OF PAGES 106	
	15. SECURITY CLASS. (of this report) unclassified	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited ⑨ Annual summary rept.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Real-time Computers	Computer Languages	Morse-Code
On-line Computers	Computer Networks	Knowledge-Based
Multi-Access Computers	Information Systems	Systems
Dynamic Modelling	Programming Languages	Complexity
Computer Systems	Computation Structures	
	Automata Theory	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Annual Summary Report of progress made at the Laboratory for Computer Science under this contract during the period January-December 1975.		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

unclassified 409648
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

LB

Work reported herein was carried out within the Laboratory for Computer Science (formerly Project MAC), a Massachusetts Institute of Technology Interdepartmental laboratory. Support was provided by the Advanced Research Projects Agency of the Department of Defense, under Office of Naval Research Contract N00014-75-C-0661.

Reproduction of this report, in whole or in part, is permitted for any purpose of the United States Government. Distribution of this report is unlimited.

LABORATORY FOR COMPUTER SCIENCE
(formerly Project MAC)
PROGRESS REPORT XIII

JANUARY - DECEMBER 1975

LABORATORY FOR COMPUTER SCIENCE
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
CAMBRIDGE, MASSACHUSETTS 02139

ACCESSION for	
NTIS	Write Section <input checked="" type="checkbox"/>
DDC	S. H. Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION	<input type="checkbox"/>
BY	
DISTRIBUTION/AVAILABILITY CODES	
Disc.	or SPECIAL
A	

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

INTRODUCTION	1
LABORATORY FOR COMPUTER SCIENCE ADMINISTRATION	3
<u>COMPUTER SYSTEMS RESEARCH GROUP</u>	5
A. Introduction	7
B. The Information Sharing Kernel Design Project	7
C. Other Activities	13
<u>DATA BASE SYSTEMS GROUP</u>	19
A. Introduction	21
B. Self-Organizing Data Bases	21
C. Error Detection	24
D. Distributed Data Bases	26
<u>DOMAIN SPECIFIC SYSTEMS RESEARCH GROUP</u>	29
A. Introduction	31
B. Domain Specific Systems	31
<u>KNOWLEDGE-BASED SYSTEMS GROUP</u>	41
A. Introduction	43
B. Roster of Projects	43
Appendix	47
A. Protosystem 1: An Automatic Programming System Prototype	47
B. A Model of the Program Writing Process	47
C. Efficiency Enhancement in System Development	49
D. The Development of Protosystem 1	53
E. The Protosystem 1 Data Processing System Model and the System Specification Language	53
F. The Translator and the Data Set Language	56
G. The Design Criterion and the Job Cost Estimator	57
H. The Question Answerer	57
I. The Optimizing Designer	59
J. Code Generation	63
K. Conclusion	63

TABLE OF CONTENTS

<u>MATHLAB GROUP</u>	67
A. Introduction	69
B. The MACSYMA Consortium Machine	69
C. Matrix Inversion Algorithms	69
D. Polynomial Decomposition	71
E. Definite Integration of Special Functions	71
<u>PROGRAMMING METHODOLOGY GROUP</u>	77
A. Introduction	79
B. The CLU Language/System	79
C. Structured Handling	81
D. Specifications for Data Abstractions	85
<u>PROGRAMMING TECHNOLOGY GROUP</u>	95
A. Introduction	97
B. Expert Knowledge Applied to the Morse-Code Domain	97
C. Message System	105
D. DMS Activities	108
E. English Parser	110
<u>LABORATORY FOR COMPUTER SCIENCE PUBLICATIONS</u>	115

INTRODUCTION

This annual progress report to the Advanced Research Projects Agency (ARPA) of the Department of Defense describes research performed at the M.I.T. Laboratory for Computer Science (formerly Project MAC), funded by that agency and monitored by the Office of Naval Research during the period January 1-December 31, 1975.


The Laboratory was organized at M.I.T. in 1963 to conduct research in Time-Shared Computer Systems. Contributions of L.C.S. include the Compatible Time-Sharing System (CTSS), Multics, the mathematical-expert program MACSYMA, and a variety of programming languages, systems and techniques. The research described in this report reflects the current research directions of the Laboratory, oriented to promising areas as well as pressing technological needs of the computer science field.

During the reporting period (January 1975-December 1975), L.C.S. personnel entailed approximately 295 people, including 39 faculty, 68 research and support staff members, 111 graduate students, 69 undergraduate students, and 8 visiting researchers and scientists.

△ The main focus of the research reported herein has been in the reduction of the substantive and increasing costs associated with the generation, maintenance and documentation of programs. In particular, work carried out by the Knowledge-Based Systems group focused on the identification of a very high level language in which inventory control programs are specified, and on the associated compiler that translates such a program to PL/1 code. In the Domain Specific Systems Research group research commenced on the programming of microcomputers from high-level languages for such purposes as the automatic control of physical processes, maintenance and instrumentation. Developments in the Matlab group have led to formation of the MACSYMA consortium of users.

△ The Programming Technology group concentrated its research on the development of a Morse Code system. Through this system the group seeks to understand and develop techniques for embedding a great deal of structural knowledge (in this case about Morse Code) into computer programs.

△ The Computer Systems Research group focused its research on the analysis and certification of large systems using the MULTICS systems as its principal model and laboratory. In addition, work was inflated on a local-network that will link the laboratory's computational resources. The Programming Methodology group continued the development of the structured programming language CLU which has a modular construction that facilitates the representation of abstractions. Finally, the newly formed Data Base group has begun work on improving the efficiency and accuracy of very large data bases.



Acknowledgements

Assembly and compilation of this report was done by Paulyn Heinmiller. Illustrations were done by Allison Platt.

ADMINISTRATION

Academic Staff

M. L. Dertouzos
J. Moses

Director
Associate Director

Administrative Staff

M. E. Baker
L. G. Daniels
H. S. Hughes
C. P. Kent
T. L. Lightburn
G. W. Oro
A. A. Platt
D. C. Scanlon
G. L. Wallace

Administrative Assistant
Librarian
Administrative Services
Assistant Fiscal Officer
Fiscal Consultant
Fiscal Officer
Information Services
Administrative Officer
Purchasing Agent

Support Staff

G. W. Brown
L. S. Cavallaro
P. Heinmiller
J. Jones
D. Kontrimus

M. K. Martucci
E. M. Roderick
T. Sealy
R. Varjebedian

C. S. R. GROUP

5

C. S. R. GROUP

COMPUTER SYSTEMS RESEARCH

Academic Staff

J. H. Saltzer, Group Leader
D. D. Clark
F. J. Corbato
C. A. Ellis

D. D. Redell
M. D. Schroeder
L. Svobodova

Research Staff

N. C. Federman
R. K. Kanodia
R. F. Mabee

K. T. Pogram
D. M. Wells

Graduate Students

A. J. Benjamin
T. Bloom
E. C. Ciccarelli
R. J. Feiertag
H. C. Forsdick
R. M. Frankston
H. J. Goldberg
A. R. Huber
D. H. Hunt

P. A. Janson
S. T. Kent
A. W. Luniewski
A. H. Mason
W. A. Montgomery
D. P. Reed
M. Shibuya
V. L. Voydock

Undergraduate Students

C. R. Davis
R. S. Gale
D. L. Gifford
B. M. Grant

T. B. Lake
R. P. Planalp
G. J. Rudisin
S. A. Swernofsky

Support Staff

P. G. Heinmiller
J. P. Knowlton
V. M. Newcomb

C. Sarner
M. F. Webber

C. S. R. GROUP

6

C. S. R. GROUP

Guests

N. A. Adleman
S. E. Estes

W. Maczko

COMPUTER SYSTEMS RESEARCH

A. INTRODUCTION

The Computer Systems Research Division of the M.I.T. Laboratory for Computer Science completed several key parts of its information sharing kernel design project during 1975. Several other network-related activities were also accomplished. These activities are described separately, in the two following sections.

B. THE INFORMATION SHARING KERNEL DESIGN PROJECT

Three years ago, we entered into a project to perform engineering studies on strategies for simplifying the design of the resource-sharing and information-sharing kernel of a full-scale computer system, with the goal of making the security aspects of a system simple enough that certification of correctness might be possible. Multics is the laboratory in which these experiments have been performed. This year, significant progress occurred on several key aspects of this work:

1. Development of the use of type-extension as a strategy for systematic design of the kernel itself.
2. Organization of processor multiplexing in two layers, with memory multiplexing sandwiched between, to untangle these two complex mechanisms.
3. Organization of memory multiplexing in identified parallel processes rather than in a central structure.
4. Organization of process initiation as an unprivileged operation controlled by domain entry mechanisms.
5. Development of a new model of process synchronization, called the "eventcount" model, that leads to simpler coordination algorithms and minimizes unnecessary communication, a feature important to security.

The cumulative impact of these projects on the structure of a system kernel, together with a variety of other ideas currently being explored, appears to be significant in that the kernel becomes modular, ordered, and thereby incrementally verifiable.

This year, the project's activities are of a different nature than those reported in previous years. Earlier reports concentrated on reducing the size of the security kernel by removing unnecessary functions, while this year's work has concentrated on better understanding of how the remaining, essential functions might be more systematically organized. Two key ideas have led us to this understanding. First, the use of abstract

types as a methodology for choosing and specifying the interfaces inside the kernel (as pioneered in HYDRA, CLU, and SIMULA) gives a useful and clear decomposition of the kernel. Second, the use of processes within the kernel to multiplex the resources used in implementing objects of abstract type gives a much simpler control structure inside the kernel.

Our basic approach to simplifying the structure of the kernel is to decompose its design and implementation into modules. By structuring the decomposition into modules correctly, we hope to obtain a system in which understanding or verifying the system as a whole requires little more effort than understanding or verifying every module separately. The problem with obtaining such a well-structured decomposition of the system is to find a way to decompose the system into modules that are internally simple and have simple interactions with the other modules of the system.

Simplifying the interactions among modules is aided by two techniques. First, the method by which interacting modules communicate can be simplified. Philippe Janson, in his Ph.D. thesis, has categorized modularizations into two classes: strict modularization, in which modules interact with another module only by invoking procedures in the other module, and weak modularization, in which modules may communicate via shared data bases. By designing a system in terms of strict modules, it is much simpler to define the effect of a particular intermodule interaction. The second technique for simplifying interaction is to define a partial ordering of modules based on functional dependency. Module A depends on module B if B must correctly meet its functional specification in order for A to meet its functional specification. If all dependencies are uni-directional, and form a partial ordering, then it can be quite simple to verify the correct operation of all modules. One starts with modules that are assumed to be correct (for example, the hardware) and proceeds to verify all modules by induction on the partially-ordered structure.

1. Abstract Types as a Structuring Tool

A structuring methodology that leads to both a strict modularization and a modularization that is partially ordered in functional dependency is the type-extension mechanism for creating abstract types. An abstract type is a collection of abstract objects and operations on the abstract objects. The specification of the properties of and interface to the objects of the type is independent of the actual storage representation of the objects or implementation of the operations in terms of the storage representation. The only way to manipulate objects of the type is to call on the operations of the type. Thus a modularization based on abstract types is strict. Types may be implemented in terms of objects of other types. This results in a uni-directional functional dependency.

Both Janson and David Reed have investigated the use of abstract types in the design of the kernel of an operating system such as Multics. In an operating system, the implementation of abstract types and the process of type extension cause difficulties not present in abstract type concepts as implemented in programming languages such as CLU. The major difficulty arises from scarcity of memory and processing resources to implement objects and operations, requiring multiplexing of those resources. Using the abstract type concept to structure the multiplexing functions has led to some new insights into the structure of operating systems and the mechanism of type extension. In contrast, the HYDRA system, which supports abstract types outside the kernel, does not use abstract types in the multiplexing of memory and processors to provide virtual memory or virtual processors.

Janson has defined a new model of abstract types to be used in the design of the kernel of the system where multiplexing of objects is the key problem. The primary difference between this model and older ones is that he explicitly recognizes the limitations on the supply of low-level resources, such as primary memory and processor resources. He also recognizes the multiplexing function, by explicitly including in his model a time-varying mapping between objects of abstract type and the objects used in their representation.

2. Disentangling Processor and Memory Multiplexing

An important result of our work on structuring the kernel is actually disentangling the interdependency between processor and memory multiplexing algorithms. This interdependency results from the need to provide a large amount of memory for tables used in implementing virtual processors for user computations and the simultaneous need to provide and control the processing power used to interpret the virtual memory algorithms.

The technique for breaking this interdependency is to divide processor multiplexing into two levels. The first level of processor multiplexing provides a small set of virtual processors, called level 1 processors, that have sufficient functionality to implement the virtual memory algorithms. These virtual processors access primary memory in exactly the same way that physical processors do, through address translation hardware. Any attempt to access an object not in primary memory is reflected as a fault, just as in the real processor. The virtual memory software is implemented in terms of these level 1 processors. Andrew Huber has proposed a design for virtual memory implementation that uses multiple dedicated virtual processors to perform its functions. The second level of processor multiplexing takes a subset of the level 1 virtual processors and multiplexes them to provide a large set of level 2 virtual processors, used to run user processes. The data bases of the level 2 processor multiplexing algorithms are implemented in terms of virtual memory objects. The processor resources for the level 2 manager algorithms are provided by three dedicated level 1 processors.

3. Using Processes as a Structuring Tool

As a result of this two level design, level 1 virtual processors can be dedicated to handle management of many multiplexed operating system resources. Level 1 processors are relatively cheap compared to real physical processors, so dedicating them gives some of the effect of dedicating a physical processor, without the cost.

Structuring the kernel as a set of processes running on dedicated level 1 processors is another powerful tool for structuring the kernel. The opposite approach, used in operating systems like Multics, TENEX, and OS/360, is to implement kernel operations as subroutines called by users of those operations. Let us call the first approach the multi-process supervisor approach, and the second the distributed supervisor approach.

The multi-process supervisor approach simplifies the handling of types built of multiplexed resources by centralizing the operations that manage those resources in one or more dedicated processes. In such a design, a type manager process is isolated from the processes that request operations on the resources. Consequently, interference with the implementation of the type by processes using the type is precluded.

One advantage of implementing a type manager as a process is that it need not share a data base with other instances of itself acting in parallel. Only the type manager process needs access to the data structures used in managing the objects it implements. The sequentiality imposed by interlocking in the distributed supervisor is achieved by using the sequentiality inherent in the queue of the type manager process. The sequence of actions that may be performed on objects is explicitly represented in the programs of the type manager process, rather than implicitly in the locking protocols.

Another advantage of implementing a type manager as a process on a dedicated processor is isolation of its environment and control point from accidental (or intentional) interference. As noted above, the environment of a type manager executing on its own dedicated processor need not be managed by the same manager that performs the complex operations needed to manage user process environments. This simplifies the dependency structure by eliminating environment dependencies. Similarly, the multiplexing of processor resources that provides resources to type managers need not include the complexity of the resource controls used to limit user process resource usage. On the other side, the implementation of user process environments and scheduling algorithms for user processes need not take into account the special requirements of user processes when executing kernel algorithms (such as protecting the process from destruction while in the kernel or protecting the kernel type manager environment from tampering). In any case, taking these requirements into account would probably result in a cyclic dependency.

The allocation of kernel type managers to dedicated level 1 processors also aids the principle of least privilege. Each type manager need have only the privileges necessary to access its own data bases. This principle can be enforced by restricting the environment (by controlling the set of descriptors in the descriptor segment) of the type manager processes. In a distributed supervisor, on the other hand, the kernel operations have access to more objects than they need. For example, in the present Multics, every kernel operation has access to all objects in the environment of the user process that invokes it. An operation that maps a page into primary memory has the capability of simultaneously copying data from one user object to another. Thus, in a distributed supervisor, each supervisor operation must be inspected to see that it does not do additional operations extraneous to its function. The multi-process structure provides a natural mechanism for mutual protection.

Finally, the multi-process structure helps simplify the structure of the system by avoiding the need to specify unnecessary ordering constraints. An example of this can be found in the design of a multi-process page control by Huber. The page removal algorithm is only indirectly coupled to the algorithm that handles page faults. Each page fault requires using up a page frame in primary memory, but waiting until a page fault occurs to write pages out of primary memory would result in unnecessary delay. To avoid this delay, the pages that are to be written should be located and the write started by a predictive algorithm, which is very hard to fit into a page manager that is invoked only on each fault. A much better structure would be to implement the page removal algorithm as a process that controls the rate of removal of pages in a way that is only loosely coupled to the fault sequence. The page removal algorithm can then easily be designed to run at the optimal times, rather than being constrained to execute only at page fault time. This use of processes also exemplifies the principle of least privilege, because the faulting process need never touch a page other than the one it requires (and presumably has access to). In a distributed supervisor, where removal is done at fault time, the fault handler doing the removal must touch pages that the user process should not have access to.

4. Impact on the Kernel Design Project

The work of Janson, Huber, and Reed has led to a fairly cohesive and implementable kernel design. Janson and Reed have worked out a structuring of the Multics kernel into modules that manage one abstract type each. The use of processes to structure the kernel has been investigated by Huber and Reed.

The status of the use of these ideas in the design of a Multics kernel varies. Huber implemented and tested his use of processes in page control in a special version of Multics. Reed has proposed a detailed design for the two levels of processor multiplexing. A test implementation of part of this design is in progress. Janson has proposed a very detailed structure for the virtual memory management portion of the Multics kernel.

5. Related Activities

In addition to the closely interrelated activities just mentioned, two other activities in the kernel design project either were completed or achieved significant progress during the year:

- a. Rajendra Kanodia and Reed completed an internal report describing the use and implementation of the "eventcount" process coordination model. Basically, eventcounts are semaphore-like coordination variables that are constrained to take on monotonically increasing values. Coordination of parallel activities is achieved by having a process wait for an eventcount to attain a given value; one process signals another by incrementing the value of an eventcount. Any coordination problem for which a solution has been developed using semaphores can be easily converted to a solution using eventcounts. In addition, many eventcount solutions seem to have the property that most eventcounts are written into by only one process; this reduction in write contention has beneficial effects on security problems and on coordination of processes separated by a transmission delay, as in a "distributed" computer system. Eventcounts provide a solution to the "confined readers" problem, a version of the readers-writers coordination problem in which readers of the information are supposed to be confined in such a way that they cannot communicate information to the writers. Finally, for the class of synchronization problems encountered inside an operating system kernel, eventcounts appear to lead to simple, easy-to-verify solutions.
- b. Warren Montgomery's thesis and trial implementation establishes that it is practical to remove many of the traditional constraints on process creation without creating problems for security or resource administration. The concern here is that when a process is created, e.g., in response to a user's dial-up and request for service, the designation of the principal identifier for the new process must be done correctly, or else all access control will be worthless. For this reason, process-creating programs of the "network logger," the "answering service" and the "absentee user manager" have been considered sensitive, privileged programs. Montgomery's approach is to allow any process to request creation of other processes without restraint on principal identifiers proposed; control is provided by associating with every principal identifier a designated starting procedure for the new process. This starting procedure checks to see if proper identification has been submitted by the requestor of the creation. By decentralizing this check, making it the responsibility of the concerned party, a strategy parallel to that of entering a protected subsystem (at a designated starting point) has been created. The result is to remove from the security kernel several large programs previously thought to require certification.

With the completion of the activities described above, the majority of work planned for the kernel design project is finished. We expect that the coming year will

see the completion of the remaining research tasks for this project, and a final report; activity will continue, however, to provide support and technology transfer to the larger Air Force/Honeywell project of which this work has been a part.

C. OTHER ACTIVITIES

The division carried out several other activities, which can be loosely described as experiments in issues of intercomputer network connection:

1. National Software Works

Douglas Wells has worked to help define and criticize the protocols that underly the National Software Works, and also to design and implement the software required to make Multics a participant (a "tool-bearing host") in the National Software Works. This activity has proceeded effectively, and we expect to have Multics participate in early demonstrations of NSW capabilities. The NSW requirements have been met with minimum modification to Multics, although the opportunity was taken to slightly remodularize the Multics ARPANET Network Control Program and the libraries of network support programs to provide more effective support of NSW. One interesting result of this work (which involves judicious replacement of library routines) is that most Multics programs can be expected to work as NSW "tools" with little or no modification.

2. Multics/ARPANET Technology Transfer

This activity concerns the software developed at M.I.T. to attach Multics to the ARPANET; the objective is to have Honeywell make this software a standard product option of the Multics system. During the year, meetings were held with several interested parties concerning attachment of Honeywell's Phoenix Multics site to the ARPANET, and further discussions were held regarding the amount of effort required to make the software into a standard product. Progress has been slow but movement is perceptible. Also, during the year, a few minor changes were made to the software to keep it in step with changes made to the rest of Multics.

3. Local Network

A project was begun during the year by Kenneth Pogran to design a local network to interconnect the several PDP-10, PDP-11, and Multics computers used by the laboratory, and to provide a "gateway" to the ARPANET so that computers at the laboratory that are not ARPANET hosts can access the ARPANET. The first draft of an implementation proposal for the local network was completed at year's end, and is expected to be available early in the coming year.

Two alternative technologies were considered for the network: a ring network, such as that developed by D. Farber at the University of California at Irvine for the

Distributed Computing System, and a packet broadcast net, such as the Ethernet developed by R. Metcalfe and D. Boggs at the Xerox Palo Alto Research Center. We have concluded that nearly identical function is provided by the two technologies, and have elected to implement a version of the packet broadcast net using host interface hardware containing packet buffers, introducing the concept of a "buffered packet broadcast net." This concept was first suggested by R. Greenblatt of the M.I.T. Artificial Intelligence Laboratory. It has been our goal to design the network hardware interface seen by a host to be as independent as possible of the final choice of underlying network technology, making it possible to adopt some other technology in the future, if appropriate.

Protocols for use with the network are being devised and reviewed at the present time. Another goal of ours has been to design the network and its protocols so that they can cover the needs of an organization the size of M.I.T., which could potentially involve computers numbering in the hundreds, and terminals numbering in the thousands. Our current thinking along these lines is to organize a campus-wide network as a group of interconnected subnetworks. Our design will, in effect, make the laboratory's network the first subnetwork of this future campus-wide network.

A third goal for the local network is to eliminate the need for the laboratory to have six or more separate ARPANET attachments and an ARPANET TIP. Purely local, intra-laboratory communication should not burden our ARPANET IMP and TIP, as it does today.

Publications

1. Pograd, Kenneth T. "Network Users' Supplement to the Multics Programmers' Manual." (Initial Issue) M.I.T., Laboratory for Computer Science, Cambridge, Ma., January 1975.

Theses Completed

1. Bratt, Richard. Minimizing the Naming Facilities Requiring Protection in a Computer Utility. M.I.T., Laboratory for Computer Science, LCS/TR-156. Cambridge, Ma., 1975.
2. Rudisin, Gerard. "Multics Implementation of a Server for the ARPANET RSEXEC Protocol." unpublished B.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, 1975.

Theses in Progress

1. Benjamin, Arthur. "Improving Information Storage Reliability Using a Data Network." S.M. and E.E. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, September 1976.
2. Goldberg, Harold. "Protecting User Environments." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, November 1976.
3. Hunt, Douglas. "A Case Study of Intermodule Dependencies in a Virtual Memory Subsystem." E.E. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, September 1976.
4. Huber, Andrew. "A Multi-Process Design of a Paging System." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, December 1976.
5. Janson, Philippe. "Using Type Extension to Organize Virtual Memory Mechanisms." Ph.D. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1976.
6. Kent, Stephen. "Encryption-Based Protocols for Interactive User-Computer Communication." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, May 1976.

7. Montgomery, Warren. "A Secure and Flexible Model of Process Initiation for a Computer Utility." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1976.
8. Reed, David. "Process Multiplexing in a Layered Operating System." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1976.
9. Feiertag, Richard. "A Methodology for Designing Certifiably Secure Computer Systems." Ph.D. Thesis, M.I.T., Department of Electrical Engineering and Computer Science.

Talks

1. Clark, David. "Engineering a Security Kernel for Multics." University of Southwestern Louisiana, Lafayette, La., November, 1975.
2. Clark, David. "Multics Computer Systems Research at Project MAC." Honeywell Information Systems Inc. Multics Symposium, Cambridge, Ma., December, 1975.
3. Feiertag, Richard. "A Methodology for Designing Certifiably Secure Computer Systems." University of Illinois, Urbana-Champaign, Il., March, 1975; Purdue University, Lafayette, In., March, 1975; University of California at Berkeley, Berkeley, Ca., March, 1975; Stanford Research Institute, Menlo Park, Ca., March 1975; IBM San Jose Research Laboratory, San Jose, Ca., March, 1975; University of Utah, Salt Lake City, Ut., March 1975; Cornell University, Ithica, N.Y., March, 1975; City College of New York, New York, N.Y., April, 1975.
4. Hunt, Douglas. "Implementing Extended Type Objects Using an Access Control List Protection Mechanism." Institute of Advanced Computation, Sunnyvale, Ca., June, 1975; Stanford Research Institute, Menlo Park, Ca., June, 1975.
5. Hunt, Douglas. "A Case Study of Intermodule Dependencies in a Virtual Memory Subsystem." Sperry Research Center, Sudbury, Ma., October, 1975.
6. Kanodia, Rajendra. "Eventcounts: A New Model of Process Synchronization." Institute for Advanced Computation, Sunnyvale, Ca., August, 1975.
7. Pogran, Kenneth. "Introduction to the ARPA Network." Communications Forum at Honeywell Information Systems Inc., Phoenix, Az., July, 1975.
8. Pogran, Kenneth. "Multics and the ARPA Network." Rome Air Development Center, Rome, New York, May, 1975.

9. Redell, David. "Proprietary Subsystems and Personal Computers." IBM San Jose Research Laboratory, San Jose, Ca., November, 1975.
10. Redell, David, and Clark, David. "Protection of Information in Computer Systems." Eleventh IEEE Computer Society Conference, Washington, D.C., September, 1975.
11. Saltzer, Jerome. "Computer Science at M.I.T." University of Southwestern Louisiana, Lafayette, La., November, 1975.
12. Saltzer, Jerome. Current Research on Information Protection." Honeywell Symposium on Privacy and Security, Phoenix, Az., April 1975.
13. Saltzer, Jerome. Session Chairman. "The Impact of Technology on System Organization." AFIPS National Computer Conference, Anaheim, Ca., May, 1975.

Committee Memberships

Kanodia, Rajendra, National Software Works Protocol Committee

Pogran, Kenneth T., ARPA Message Service Committee

Pogran, Kenneth T., ARPA Committee on Computer-Aided Human Communication

Saltzer, Jerome H., ARPA Secure Systems Working Group

Wells, Douglas, ARPA National Software Works Protocol Committee

DATA BASE SYSTEMS

Academic Staff

M. Hammer, Group Leader

Graduate Students

D. Carnese
A. Chan
B. Daniels
M. Essrig

R. Grossman
D. McLeod
H. Okrent
S. Sarin

Undergraduate Students

B. Mau

W. Van Roggen

Support Staff

M. Nieuwkerk

DATA BASE SYSTEMS

A. INTRODUCTION

The research efforts of the Data Base Systems group are directed towards two ends: automating human decision-making in various areas of data base organization, maintenance and use, and thereby improving the performance and reliability of data base systems; and developing new functional capabilities unavailable in conventional data management systems. Though our results should be applicable to a wide variety of data bases, our principal concern is with large and/or complex data bases, where conventional brute-force processing techniques are inadequate. Our current focus is in three areas: self-organizing data bases and distributed data base systems.

B. SELF-ORGANIZING DATA BASES

In order for data bases to be effectively used, the data management systems that support them will have to manifest two important characteristics: data independence and non-procedural access. By data independence we mean that users and their application programs are shielded from knowledge of the actual physical organizations used to represent their data, and concentrate instead on a logical view of the data. This makes the data base easier to use and avoids the need for application programs to change when the data base's physical structure is reorganized. Non-procedural access also makes the data base easy to use; this means the use of access languages which allow the specification of desired data in terms of properties it possesses rather than in terms of the search algorithm used to locate it in the data base.

The relational model of data has been proposed as a means of achieving these two goals. The user of a relational data base is provided with a simple and uniform view of his data, a logical view which is completely independent of the actual storage structures used to represent this data. The simplicity of this logical data structure lends itself to access by means of easy-to-use languages, which provide associative referencing (content addressing) of the data base contents.

Because of the distance of the user's view of a relational data base (and of his queries against it) from the realities of the data base's organization, more responsibility is placed on a relational data base system than on a conventional system. This responsibility takes two forms: choosing the physical representation for a relation; and optimizing the execution of queries against a relation, making optimally efficient use of the available access structures. Relational data base systems must possess "intelligence" in order to make decisions in these areas, which have heretofore been the province of human decision-makers.

We believe that the selection of good storage structures is the primary issue in relational data base implementation, since the efficiency that can be achieved by a query

optimizer is strictly delimited by the available storage structures. Furthermore, the efficient utilization of a data base is highly dependent on the optimal matching of its physical organization to its mode of use, as well as to other of its characteristics (such as the distribution of values in it). (For example, certain data base organizations are suitable for low update-high retrieval situations, while others yield optimal performance in opposite circumstances.) Hence, the usage pattern of a data base should be ascertained and utilized in choosing its physical organization. In addition, when viewed as the repository of all information used in managing an enterprise, an integrated data base can no longer be considered as a static entity. Instead, it is continually changing in size, and its access requirements gradually alter as applications evolve and users develop familiarity with the system. Accordingly, the tuning of a data base's physical organization to fit its usage pattern must be an ongoing process.

In current relational data base systems, the system chooses the representation for a relation without any direct information as to its anticipated mode of use. The data base administrator (DBA) may make recommendations to the system about desirable auxiliary access structures, but these judgments are based largely on intuition and on a limited amount of communication with some individual users. For large integrated data bases, a more systematic means of acquiring information about data base usage, and a more algorithmic way of evaluating the costs of alternative configurations will be essential. A minimal capability of a data base management system should be the incorporation of monitoring mechanisms that collect usage statistics while performing query processing. A more sophisticated system would sense a change in access requirements, evaluate the cost/benefits of various reorganization strategies, and determine an optimal organization to be recommended to the DBA; eventually, such a system might itself perform the necessary tuning.

We are involved in the development of a facility that monitors the use of a relational data base and chooses near-optimal physical storage structures based on the evidenced mode of use; furthermore, shifts in usage pattern will be detected and will result in timely reorganization of the data base. As a first cut at this problem, we have concentrated on the problem of index selection. A secondary index (sometimes referred to as an inversion) is a well-known software structure which can improve the performance of accesses to a relation (file). For each domain (field) of the relation that is inverted, a table is maintained, which for each value of the domain in question contains pointers to all those tuples (records), whose contents in the designated domain is the specified value. Clearly, the presence of a secondary index for a particular domain can improve the execution of many queries that reference that domain; on the other hand maintenance of such an index has costs that slow down performance of the data base updates, insertions, and deletions. Roughly speaking, a domain that is referenced frequently relative to its modification is a good candidate for index maintenance. The choice of which (if any) domains to invert must be done with care, a good choice can significantly improve the performance of the system, while a bad selection can seriously degrade it. The goal of our system is to make a good choice of those domains for which

to maintain secondary indices, based on how the data base is actually used.

The operation of the initial version of our prototype system can be described as follows. The specifications of data base interactions, by both interactive users and application programs, are expressed in a non-procedural language; these are first translated into an internal representation made up of calls to system level modules. The language processor has available to it a model of the current state of the data base, which contains, among other things, a list of the currently maintained set of secondary indices, plus various information about these indices. Using this information, the language processor can choose the best strategy for processing each data base operation in the current environment. Statistics gathering mechanisms are embedded within the system modules that interpret the object code of the language processor; they are used to record data concerning the execution of every data base transaction. The statistical information gathered for a run is deposited in a collection area and is summarized from time to time. When the reorganization component of the system is invoked (which will occur at fixed intervals of time), the statistical information collected over the preceding interval is combined with statistics from previous intervals and used to obtain a forecast of the access requirements of the upcoming interval; in addition, a projected assessment of various characteristics of the data in the data base is made. A near-optimal set of domains for which indices should be maintained is then heuristically determined; optimality means with respect to total cost, taking into account the expense of index storage and maintenance. This minimal cost is compared with the projected cost for the existing set of indices. Data base reorganization is performed only if its payoff is great enough to cover its cost as well as that of application program retranslation.

We have implemented a first version of our index selection procedure and are currently experimenting with it. We have compared its performance with that of a system which selects optimal indices by exhaustive search; our system almost always finds the optimal index set in a small fraction of the time expended by the exhaustive scheme. We plan to experiment further with our heuristics, determining their efficacy for a wide variety of usage patterns, and seeing how they can be further speeded up without sacrificing accuracy. We have plans to adapt our system to reflect the cost parameters of an actual operational data management system, probably the Datacomputer or RDMS. We will then be able to analyze actual usage histories of data bases maintained by that system, and select indices for them. In addition, we will be able to study how well our predictive schemes serve to detect developing trends of actual data base interaction, as well as the payoffs and hazards of self-adaptive data base systems.

We are also beginning to enrich our model of data base interaction and allow a variety of complex data base operations in the access language. Concomitant with this, we are expanding the range of organizational decisions which our system will automatically make. We have begun consideration of the issue of clustering, which roughly means the selection of the field on which to sort the file; the choice of clustering field has an enormous impact on the efficiency of complex data base operations

such as join (inter-record correlations). We have also started to look at the problem of horizontally partitioning a file into subfiles, with some fields of a record in one subfile, the rest in another. Queries that do not utilize fields from both subfiles can then be processed very rapidly.

For the future, we plan to expand the scope of our effort and automate more and more data base organizational decisions. We are also beginning to consider the global optimization problem of making all these decisions simultaneously.

As a spin-off of the foregoing work, we expect to develop a query cost estimator. It is often easy for a naive data base user to ask a simple-to-phrase query that will take a great deal of computation resource and time to answer. Frequently, the value of this information to the user is not commensurate with the resources expended to obtain it. Accordingly, the data base system should estimate the resources needed to evaluate a query and provide a user with an estimate of this cost. The information needed to provide such an estimate is a knowledge of the data base's physical organization and a summary of the characteristics of the data in it; in short, it is the same information needed for the self-organization function discussed above. Of course, such a cost estimator must be efficient and not consume any substantial resources in its operation.

C. ERROR DETECTION

The second major focus of our effort is on data base semantics, and especially its application to the detection of errors in data. Conventional data base systems do not possess any knowledge of the meanings of the data structures which they maintain and manipulate. Consequently, numerous problems which are related to the semantics of the data are currently handled in an entirely ad hoc and unsystematic fashion. One of the most important of these issues is the detection of errors in new data being submitted to a data base by examining it for the violation of rules governing the application world of the data base.

In most conventional systems, erroneous data is detected by means of ad hoc, special purpose application programs, which "edit" transactions with the data base and subject them to a variety of validity checks. In any but the simplest cases, this approach breaks down; the data checking programs are expensive to construct, inefficient and unreliable in their operation, and virtually impossible to change. We believe that the specification of the semantics of a data base should be done as part of the definition of the data base itself; then the data management system can assume responsibility for the detection of erroneous data which violates the "semantic integrity" of the data base.

One approach that we have been taking is based on the notion of "constraints," logical predicates on the data base which must hold after every data base transaction. Our efforts have concentrated on refining and structuring this idea, and investigating the considerable problems involved in achieving an efficient implementation of a constraints

driven error detection system.

We have attempted to give some structure to the constraint specification process by classifying semantic integrity information into several components, including the nature of each constraint, the occasions at which it is to hold, and the response to be taken upon the detection of its violation. We have also analyzed the kinds of rules that might be stated about a data base, to the end of providing a structured framework for describing the semantic integrity requirements of a data base. Rather than allowing arbitrary predicates of the first-order calculus, we want the DBA to follow a well-directed specification methodology which reflects this view of the data base's meaning.

We are also involved in the design and implementation of a semantic integrity subsystem for a relational data base system. This system must check for the violation of the specified rules on the occasion of each transaction with the data base. A major issue here is efficiency, since the semantics of a real data base may be described by a great many constraints, each of whose verification could involve accessing much of the data base and so be very time-consuming. The approach we are taking is twofold: for a given data base transaction, to determine as precisely as possible the set of rules, to determine those auxiliary access structures which would be of greatest use in checking the rules. This latter issue depends heavily on the updating and accessing patterns of the data base. This becomes a global optimization problem, of choosing a strategy for integrity checking that will be efficient in the overall context of the data base.

We are also at work at a more powerful error detection system, which will be even more "intelligent" in the detection and processing of erroneous data. Some of our design goals for this system are as follows:

1. The resource expended to detect a particular type of error situation shall be related to the seriousness of that error condition in the information and decision-making system as a whole.
2. Information about the actual occurrence rates of different error types shall be used to allocate available computational resources in the most effective fashion.
3. A model of the data gathering and transmission process will be used to identify the nature of a detected error and to drive the error correction process.
4. The system should be able to operate at a remote site from the data base itself, accessing the data base only when necessary.

A complex system of this kind will need a richer mechanism for describing the world-model of the application domain than is afforded by constraints. We are currently surveying the various knowledge-representation techniques in order to determine which is most appropriate for our environment.

The semantic model of a data base can be used for purposes other than error detection. We are investigating how it could be used to support an easy-to-use query language, in which the user expresses his request in terms of the semantic constructs of his domain, rather than the data structures of the data base.

We are building an error detection system for the ARPA/Navy Command and Control Testbed data base, which resides on the Datacomputer. Our system will utilize a world model of that domain to detect errors in data typical of the kind submitted to such data bases in practice.

D. DISTRIBUTED DATA BASES

We are just beginning to examine the problems of using and maintaining a distributed data base in a network environment. Our principal motivation is to provide the data management support needed by a real-time command and control information system. The requirements of such a system far exceed the capabilities of current data management systems. Multiple copies of a data base may exist at different sites in the network, and they must be consistently maintained. The location of a data base may dynamically change, upon command of the DBA. A user should not have to know the location of the data base he wishes to access; an access planner should locate for him the copy which will give him the best service. Furthermore, the entire system must be very robust: should one of the hosts be disconnected because of communication failure, the rest of the network must contain to function, and the isolated node must operate on its own; when connection is re-established, global consistency must again be achieved.

We believe that each of these problems is individually soluble, but that the real challenge will lie in providing all these features simultaneously. Our approach is based on establishing priorities among conflicting goals, accepting some (controlled) inaccuracy in the lesser ones in order to concentrate on those that are more important.

Publications

1. Boyce, R.; Chamberlin, D.; Hammer, M.; and King, W. "Specifying Queries as Relational Expressions; the SQUARE Data Sublanguage." Communications of the ACM, Vol. 18 No. 11 (1975), 621-629.
2. Hammer, M. and McLeod, D. "Semantic Integrity in a Relational Data Base System." Proceedings of the ACM International Conference on Very Large Data Bases. Framingham, Massachusetts, September 1975.
3. Hammer, M. "The Design of Useable Programming Languages." Proceedings of the ACM National Conference. Minneapolis, Minnesota, October 1975.
4. Hammer, M. "Data Abstractions for Data Bases." To be published in Proceedings of the ACM Conference on Data.
5. Hammer, M. and Chan, A. "Index Selection in a Self-Adaptive Data Base Management System" To be published in Proceedings of the 1976 SIGMOD International Conference on Management of Data.
6. Hammer, M. "Error Detection in Data Base Systems." To be published in Proceedings of the 1976 National Computer Conference.
7. Hammer, M. and Chan, A. "Acquisition and Utilization of Usage Patterns in Relational Data Base Implementation." To be published in Proceedings of the IEEE Joint Workshop on Pattern Recognition and Artificial Intelligence.
8. McLeod, D. "High Level Domain Definition in a Relational Data Base System." To be published in Proceedings of the ACM Conference on Data.

Theses In Progress

1. Chan, A. "Automatic Selection of Inversions in an Integrated Data Base Environment." M.S. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1976.
2. Grossman, R. "Data Base Applications of Constraint Expressions." unpublished S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, January 1976.
3. Mau, B. "Automatic Index Selection on the Datacomputer." S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, January 1977.

4. McLeod, D. "High Level Expression of Semantic Integrity Specification in a Relational Data Base System." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, June 1976.
5. Okrent, H. "Synthesis of Data Structures from their Algebraic Description." Ph.D. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1976.
6. Sarin, S. "Design of a Semantic Integrity Subsystem for Relational Data Base Systems." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, January 1977.
7. Sheckler, D. "SEQUEL on RDMS." S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, August 1976.

Talks and Presentations

1. Hammer, M. "The Design of Usable Programming Languages." ACM 1975 National Conference, Minneapolis, Mn., October 21, 1975.
2. Hammer, M. "Error Detection in Data Base Systems." Bureau of Naval Personnel, Arlington, Va., November 4, 1975.
3. Hammer, M., Session Chairman, "Database Design Tools." International Conference on Very Large Data Bases, Framingham, Ma., September 23, 1975.
4. McLeod, D. "Semantic Integrity in a Relational Data Base System." International Conference on Very Large Data Bases, Framingham, Ma., September 22, 1975

DOMAIN SPECIFIC SYSTEMS RESEARCH

Academic Staff

S. A. Ward, Group Leader
M. L. Dertouzos

P. G. Jessel
J. Weizenbaum

Graduate Students

R. Bahr
G. Bunza
C. Cesar
S. Y. Chiu
J. Gula
R. Haagens
R. Halstead
B. Hampson
C. Hayseen
E. Israeli

D. Kinney
Y. Lee
B. Musicus
J. Pershing
J. Poulos
B. Robinson
T. Teixeira
C. Terman
J. Valvano

Undergraduate Students

J. Avery
C. Baker
T. Bell
T. Dutton
D. Emberson
I. Gerson
J. Grossman

S. McConnel
R. Ottolini
M. Partick
D. Slutz
S. Thomas
C. Topolcic
R. Vanderkloot
M. Ziegler

Support Staff

N. An
R. Kao

J. Pinella
N. MacKenzie

DOMAIN SPECIFIC SYSTEMS RESEARCH

A. INTRODUCTION

This year formed the foundation for our future research. It centered on the development of a conceptual framework consistent with the various activities of the group and the establishment of an integrated laboratory facility.

The group's principal goal remains the development of an integrated system for the efficient design of domain specific systems. However, during the past year, we have attempted to focus our work on more specific research topics: automatic code generation, microprocessors as substitutes for random logic, real time block diagram schemata, languages and architectures for real time control and multiprocessor networks.

Construction of the new laboratory facility was begun during the current year. The host PDP 11/70 was delivered in September 1975 and DELPHI operating system was converted to run on the new machine.

In addition, the first implementation of our common system language, D, was developed and several system programs for microprocessors were written using D.

Unfortunately, the construction of new facilities for the hardware laboratory were delayed by space problems. A final plan has now been approved and we expect to take occupancy by August 1976. Most equipment has been ordered and we expect the laboratory to become operational by October 1976.

B. DOMAIN SPECIFIC SYSTEMS

1. Continuing Work on Automatic Code Generation

Research by C. Terman on a domain-specific production language for the implementation of code generators, described in the group's report for last year, is proceeding according to schedule. Similar approaches have attracted attention elsewhere in recent months, and have demonstrated that current global optimization techniques can be coded in a language based on an attribute grammar. Unfortunately, such efforts rely heavily on the Turing universality of the strictly deterministic parsing algorithm used, and require that the program synthesize ad-hoc attributes which play the role of local variables having no natural significance in the semantics of code generation.

Our approach differs primarily in that it assumes a global goal orientation, namely the minimization of the "cost" of the object programs generated. The "cost" metric will be built into our system (rather than being specified in the source language) and will reflect some heuristic combination of space and time criteria, possibly involving user

specified parameters which bias the code generator toward space or time optimization. The goal orientation of our language allows it to assume a declarative rather than procedural form, and leads to source language semantics which more closely reflect the structure of the problem domain. The goal of our project may thus be viewed as a system which synthesizes the local attribute structure of an attribute-grammar-based code generator from the combination of declarative information and a global goal.

2. Microprocessors as Substitutes for Random Logic

A new area of research is attempting to develop techniques for translating traditional random logic structures into microprocessor code. A key element in this work is the specification and translation of timing constraints and tolerances. We have classified random logic structures into four design types:

- a. generators-circuits with either no inputs or completely defined inputs
- b. combinatorial inputs-memoryless circuits with variable inputs
- c. asynchronous-circuits with variable inputs
- d. synchronous-circuits with both variable inputs and internally generated (i.e., generator) outputs

To date we have developed algorithms which can translate circuits of type a.) and b.). We have an algorithm that can take the timing description of a generator (including tolerances) and calculate both the minimum microprocessor operating frequency and operating frequency for which the microprocessor generated output can locally satisfy the tolerances; if such a frequency cannot be found, i.e., the microprocessor is too slow, the simulation is impossible. All multiples of this low frequency are guaranteed to satisfy the tolerances, thus giving us a set of solutions, which however, is not complete, but can be made so by the algorithm. Each solution is represented in matrix form (rows representing changes in time and columns the output data), which, with original random logic description, is the basis of code generation.

In its simple form the program generated will consist of blocks of code, one for each event column in the matrix (an 'event column' is one which differs from its antecedent). Each block has three parts: data preparation, delay, and data output. The first two can be intermixed, but the data output comes at the end of the block. Data preparation calculates the values that are to be sent out. Delay will be inserted to satisfy the timing expressed in the matrix: the total number of microprocessor state cycles for data preparation, delay and data output must be equal to the number of columns between that event column and the previous one.

Because of its relative simplicity and required flexibility (in terms of time), we

have studied delay code generation. Our method uses the microprocessor's idiosyncrasies and knowledge about the microprocessor internal state at the point in the program where the code is to be inserted, to produce the smallest non-iterative program that delays the required amount of time. We have not found an effective similar method for data preparation. We believe that space optimization for that block code will depend on semantic information present in the random logic description.

We have been simulating combinatorial networks from the random logic I/O table for predetermined assignments of external signal lines to input and output port pins. There does not seem to be a unique technique always yielding either the fastest or smallest program.

The PLA Approach, or its near-equivalent, the table look-up, is very consistent in what can be expected of it: fast execution, as only memory indexed accessing is needed, but with memory usage growing with table size. Generating the table instead of storing it, i.e., simulating the corresponding Boolean function, will, for large tables, execute slower on less storage. There are different programming approaches to Boolean function simulation based on some form of "test and branch" or on logical operation instructions. Execution time is constant for the latter, but varies with the input value for the former. Both need not exist in pure form; combining them leads often to better code.

Current research is aimed at developing algorithms for design types c.) and d.) and then integrating these approaches into a single model.

3. Real Time Block Diagram Schemata

A compiler for block diagram schemata which guarantees the real time performance of an object program based on programmer-supplied bounds, analysis of the topology of the source program, and inherent knowledge of the target machine is being developed by T. Teixeira. A major subproblem concerns the automatic choice of object control structures (e.g., priority interrupt vs. asynchronous object time task scheduling) to meet real time performance criteria specified by the user. While the current work focuses on single-processor target systems, we plan to explore the extension of the translator to several special case configurations involving multiple processors. The goal is to achieve a high degree of independence between source language semantics and ultimate implementation, thereby alleviating the programmer's responsibility for making implementation decisions.

4. A Control Language

During the year, several visits to service installations (e.g., Wright Patterson AFB, Kelley AFB, and Naval Electronic Laboratory Command) were undertaken in order to coordinate our research with application areas of current interest to DOD. Particular application areas included Automatic Test Equipment and Avionics and Weapons Control.

From these discussions a need for techniques for computer based control emerged that integrates modern control theory with contemporary computer science practices. In conjunction with control theorists from the Electronic Systems Laboratory at M.I.T., we are currently exploring languages and implementations that lend themselves to real time process control. In particular, we are currently investigating the feasibility of utilizing message passing semantics to express real time scheduling constraints. Such a mechanism would provide a uniform basis for describing both single and multiprocessor systems.

5. The Language D

During the past year, the initial PDP-11 implementation of the language D was completed, and D is currently in active use by members of the group. D was designed to serve as our common general purpose, high level language for both the direct implementation of microprocessor software and for the coding of support tools on large scale host machines. The major design goals of D include machine independence of the source language, provisions for real-time programming, and amenability to the production of efficient object code.

The current implementation of D runs on and produces code for PDP-11/70 and realizes certain of the above goals lack constructs for real-time applications. Further development of D is planned in the following areas:

- a. Mechanisms for limiting access to variables beyond the constraints of conventional block structure, affording a measure of the data abstraction advantages of CLU and SIMULA.
- b. Generalization of numeric data types to promote further machine independence and to allow sophisticated compilers to produce highly optimized code, e.g., by "compilation out" of floating point operations from a program for a target microprocessor.
- c. The addition of control and data structures to support real-time programming, including constructs for interrupt and possibly for multiprocessing.
- d. Extension of the facilities for compile-time processing, including the specified (but yet unimplemented) mechanism for compile-time libraries. The intent is to provide a syntactically transparent mechanism rather than an explicit preprocessor, so that an optimizing compiler might perform at compile time, certain computations which a simple compiler postpones to object time. The initialization of a table of sines might be such a compile-time computation.
- e. The production of complete and effective user documentation of the language.

The close correspondence between the design goals of D and those of CL76 (the currently favored name for the proposed DOD common language) has been noticed by both us and our sponsors, suggesting the potential for mutual influence between these efforts. While the current D implementation falls short of the DOD "Tinman" criteria (as well as our own), anticipated D development will satisfy those DOD requirements which seem attainable and justified. Our hope is that these efforts will converge.

6. Automatic Multitarget Code Generation

A continuing objective of our work is the development of a programming system to translate high-level domain specific language constructs onto target systems that can be uni-processor or multiprocessor in nature. A key link in this down-loading procedure is the development of automatic multitarget code generation techniques. This approach is an attempt to provide an easy means of adapting a high-level language to new computer organizations by allowing all source programs in the language to be compiled for a large set of processors. The present scheme by Bunza involves a system to generate machine code for many different processors using a single compiler and a single code generator, utilizing processor specific databases. Adaption of the code generator to new processor architectures is limited to the creation of a single Processor Description Database.

Various schemes have been examined including simple macro expansion, hierarchical macro interpretation, pseudo machines and abstract machine models, interpretation of abstract machine code and special executive calls, and code to code translation. These do not appear viable in the present context. A new code generation system is proposed which solves most of the recognizable problems and provides an effective solution for realistic implementations. BCPL OCODE is accepted as the intermediate input language and undergoes a series of transformations, determined by a Hierarchical Operator Table and a Processor Description Database. The Processor Description Database is a target machine description interactively built employing a special variant of ISP. The key to the new code generator lies in the separation of functional classes of activity in the system: operator functional decomposition, storage allocation, register allocation, operand addressing, and coding strategy decisions.

7. Shared Bus Structures

Research on bus architectures has focused on dedicated and non-dedicated bus structures, bus arbitration, bus protocol, and deadlock prevention. The current study deals in particular with protocol issues relating to the configuring of multiple processors on a shared bus. Examination of current bus protocols yield major deficiencies in their ability to support large numbers of closely coupled processors. The concept of a "pended transaction" bus is introduced and shown to possess many desirable characteristics necessary for multi-initiator systems. Self-clocking asynchronous logic is used to implement the new protocol. This concept is also extendable to multi-bus systems.

Further studies have demonstrated the fundamental unsuitability of the current generation of microprocessors for high throughput shared bus designs. Benchmarks indicate that no more than three such processors can actively contend for the bus without significant queuing. These results stem directly from the basic structure of the current microprocessors, and reinforce the conclusion that fundamental architecture changes must be made before such shared bus configurations are justifiable. These conclusions have provided direct reinforcement for the pending bus design and additional motivation for bus contention studies.

Queuing theory models for multiprocessor systems have been proposed as an approach to isolating the primary factors which control overall performance. Two major factors were selected for study: processor specialization and interprocessor communications hardware. In particular, attention has focused on processor specialization on throughput, interprocessor communication overhead, and reliability. M master, N slave systems have been extensively studied to analyze system throughput under various conditions. Poisson characterizations have been incorporated in these models with primary interest focusing on the effects of increased number of masters, increased number of slaves per master, and differing request and service rates for operating system routines. Additional features considered have included the effect of IPC message processing overhead on throughput, the effect of non-Poisson distributed serving times, and the complications introduced by a heterogeneous operating system with different service and request rates for different processes. The method of analysis has also been extended to less specialized systems. Elementary performance criteria have been investigated for master/slave systems.

8. Hierarchical Structures

Hierarchical structures offer inherent control mechanisms that are applicable to a wide variety of problem areas, including linear control. Since the group's focus has recently centered about modern control theory and its applicability to the integrated software/hardware system under development, it was decided to study a simple linear control application to gain some knowledge in the area. The demands of many control systems exceed the capabilities of a single processor system; moreover, multiprocessor hierarchical architectures will satisfy certain classes of linear control problems. Consequently, such a hierarchical structure is proposed and shown sufficient to satisfy the real-time constraints required by an example benchmark linear control system. Models of data and control flow within the structure were developed. Alternative structures were investigated and shown to be insufficient for the linear control problem.

9. Ring Structures

Systems requiring uniform access to shared system resources often lend themselves to a ring structure. Selection of such a loop topology is often motivated by low incremental expansion cost, simple interfaces, and a straightforward routing path.

The group has implemented a ring structure, principally as an experimental target vehicle for continued exploration of software/hardware strategies. Four nodes now exist connected via unidirectional, 14 MHZ di-phase encoded serial communication links, in a loop topology. In addition to the communication hardware, each node supports a microprocessor controller and an extensive DMA facility.

Key to the successful performance of this multiprocessor system is the competence of the controlling software responsibility. Message transmission and process scheduling modules have been examined as well as more global functions including file management. A ring operating system has been proposed taking into account the basic properties of process execution rights, levels of process priority, interprocess communication, and the ability to have processes spawn other processes.

10. Integrated Facility

One of the goals of the D.S.S.R. Group is to provide an ideal laboratory environment which facilitates the production and verification of a target microprocessor based system. Domain Specific Systems require customizing both hardware and software. Accordingly, the host should provide an integrated set of aids that allows a domain specific system to be designed and tested as a total system. Several downloading facilities have been developed during the past year including hardware microcomputer systems, cross-software, and a portable computer system.

A truly portable general purpose computer system has been developed to provide remote downloading and field diagnostic capabilities. The system is completely self-contained, incorporating the basic elements of an ASCII keyboard, printer, memory, modem, serial and parallel I/O buses, software monitor-debugger with bootstraps, power supplies, and complete user documentation. The system has room for 32K bytes of memory configurable into ROM and RAM modules. Future enhancements include mass storage capabilities.

Work is also underway to provide a class of design aids which interact closely with both hardware and software. One such tool currently under development (B. Robinson) is a sophisticated debugging interface (DIF) which couples the host computer to the target microprocessor. The DIF is organized around a bipolar microprocessor which:

- a. specifies whether a lead is to be used for either input or output. The desire to use the DIF for a wide variety of microprocessors and for a wide variety of signals associated with a specific microprocessor system implies that all leads be bidirectional, enabling each lead to monitor and control a signal. The direction of any lead can be changed dynamically.
- b. establishes the sampling rate. This will generally be a function of the cycle speed of the target microprocessor.

- c. is capable of generating a sequence of control signals. This allows the DIF to emulate various external signals. Examples include a temperature of 2000 degrees or a sequence of events that occur at the limit of the target system's intended response rate. Depending on the rate at which these signals change, they may be generated directly by the bipolar microprocessor or be buffered.
- d. process the incoming data and compensate for the differences between the bandwidth of the host-DIF link and the DIF-target link. If TTY lines are used this difference may be considerable and a substantial buffer must be provided.

Associated with the Debugging Interface is a considerable amount of software residing in the host. Our goal of removing the designer from the tedium of programming would be compromised if we were to immerse him in the details of debugging his system. Accordingly, the software associated with the DIF entails a Domain Specific Language which allows the user to specify various control and monitoring operations independently of the way they are implemented.

11. Educational Computer Generated Films

We have examined a number of display systems in order to determine their suitability for making the kind of educational computer generated movies we have long planned. We are about to order a RAMTEK display which we consider very well suited to our task.

Professor Weizenbaum has been teaching course 6.030, the introductory computer science subject, and has been alert for places in the subject where movies might be particularly instructive. A number of such places have been identified and appropriate imagery has been discussed. We have, for example, plans to make films illustrating the fundamentals of a LISP interpreter from a number of different perspectives, e.g., from that of the naive user and from that of the control structure. Other examples are: recursion, the contour model for a PL/1-like language, the FUNARG problem, etc.

Publications

1. Ward, S.; Halstead, R.; and Terman, C. D Reference Manual. M.I.T., Laboratory for Computer Science, D.S.S.R. Working Paper 16, Cambridge, Ma., October 1975.
2. Jessel, P.; and Ward, S. "Domain Specific Systems: A New Approach to Microprocessor Based Design." To be presented at EUROMICRO Symposium. Amsterdam: North-Holland, October 1976.

Theses in Progress

1. Cesar, C. "Microprocessors as a Substitute for Random Logic." Ph.D. thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion June 1978.
2. Gula, J. "A Distributed Operating System for an Object Based Network." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion January 1977.
3. Halstead, R. "Multiprocessor Implementations of Message Passing Systems." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion September 1977.
4. Mok, A. "Task Scheduling in the Control Robotics Environment." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion June 1976.
5. Pershing, J. "Design of a Domain Specific Meta-Compiler for Systems Using Graphical Input as a Source Language." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion September 1977.
6. Robinson, B. "A Programmable Microprocessor System Debugger." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion June 1976.
7. Teixeira, T. "Block Diagram Languages for Process Monitoring and Control." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion September 1977.

8. Terman, C. "Descriptive Approach to Automatic Code Generation." S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion September 1977.

Talks

1. Jessel, P.; Chen, R.; and Patterson, R. "A Microprocessor Controlled Data Switch." COMPCON '75, Washington, D.C., September 9-11, 1975. Also presented at the Moore School Research Symposium, University of Pennsylvania, Philadelphia, Pa., October 31, 1975.
2. Jessel, P. "Higher Level Languages for Microprocessors." Mini-Micro Computer Applications Workshop, San Diego, Ca., November 18-20, 1975.
3. Ward, S. "Microprocessor Systems." Guest lecture at M.I.T. Seminar Program in Artificial Intelligence, Machine Vision and Productivity, Cambridge, Ma., June 1975.

KNOWLEDGE-BASED SYSTEMS

Academic Staff

W. A. Martin, Group Leader
L. B. Hawkinson
A. C. Hax

P. Szolovits
G. R. Ruth

Research Staff

S. Alter
R. V. Baron
G. P. Brown
R. Fisher

N. R. Greenfeld
W. A. Kornfeld
G. A. Moulton
A. Sunguroff

Graduate Students

H. G. Baker
V. Berzins
M. Bosyj
R. B. Krumland
W. J. Long

W. S. Mark
M. L. Morgenstern
B. Niamir
S. K. Sarin
W. R. Swartout

Undergraduate Students

C. Kiselyak
B. Levin

G. Thomas

Support Staff

M. S. Belyavski
B. J. Demps

V. E. Lewis

KNOWLEDGE-BASED SYSTEMS

A. INTRODUCTION

Our group is investigating methods by which software system design and implementation can be automated or semi-automated. Our approach is to write programs embodying the techniques used in the manual processes. The challenge is to discover which of these techniques or aspects of them are machinable and to develop formalisms which will make more of them machinable. In this year's report we will give a roster of our current projects and then present survey pieces which explore a project in more depth.

B. ROSTER OF PROJECTS

1. The Very High Level Language SSL

We are developing the prototype, SSL, of a language which could replace COBOL. By omitting the details of data-structure and data-movement bookkeeping the language achieves about a factor of ten compression over a conventional high level language. It allows the coder or maintainer to focus on how outputs are defined in terms of inputs. This should have a dramatic effect on the serious software maintenance problem faced by large business data processing systems. We have a system which "compiles" SSL into IBM 370 PL/I.

We are now at the point where a problem comprising approximately 100 data items and 60 computations has been run through our system in a hand holding and restricted fashion. The system is described at length in the first survey below. People and tasks on this project are:

- a. G. R. Ruth and S. Alter have completed about 1/2 of a monograph describing this work.
- b. G. R. Ruth has also improved the question answerer and JCL generator modules.
- c. R. V. Baron, W. S. Mark, C. Kiselyak, and G. A. Moulton have written an improved PL/I generator.
- d. G. Thomas completed a B.S. thesis defining the report generation language component.
- e. The SSL syntax was designed by the group.
- f. W. A. Kornfeld implemented a parser for SSL.

- g. M. L. Morgenstern got his Ph.D. thesis optimizer to run on the example mentioned above.
- h. R. V. Baron performed overall system management and integration.

2. Automation of the Business Consultant

- a. M. Bosyj completed a M.S. thesis program which questions a user with an on-line multiple choice branching questionnaire and does an initial design of a procurement system for him.
- b. W. S. Mark now has a Ph.D. thesis program which attempts to diagnose business problems expressed in stylized English by identifying troublesome feedback loops in the organization. The program has run on his initial target example.
- c. R. B. Krumland has a Ph.D. thesis program which builds aggregate cash flow models on the basis of a stylized English description of a situation and then makes runs to answer various what-if questions. His program has also run on his initial target example.
- d. W. J. Long is doing Ph.D. thesis work on a program-writing program which will write programs for such simple activities as ticket selling, etc.
- e. R. Fisher improved the KOMS operations management software packages.

3. The OWL System

- a. W. A. Martin has refined his grammar of English.
- b. L. B. Hawkinson has improved his data base system for handling conceptual structures.
- c. P. Szolovits has got a new parser running on one English sentence using Martin's grammar and Hawkinson's data base.
- d. A. Sunguroff with help from W. J. Long and W. R. Swartout got the OWL interpreter to execute OWL blocks world procedures similar in function to those in the M.I.T. Ph.D. thesis of Sussman.
- e. G. Brown has debugged her OWL dialogue routines to the point where several lines of the Susie Software dialogue given in an earlier Project MAC report are operational.

KNOWLEDGE-BASED SYSTEMS GROUP 45 KNOWLEDGE-BASED SYSTEMS GROUP

- f. W. R. Swartout coded a running version of the Pauker-Silverman-Gorry digitalis dosage program in OWL preparatory to adding a question answering facility.
- g. B. Levin did a B.S. thesis which was a study of the non-physical uses of the preposition under.

Publications

1. Hawkinson, Lowell. "The Representation of Concepts In OWL." Fourth International Joint Conference on Artificial Intelligence. Tibilisi, U.S.S.R., Sept. 1975.
2. Ruth, Gregory. The Question Answerer. M.I.T., Laboratory for Computer Science, Automatic Programming Group, Internal Memo 21, Cambridge, Ma., 1975.

Theses Completed

1. Mark, William. "The Reformulation Model of Expertise." unpublished Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, August 1975.

Theses in Progress

1. Baron, Robert V. "Structural Analysis in A Very High Level Language." B.S. and M.S. thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion Sept. 1977.

APPENDIX

A. PROTOTYPE I: AN AUTOMATIC PROGRAMMING SYSTEM PROTOTYPE

Programming is the activity of going from a task specification to code capable of performing the task on some actual computing system. This is essentially a problem-solving process. But over the years people have come to understand certain functions of the programming process well enough to automate them--that is to replace those functions by programs. The most notable results were assemblers, compilers, and operating systems. The gains realized from this automation were reduced operating errors, increased complexity of systems which could be completed and more efficient use of resources (time, people, machines) in the design-implement-evaluate cycle.

Our knowledge and understanding of programming is once again reaching a level where a significant advance in automation is both necessary and possible. In fact, we believe that the entire programming process can now be effectively automated. That is, a system can be developed that will engage the user in English language discourse about a desired task and will produce, as a result of the interaction with the user, a satisfactory program. To demonstrate feasibility and gain insight into the issues and technology involved in creating such a system, a prototype automatic programming system (Protosystem I) for generating business data processing systems is currently being developed at M.I.T.

B. A MODEL OF THE PROGRAM WRITING PROCESS

The data processing system writing process may be conceived as a sequence of phases leading from the conception of a system to its implementation as executable machine code. A useful and plausible model for this sequence of phases is:

1. Problem Definition (English → OWL)

The system specification is expressed in domain dependent terms in English that is understandable by the program developers.

2. General System Analysis and Design (OWL → SSL)

The problem is reformulated in standard data processing terms and expressed as an instance of a known solvable problem class (in our case a subset of the class of all batch oriented dps's). Domain dependent policy and procedures are worked out in detail at this stage.

3. System Implementation (SSL → CDSL)

The system--the actual procedural steps and data representations and organizations--is constructed by intelligent selection from and adaptation of a number of standard implementations possibilities.

4. Code Generation (CDSL → PL/I & JCL)

The design specifications are implemented in a high-level language (e.g. PL/I, COBOL) in a fairly straightforward, but not totally mechanical, way.

5. Compilation and Loading (PL/I & JCL → machine executable form)

A form is produced that can be "understood" and executed by the target computer.

These phases progress from a general notion of what is to be done by the desired system toward a detailed specification of how it can be accomplished. They also represent the classes of design and implementation problems involved in program writing, progressing from the most global and general considerations toward the most local and detailed issues.

Protosystem I seeks to automate the program writing process by automating and tying together the phases described in the model given above. That is, Protosystem I is designed in such a way that there are explicit parts or stages corresponding to each of the model phases. Each such stage embodies the knowledge and expertise of the human agent(s) for the corresponding phase, so that, given the same or similar input, it can intelligently produce comparable corresponding results. Drawing on experience gained in recent artificial intelligence and knowledge-based systems research, we have chosen to represent the knowledge in each stage in the form of procedures as opposed to the approach used, for example, in table driven compilers.

The products of each stage should not be so rigidly deterministic so that the courses of action in further stages are narrowly prescribed. They must be sufficiently general and malleable so that further stages can have the maximum freedom in making their design contributions in the most effective and efficient ways. Consequently, we have chosen in Protosystem I to make the product of each stage a descriptive representation of the dps in terms of concepts and considerations appropriate for the next stage of development. Such a description provides a medium in which the next stage can manipulate relevant concepts and analyze the dps for relevant properties, so that it can perform its design job more in the manner of a problem solver than in that of an automaton. In this way the programming process is conceived as the development of a succession of evermore precise system descriptions until, ultimately, a level is reached where every detail has been decided and the result is an executable computer program.

Put another way, we have borrowed from structured programming the notion of program development by successive refinement, but we have approached this by extending the levels of language--machine, assembly, and compiler--to include the very high level language, SSL, and an English-like language, OWL. We recognize that to succeed with this method one must make appropriate abstractions so that the more abstract statements of the problem lend themselves to further refinement. While the wide use of business data processing systems in the past two decades has not led to the definition of standard modules which would handle any situation, it has led to a better understanding of the useful abstractions in that field. We believe that these abstractions can be grouped naturally according to the phases of the dps development process.

C. EFFICIENCY ENHANCEMENT IN SYSTEM DEVELOPMENT

To produce a credible and practical result an automatic programming system must perform a reasonable degree of optimization. Current formal optimization methods pertain mainly to the compilation level, principally because this is the only phase of the program writing process that has been automated. When the entire program development process is automated, new, additional types of optimization will have to be included. The combination (for the sake of I/O efficiency) of computations accessing the same data set is an example. At compilation time the decision to include these computations in the same job step or not to do so has already been made. If they are not in the same job step (and hence compiled separately) the opportunity for optimization has already been lost. Even if they are in the same job step, the compiler can derive little, if any, information about their I/O characteristics. Therefore, it has no basis on which to evaluate the relative efficiency of possible combinations and is incapable of making an intelligent optimization decision.

This type of optimization problem is not unique. It should be easy for the reader to think of many other examples where it is impossible to perform adequate optimization of the type necessary if we wait until Phase 5 to do it. The information needed to make good design decisions of a more global nature is just not available at that stage. That is, the (so-called "high level") compiler language is too low level to allow the system specification to be expressed in a form where such optimization issues will be apparent.

The various possible types of optimizations fall quite naturally into categories that correspond to the program writing levels in our model. For instance, the combination of computations as in the above example is something that should be considered during Phase 3 (system implementation) where the data and computational interrelationships among conceptual processing units are most evident. Problems involving machine language inefficiencies should be handled in a later phase.

An attempt to apply an optimizing process at a higher stage than that to which it pertains would require an overspecified system description at that level; that is, a description containing details extraneous to the purpose of that stage. Trying to apply a

KNOWLEDGE-BASED SYSTEMS GROUP 50 KNOWLEDGE-BASED SYSTEMS GROUP

transformation at a later stage than that to which it naturally corresponds would effectively require "unprogramming" (translation from a lower level description to a higher level one). This would be a difficult, if not impossible, task. It would also require the lower phase to contain knowledge which belongs at a higher level. Optimizations are most effectively performed at their corresponding level of translation, where exactly the sort of information and visibility needed is present. Since all levels of program writing are included in our automatic programming system, there is no need for overspecification or unprogramming.

At each stage in an automatic programming system intelligent, efficiency enhancing design and implementation require: knowledge of possible ways of implementing each requirement determined and described by the previous stage, methods for evaluating alternate designs so that the best choice can be made, and information on which such decisions can be based. The first two of these embody the expertise of the human agents for the corresponding phase of each stage and are built into the programs for that stage.

The information used to make decisions is specific to the particular dps construction project. It consists of three parts: (1) the design decisions that have been made so far for the dps part under consideration and for all parts relevant to the further development of that part, (2) the consequences that a decision will have on further development, and (3) the environment in which that part and related parts will operate. The first of these is provided by the description output of the previous stage, which is input to the present stage. It also includes the current partially determined description of the dps being produced by the present stage. This contains the decisions that have been made so far in this stage.

The second type of information is of two kinds: (a) knowledge of the effects of a decision on others to be made in the same stage and (b) knowledge of its effects on decisions to be made in later stages. Information of type (a) is provided by maintaining a global awareness within each stage of its aggregate design contribution. Type (b) information is commonly provided by feedback in the non-automated program development process. Consider the problem of avoiding machine language inefficiencies. A number of such inefficiencies can be eliminated simply by finding better sequences of instructions to implement the constructs of the next higher level language; but others can only be prevented by using only algorithms (which are determined at a higher level) that require constructs that can be efficiently implemented. In the latter case one could imagine a strong interaction between levels, for in choosing an algorithm it would be necessary to determine whether the constructs it requires could be implemented efficiently. We believe that in the semi-repetitive design of business data processing systems this strong interaction is not required. We eliminate most difficulties by removing from the design any algorithm which has not been part of an efficient implementation.

We can do this because our goal is not to be the first to create radically new systems, but to implement standard systems quickly, cheaply, and accurately. The semi-repetitive nature of the design of dps software also makes it possible to get ball-park estimates of cost without detailed coding of key sections. One could cite, for example, estimating systems like SCERT which have been in commercial use for some time.

As stated above, we take the view (also held by some structured programming advocates) that one design phase should be completed before the next is begun. Because we are automating the design process, we can make many passes through the whole process in the time formerly required for a single pass. Feedback from several passes, including evaluation by the using organization, we hold to be critical. Feedback among phases as they run greatly complicates the design process. We feel that it will not yield a corresponding improvement in results. It opens the door to the design of incredibly complex "heterarchical" systems where control of the program development process would dance unpredictably among the various stages in a complicated way. To avoid the complexities of a heterarchically structured system, we have outlawed feedback. Instead, each stage has a gross model (often implicit) of the stage following it. In this way it can see the basic ramifications of its design decisions in the next stage by effectively interrogating its own model of that stage, rather than having to rely on that stage to feed information back.

The third type of information needed in the design and implementation process has to do with the context in which the dps will operate, namely: (a) the machine/operating-system configuration on which the ultimate dps code will be executed and (b) the characteristics of the data it will receive and produce. Because machine/operating-system configurations are standard and relatively few in number, type (a) information is encoded directly into the automatic programming system in the form of separable, interchangeable modules; the automatic programming system is thus specialized to a particular configuration by selecting the appropriate module (e.g. the OS/360 module). Further installation dependent information (e.g. the number and type of secondary storage devices) is supplied directly by the user. Information of type (b), concerning types and quantities of, and interrelationships among, data processed by the dps, is too broad and varied in nature to be entirely supplied by, or derived from, an initial user statement of any reasonable length. The inclusion of all facts that might possibly become relevant to design decisions would require much effort. Much information is difficult to obtain and most would never be used. Therefore, it is best to require special information from the user only when it becomes important in the course of the development process. To do this the automatic programming system must be able to ask the user specific questions as additional information becomes necessary.

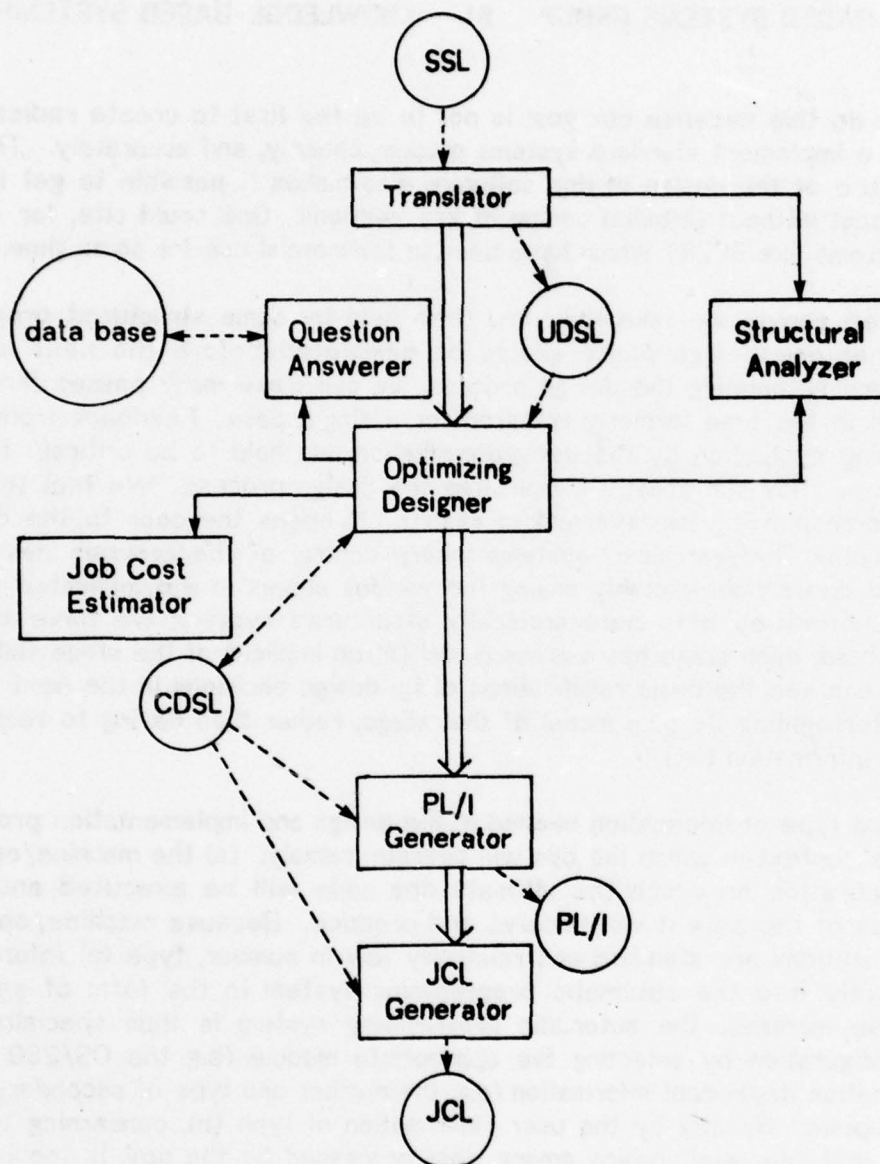


Figure 1. Protosystem I: Structure of the Bottom Part

D. THE DEVELOPMENT OF PROTOTYPE I

The research and development of Protosystem I at M.I.T.'s Laboratory for Computer Science (formerly Project MAC) began in 1971. Early on it became apparent that the natures of the technologies to be used in the first (or "top") part of the system (Phases 1 and 2) and the latter (or "bottom") part of the system (Phases 3 and 4) were clearly different. Consequently the work was divided in two parallel efforts: (1) a top-part-of-the-system effort, essentially of an artificial intelligence nature, involved with the comprehension of natural language by machines, user requirements acquisition, model formation, problem solving and the development of a supporting high-level language/system called OWL, and (2) a bottom-part-of-the-system-effort addressing the problems of implementation and optimization of a program given an abstract relational specification (ultimately to be passed down from the top part of Protosystem I) of what it is to do. The bottom part of Protosystem I has been completely implemented in the MACLISP language and is operational on the M.I.T. Laboratory for Computer Science PDP-10 computers. Research and development on the top part, being considerably more ambitious and novel, is somewhat less mature. It is not expected to cross the threshold of practical applicability for another five years, and so will not be discussed further in this paper.

A structural diagram, indicating the major modules, flows of control and flows of data in the bottom part of Protosystem I is shown in Fig. 1. The following sections give an explanation of the working of this part of the automatic programming system.

E. THE PROTOTYPE I DATA PROCESSING SYSTEM MODEL AND THE SYSTEM SPECIFICATION LANGUAGE

Protosystem I handles a restricted but significant subset of all data processing applications: I/O intensive batch oriented systems. Such systems involve a sequence of runs or job steps that are to be performed at specified times. They are assumed to involve significant I/O activity due to repetitive processing of keyed records from large files of data. Systems such as inventory control, payroll, and employee or student record keeping systems are of this type.

A simple example of such a dps is a software system to perform the inventory and warehousing activities in the following case:

The A & T Supermarket chain consists of 500 stores served by a centrally located warehouse. There are 4000 items, supplied by the warehouse, that these stores can carry.

Every day the warehouse receives shipments from suppliers and updates its inventory level records accordingly.

It also receives orders from the stores for various quantities of items. If for a particular item there is sufficient stock to fill all of the orders for that item, the warehouse simply fills the orders as made; but if there is insufficient stock it ships partial orders proportional to a fraction of the total quantity ordered that is on hand.

Inventory records are adjusted to reflect the decrease in levels.

Finally, a daily check is made on the inventory levels of all items. If the level of an item is lower than 100, the warehouse orders 1000 more units of that item from the appropriate supplier.

In order for the bottom part of Protosystem I to implement such a data processing system application the basic aggregate data entities and their interrelationships must be determined. This determination can be made from the English task description by a consultant or by a sophisticated, natural language comprehending software system (e.g. the top-part of Protosystem I) that has embedded in it his knowledge and experience about business systems and data processing.

Consider the inventory updating activity of the second paragraph. There are three aggregate data entities involved: (1) the set of quantities received from suppliers, (2) the set of closing inventory levels for the previous day, and (3) the set of the updated inventory levels to be used for filling store orders. Such collections of similar data that are to be processed in a similar way are termed data sets. In the domain of Protosystem I a data set is assumed to consist of fixed format records (e.g. one for the level of each inventory item). Associated with each record is a data item (e.g. the level of an inventory item) and keys. The key values of a record uniquely distinguish it (e.g. the inventory data set can be keyed by item since there is only one level [record] per item) and so can be used to select it. Thus, a data set is essentially the same as a Codd relation and its keys are what Codd calls primary keys.

Let us call the three data sets described in the last paragraph SHIPMENTS-RECEIVED, FINAL-INVENTORY and BEGINNING-INVENTORY. The relationship between the BEGINNING-INVENTORY data set and the SHIPMENTS-RECEIVED and FINAL-INVENTORY data sets may be described as follows.

For every item:

the beginning inventory level of that item
(i.e., the value of the data item for the record
in BEGINNING-INVENTORY for that item)

is the closing inventory level of that item from
the previous day
(i.e., the value of the data item in the record
of FINAL-INVENTORY for the same item)

plus the quantity of that item received
(i.e., the value of the data item in the record
of SHIPMENTS-RECEIVED for the item in
question), if any.

This relationship is expressed more succinctly in SSL (the System Specification Language):

BEGINNING-INVENTORY IS FINAL-INVENTORY(1 DAY AGO) + SHIPMENTS-
RECEIVED

Implicit in this statement is that the addition operation is performed for each item and that if one of the operands is missing (e.g. if no chicken noodle soup was received today) it is treated as having a zero value. The repetitive application of an operation to the members of a data set or sets such as this is termed a computation. The order of applications of the operation to the records of its input data sets by a computation is assumed to be unimportant to the user; in fact, he may think of them as being performed in parallel. However, every computation does, in fact, process its inputs serially, according to a particular ordering (chosen by Protosystem I) on their keys. Computations typically match records from different data sets by their keys (as above) and operate on the matching records to produce a corresponding output record. A computation may also group the members of a data set by common keys and operate on each group to produce a single corresponding output. Returning to our example, note that item orders can come from different sources (stores), so that both the item and the source of an order are needed (as keys) to distinguish it. To form the total of all orders for each item, a computation must group the orders by item and sum over the order amounts in each group. In SSL this would be expressed as:

TOTAL-ORDERS FOR EACH ITEM IS THE SUM OF THE QUANTITY-
ORDERED-BY-STORE

Fig. 2 shows the structure of the A & T inventory and warehousing data processing system in terms of computations (boxed) and data sets (unboxed). The complete SSL description of A & T dps is given in Fig. 3. Note that in addition to the relational statements a list of data sets must be included to indicate the keys by which they are accessed.

F. THE TRANSLATOR AND THE DATA SET LANGUAGE

It is characteristic of the data processing systems which Protosystem I proposes to treat that the calculations themselves are easily dealt with and that it is the structuring and manipulation of the masses of data involved that occupies by far the greater part of the Stage 3 implementation activity. Additionally, the moving and storage of aggregate data entities must be determined before the operations on their members can be considered. Consequently, the development process at Stage 3 is **data set oriented**. Therefore, to facilitate the design process the SSL dps description is first analyzed from this point of view and re-expressed in a more appropriate medium, DSL (the Data Set Language). This reformulation is performed by the Translator module.

The determination of dps characteristics that can aid in the development of the dps design is made with the aid of the Structural Analyzer and included in the Translator's output description. This output is called the UDSL (Unconstrained Data Set Language) description, because most design details remain unbound (undecided) in it. As such it forms the skeleton of the dps description ultimately to be produced by Stage 3.

One useful piece of information determined by the Structural Analyzer is the set of driving data set candidates for each computation. A driving data set is an input data set that is guaranteed to have a data item for every tuple of key values for which the computation can produce an output. The computation, then, instead of having to loop over all possible combinations of values for the keys of the inputs, can be driven by the driving data set in that it only has to consider those key value combinations for which the driving data set contains records.

Another type of information the Structural Analyzer determines is directly related to our desire to specify data set organizations and orders and computation accessing methods and orders in such a way as to minimize the cost of operating the dps. Because a dps typically involves the repetitive application of simple calculations to large quantities of data we make the first-order approximation that the cost of operation is due entirely to data accessing (reading and writing). Our design, therefore, focuses on minimizing the total number of I/O events.

Accordingly, the Structural Analyzer also determines predicates that are the conditions under which a data item will be generated and under which a data item will be used by a computation. For example, a store will be shipped an item if (it is true that) that store ordered that item and there was sufficient inventory to fill the order; the order allocation step will use the inventory level for a particular item if some store ordered it. These predicates, together with basic information concerning the sizes of data sets in the dps, are used by the Question Answerer to determine the average and maximum sizes of files (proposed by the Optimizing Designer) and the average number of a file's records a computation will access.

G. THE DESIGN CRITERION AND THE JOB COST ESTIMATOR

The design criterion for Protosystem I is the minimization of the dollars and cents cost of running the final dps program on the target machine/operating system configuration. Because the dps's are assumed to be I/O intensive, as a first approximation, this can be equated with access minimization. An access in this sense is defined as the reading or writing of a single secondary storage block, which corresponds to a single operating system I/O event. In Protosystem I, for a particular data set a block consists of a fixed number of records.

With this approximation the relative costs of alternative dps design configurations can often be assessed without knowledge of the particular target configuration. But sometimes actual cost estimates, provided by the Job Cost Estimator, are necessary. This module must thus contain knowledge of the charging scheme and operating characteristics of the target configuration (in our case the OS/360 configuration). Optimization with respect to a different configuration and/or charging scheme would require the substitution of a new appropriately tailored module.

H. THE QUESTION ANSWERER

The function of the Question Answerer is to supply answers to questions from the Optimizing Designer about the average sizes (in records) of abstract aggregate data entities. Two examples of such data aggregates are a file and the collection of records in a file that are accessed by a particular computation. Each "question" sent to the Question Answerer is in the form of a predicate describing the conditions under which a record will be in the data aggregate in question. For example, if there are records in FINAL INVENTORY, QUANTITY RECEIVED and BEGINNING INVENTORY for only those items have non-zero quantities, the predicate

there is a record in FINAL INVENTORY (for a given item)

or

there is a record in QUANTITY RECEIVED (for a given item)

describes an event equivalent to "there is a record in BEGINNING INVENTORY" for a given item. The Question Answerer makes use of the simplifying assumption that all records in an abstract aggregate data entity are equally likely to be present. Thus, if the maximum size of a data aggregate is well defined (e.g. BEGINNING INVENTORY can be no larger than the set of all items carried by the warehouse), its average size can be calculated by multiplying the probability that the event that the typical record in it will be present by its maximum size. If there is no meaningful maximum size (as, for example, with a data set that is the collection of all outstanding purchase orders) the average size of the data aggregate must be determined directly.

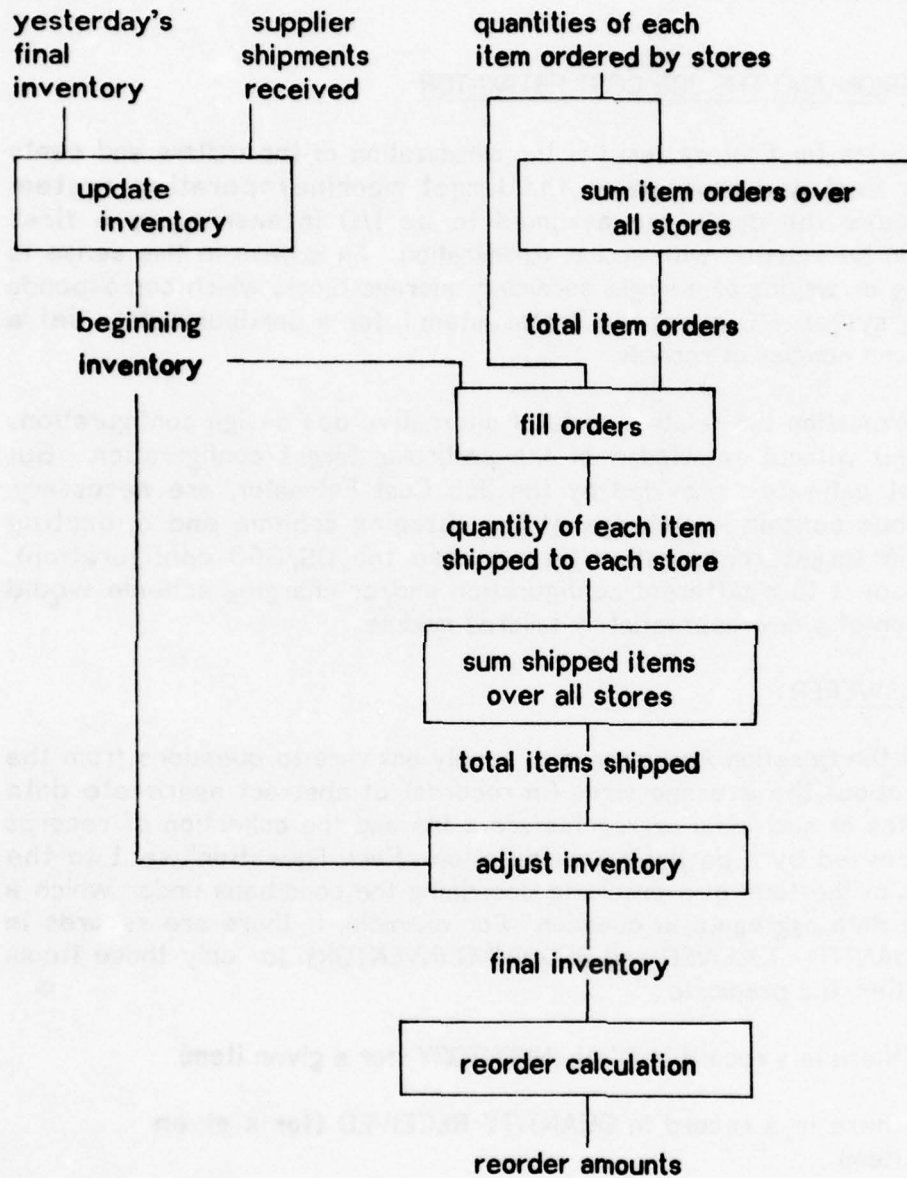


Figure 2. A & T Inventory and Warehousing System

The Question Answerer maintains a data base of all of the event probability and size information given by the user. When asked a question it attempts to find the associated size or probability directly. Failing this, it will try to calculate the probability of the event in question happening from those of its subevents and its knowledge of event independence and correlation within the dps. If the information on hand is insufficient to answer the question, the Question Answerer obtains enough additional information from the user (through a flexible line of questioning) to do so. The new information thus gained is stored in the data base for future reference.

I. THE OPTIMIZING DESIGNER

The Optimizing Designer is the heart of Stage 3; all of the other modules in this stage exist merely to serve it. When the translation from SSL to UDSL has been completed, control passes to the Optimizing Designer. This module is responsible for constructing job steps to implement computations and files to implement data sets. In particular its job is to:

- a. design each keyed file--in particular its
 1. contents (information contained)
 2. OS/360 organization (consecutive, index sequential, or regional(2))
 3. storage device
 4. associated sort ordering (by key values)
 5. blocking factor (number of records per block)
- b. design each job step of the dps--namely
 1. which computations it includes
 2. its accessing method (sequential, random, core table)
 3. its driving data set(s)
 4. the order (by key values) in which it processes the records of its input data sets
- c. determine whether sorts are necessary and where they should be performed
- d. determine the sequence of the job steps.

CALCULATIONS EVERY DAY

BEGINNING-INVENTORY IS FINAL-INVENTORY(1 DAY AGO) + SHIPMENTS-RECEIVED

TOTAL-ITEM-ORDERS IS SUMS OF GROUPS QUANTITY-ORDERED-BY-STORE BY ITEM

QUANTITY-SHIPPED-TO-STORE IS

QUANTITY-ORDERED-BY-STORE IF BEGINNING INVENTORY IS GREATER
THAN TOTAL-ITEM-ORDERS

QUANTITY-ORDERED-BY-STORE
* (BEGINNING-INVENTORY / TOTAL-ITEM-ORDERS)

IF BEGINNING INVENTORY IS

NOT

GREATER THAN TOTAL-ITEM-

ORDERS

TOTAL-SHIPPED IS SUMS OF GROUPS OF QUANTITY-SHIPPED-TO-STORE BY ITEM

FINAL-INVENTORY IS BEGINNING-INVENTORY - TOTAL-SHIPPED

REORDER-AMOUNTS IS 1000 IF FINAL-INVENTORY IS LESS THAN 100

DATA SET TABLE

(SHIPMENTS-RECEIVED	DAY ITEM)
(BEGINNING-INVENTORY	DAY ITEM)
(FINAL-INVENTORY	DAY ITEM)
(ORDERS-TO-SUPPLIERS	DAY ITEM)
(QUANTITY ORDERED-BY-STORE	DAY ITEM)
(QUANTITY-SHIPPED-TO-STORE	DAY ITEM STORE)
(TOTAL-ITEM-ORDERS	DAY ITEM)
(TOTAL-SHIPPED	DAY ITEM)
(REORDER-AMOUNTS	DAY ITEM)

Figure 3. SSL Relational Description for the A & T Data Processing System

The Optimizing Designer performs dynamic analysis (analysis of the operating behavior) on the dps to propose and evaluate alternative design configurations. Occasionally, static analysis (analysis of system structure and interrelationships) of such tentative configurations is also necessary, and this is obtained through calls to the Structural Analyzer. When additional information is needed to make evaluations and decisions the Question Answerer and the Job Cost Estimator are called.

All design decisions are made in an effort to minimize the total number of accesses that must be performed in the execution of the dps. There are three major techniques that the Optimizing Designer uses toward this end:

1. Designing Files and Job Steps in Such a Way as to Take Advantage of Blocking.

Accesses can be reduced if files are given blocking factors greater than one and if processing and file organizations are designed in such a way that the records of a each block can be used consecutively.

2. Aggregating Data Sets.

If two or more data sets that are accessed by the same computation are combined into one file (see Fig. 4) and processing is arranged so that a single record of the aggregate can be accessed where more than one record from each of the otherwise unaggregated files would have been accessed, accesses can be saved.

3. Aggregating Computations.

When two or more computations access the same data set and the orders in which they process the records of that data set are the same, it may be advantageous to combine them into a single job step. Then each record of the shared data set can be accessed once for all, rather than once for each computation (see Fig. 5).

These access minimizations techniques require that the key order of processing agree in a special way with the organization of the data being processed. This is where the fundamental difficulty in optimization lies. A data set's organization and the accessing method of a computation using it cannot be determined independently of each other or of other data set organizations and computation accessing methods. The organization of a data set limits the ways in which it can be practically accessed by a computation, and, conversely, the accessing method of a computation restricts the practicable organizations of a data set that it accesses. Furthermore, a data set is typically accessed by more than one computation with possibly conflicting preferences for its organization; and a computation accesses more than one data set with conflicting preferences for accessing methods. Finally, data set organization constraints tend to propagate through computations, because it is most efficient for a computation to write its outputs in the same key order in which it reads its inputs (since that is the order in which the output

$(N_{11} \quad 3)$	$(N_{21} \quad 2)$
$(N_{12} \quad 2)$	$(N_{22} \quad 4)$
MALE-EMPLOYEES(DEPT)	FEMALE-EMPLOYEES(DEPT)
$(N_{11} \quad --- \quad 3)$	
$(N_{12} \quad N_{21} \quad 2)$	
$(--- \quad N_{22} \quad 4)$	
MALE-&-FEMALE-EMPLOYEES(DEPT)	

Figure 4. Data Set Aggregation

(The N_{ij} are values of data items and the numbers are values of the key DEPT)

records will be generated). So, optimization of the type we are considering is necessarily a problem in global compromise.

The straightforward solution of evaluating the cost of every possible combination of assignments of sort order, device, organization, and access method for data sets and computations in every possible aggregation configuration to determine the least expensive is ruled out by the sheer combinatorics involved. Even with mathematical and special purpose tricks it would be impossibly slow.

To make optimization tractable a heuristic approach must be taken. First different kinds of decisions (e.g. choice of driving data sets, which objects to aggregate) must be decoupled wherever possible. Further decoupling must be judiciously introduced where it is not strictly possible, for the sake of additional simplicity. Such forced decoupling does not mean, though, that decisions that are in fact coupled are treated as if they were independent. The decoupled decisions are still made with a certain awareness of their effects on other decisions. Finally, as a first order approximation, the optimizer does what is reasonable locally, and then adjusts somewhat for global realities. While we make no claim that this approach will lead to the true optimum, it does produce good and usually near-optimal solutions for real and honest problems.

J. CODE GENERATION

Stage 4 of Protosystem I consists of the PL/I and JCL Generator modules. The PL/I Generator takes the fully specified output of Stage 3 (the CDSL or Constrained Data Set Language description) as input and produces PL/I code for each job step. This involves the determination and arrangement of PL/I I/O specifics, the construction of the data processing loops, and the programming of the necessary calculations. The JCL Generator then writes IBM OS/360 JCL and ASP instructions for the I/O, administration and scheduling of the compilation and execution of the dps job and job steps.

K. CONCLUSION

A model of the data processing system implementation process has been presented and a blue-print, based on that model, for automating the entire process has been developed. Protosystem I is a project to exhibit the feasibility of these ideas. Already, two of the four heretofore manual phases of the software writing process have been automated and are capable of producing acceptable implementations. The automation of the remaining two phases should easily fall within the realm of presently developing technologies within the next decade.

Directions for further investigation include:

1. Expansion of the design repertoire--additional data structures (e.g. hierarchical files, inverted files), the use of Early's iteration inversion ideas, etc.

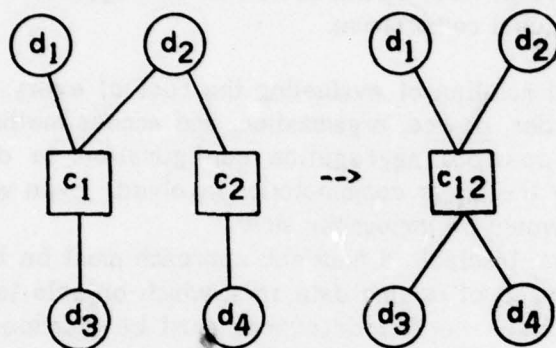


Figure 5a. Horizontal Aggregation of Computations

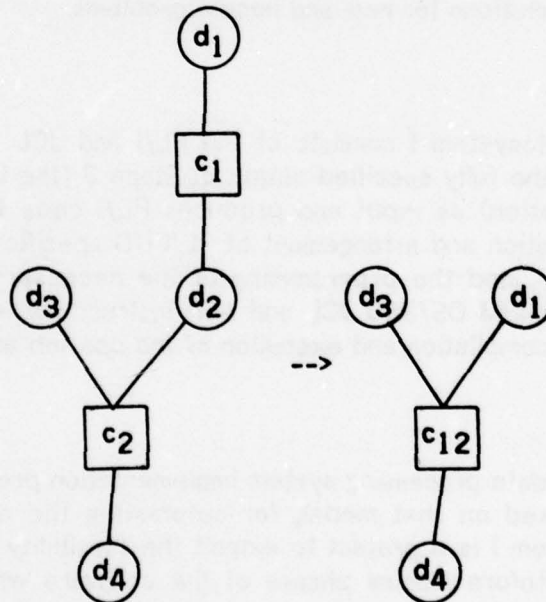


Figure 5b. Vertical Aggregation of Computations

2. Enlargement of the class of dps's handled (e.g. admitting other types of computations, on-line systems).
3. Development of peripheral automatic technologies--for example, automation of incremental changes to dps's with minimal perturbation/maximal efficiency.
4. Automatic development of dps back-up and restart capabilities.

BIBLIOGRAPHY

1. Balzer, Robert. Automatic Programming. University of Southern California, Information Sciences Institute, Technical Memo. Marina del Rey, Ca., 1973.
2. Codd, E. F. "A Relational Model of Data for Large Shared Data Banks. Communications of the ACM, Vol. 13 No. 6 (June 1970), 377-387.
3. Early, J. Relational Level Data Structures For Programming Languages. University of California, Berkeley, Ca., Computer Science Department, 1973.
4. Hammer, Michael; Howe, W.; and Wladawsky, I. "An Overview of a Business Definition System." ACM SIGPLAN Notices, Vol. 9 No. 4 (April 1974).
5. Hawkinson, Lowell. "The Representation of Concepts In OWL." Fourth International Joint Conference on Artificial Intelligence. Tbilisi, U.S.S.R. Sept. 1975.
6. Nunamaker, J. F. Jr.; Nylin, W. C. Jr.; and Konsynski, B. Jr. "Processing Systems Optimization through Automatic Design and Reorganization of Program Modules." Information Systems. edited by Tou. New York: Plenum, 1974.
7. Ruth, Gregory. "The New Question Answerer." M.I.T., Laboratory For Computer Science, Automatic Programming Group, Internal Memo 21. Cambridge, Ma. 1975.
8. Sussman, Gerald. A Computational Model of Skill Acquisition. M.I.T., Artificial Intelligence Laboratory, A.I. TR-297. Cambridge, Ma., August 1973.

MATHLAB GROUP

67

MATHLAB GROUP

MATHLAB

Academic Staff

J. Moses, Group Leader
V. Pless

P. S.-H. Wang

Research Staff

A. P. Doohovskoy
J. P. Golden
J. P. Jarvis

D. A. Moon
J. L. White

Graduate Students

I. D. Avgoustis
M. R. Genesereth
J. L. Kulp
G. L. Steele

B. M. Trager
R. D. Weiss
R. E. Zippel

Undergraduate Students

D. R. Barton
D. J. Littleboy

S. M. Macrakis
M. G. Mulligan

Support Staff

V. E. Lewis

Guests

U. Pape

T. Minamikawa

MATHLABA. INTRODUCTION

The past year saw the long-awaited beginning of a new phase of operation for the group -- MACSYMA Consortium. The Consortium purchased a KL-10 computer which is approximately five times as fast as our older KA-10 computer. Research on new algorithms was undertaken in many areas such as manipulation of algebraic functions, integration of special functions, inversion of matrices, and finding invariants for a set of polynomial ideals. As a result of the present and past research of the group, approximately 40% of the algebraic manipulation papers in the forthcoming 1976 Symposium on Symbolic Algebraic Computation are by present and former members of the group, and by our present set of users.

B. THE MACSYMA CONSORTIUM MACHINE

A DEC KL-10 system was purchased for the Consortium, largely from ARPA funds, and was delivered in July 1975. Guy L. Steele and David A. Moon rewrote the microcode for the machine so that it simulated the pager on the Mathlab KA-10 and our ITS operating system was in operation in November.

Our early experiments on the KL-10 indicated that the performance of the machine was approximately that which we anticipated. The factor of five speed improvement (circa 1.5 mips) has been very welcome indeed. We have been able to handle 15 simultaneous MACSYMA users with hardly a decrease in response. Partly this is due to the high rate of sharing in each MACSYMA (up to 150K words) and partly to the very nature of MACSYMA usage, since users spend a fair amount of time planning their next step.

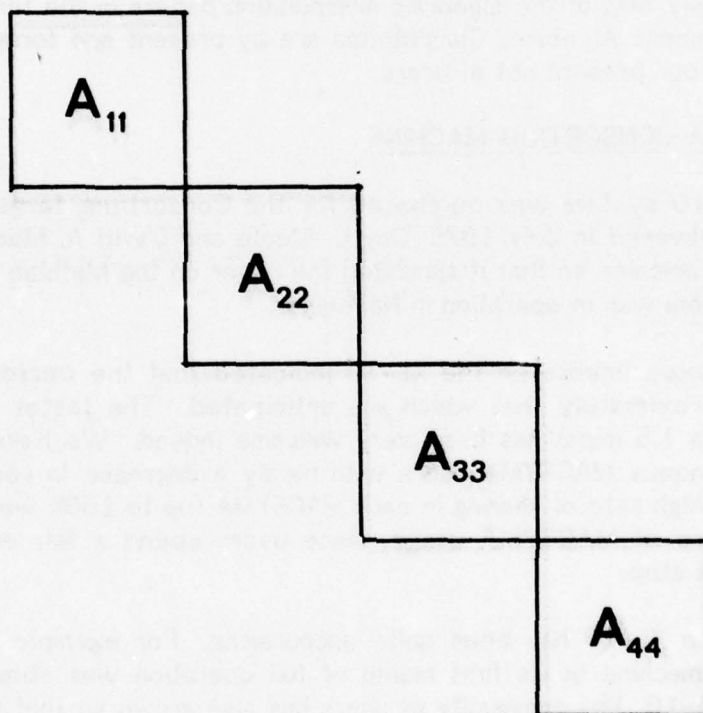
Usage of the KL-10 has been quite encouraging. For example, the total CPU utilization of the machine in its first month of full operation was about 2 times the capacity of the KA-10. The community of users has also grown so that there are over 250 different MACSYMA users each month.

C. MATRIX INVERSION ALGORITHMS

For the past two years, we have been quite interested in finding approaches for calculating determinants of matrices and their inverses more efficiently. The last Progress Report gives some of our approaches to the determinant problem. This year Professor Paul Wang and a Japanese visitor, Tadotshi Minamikawa, investigated the matrix inversion algorithm. Inverses of many sparse matrices tend to be dense ones. It would be very useful to know whether the inverse can be sparse, especially if one can predict the zero entries without too much computation. We shall consider an element in the inverse to be predictably zero, if the matrix whose determinant defines

its value is the zero matrix. We thus ignore situations in which elements in the inverse become zero because of cancellations which may occur in the determinant when one considers the specific value of the entries of the matrix. Minamikawa and Wang show that unless a nonsingular matrix has a particular "block" structure, indicated below, then no element in its inverse is predictably zero. Further any nonsingular matrix possessing a block structure can be shown to have that structure using row permutations.

For example, the matrix below has a "block structure".



The matrices along the diagonal are nonsingular square matrices of possibly different sizes. Below them all the entries are zero. The entries above them may have any value. The inverse of such a matrix may be computed easily from the inverse of the matrices along the diagonal.

The submatrix A_{11} is said to be unreachable in the following sense: consider a directed graph whose nodes are $1, 2, \dots, n$ representing the rows and columns of the

matrix A . If element a_{ij} is nonzero, there is an arc from node i to node j . A_{11} is unreachable since there is no arc into it from nodes outside of it.

Recall that the inverse of a matrix can be obtained by computing

$$\frac{\det(A_{ij}^*)}{\det(A)}$$

where A_{ij}^* is the $(n-1) \times (n-1)$ matrix with row i and column j deleted. If the matrix A has a block structure, then it must have an unreachable submatrix. Assume it did not have such a block structure. Then there is a path in each A_{ij}^* traversing all the rows. Therefore the $\det(A_{ij}^*)$ is not identically zero for all i, j .

D. POLYNOMIAL DECOMPOSITION

The group has had great success in the past few years in devising efficient algorithms for factoring polynomials. We have not known any effective way of obtaining a related factorization, that is, given a polynomial $f(x)$, find $g(x)$, $h(x)$, if they exist as polynomials, such that $f(x) = g(h(x))$. Barton and Zippel have recently found such an algorithm which is surprisingly simple. The idea is to factor, in the usual manner, the bivariate polynomial $f(x) - f(y)$. We look for factors of the form $h(x) - h(y)$. These will be candidates for the h 's; the g 's can then be found with little effort. Clearly $x - y$ is always a factor. In fact, h 's which are linear are not too interesting--they introduce an equivalent decomposition. For example, the algorithm finds that $f(x) = x^6 + x^4 + x^3 + 9x^2 + 3x - 5$, which is irreducible over the integers, has a decomposition $g(h(x))$, where $h(x) = x^3 + 3x$, $g(x) = x^2 + x - 5$.

Our application for polynomial decomposition is in finding roots of polynomials in terms of radicals. For the polynomial $f(x)$ above, which is irreducible and of degree 6, there is no known general procedure for its roots in terms of radicals. But viewed as $g(h(x))$, we can solve for the roots of h , and iterate on the roots of g .

E. DEFINITE INTEGRATION OF SPECIAL FUNCTIONS

The group has had much success in the past with decision procedures for indefinite integration of the elementary functions of the calculus. Unfortunately, definite integration problems, (e.g. integrals from 0 to infinity) do not possess general decision procedures. Furthermore practical interest in definite integration centers around special functions (e.g. Bessel functions), for which no general indefinite integration algorithms exist. The approaches taken by applied mathematicians in the past 150 years for solving definite integration problems have been codified in a series of volumes called the Bateman Manuscript Project. Ioannis Avgoustis has been distilling the information in these volumes particularly those related to Laplace Transforms of special functions, in a relatively few rules and procedures. The general approach used

by him is to translate the problem to one involving hypergeometric functions, solve the integration problem in terms of generalized hypergeometric functions, and finally solve the difficult problem of representing the result in terms of classical special functions.

Publications

1. Lewis, V. Ellen. An Introduction to ITS for MACSYMA Users. M.I.T. Laboratory for Computer Science, Mathlab Group Memo. Cambridge, Ma., June 1975.
2. Moses, Joel. "Current Capabilities of the MACSYMA System." Proceedings of the ACM Annual Conference. ACM, Minneapolis, Minn., October 1975.
3. Pless, Vera. Encryption Schemes for Computer Confidentiality. M.I.T. Project MAC, TM-63. Cambridge, Ma., May 1975.
4. Pless, Vera. "Symmetry Codes and their Invariant Subcodes." Journal of Combinatorial Theory, (A). Volume 18 (1975).
5. Pless, Vera. "CAMAC--Combinatorial and Algebraic Machine Aided Computation." Proceedings of the Sixth Southeastern Conference on Combinatorics, Graph Theory, and Computing. Florida Atlantic University, Boca Raton, Fl., February 1975.
6. Pless, Vera and Sloane, Neil J. A. "Classification and Enumeration of Self-Dual Codes." Journal of Combinatorial Theory, (A). Volume 18 (1975).
7. Weiss, Randall B. How to Use the CAMAC Group Manipulation System. M.I.T. Project MAC, TM-60. Cambridge, Ma., March 1975.
8. Zippel, Richard E. "Solution to Problem 8." SIGSAM Bulletin. (March 1975).
9. Zippel, Richard E. "Power Series Expansions in MACSYMA." Proceedings of Conference on Mathematical Software II. ACM, Purdue University, W. Lafayette, In., May 1974.

Theses Completed

1. Dadashzadeh, Mohammed. "A Program for Drilling Students in Ordinary Differential Equations Problems." unpublished S.B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1975.
2. Weiss, Randall B. "Finding Isomorph Classes for Combinatorial Structures." unpublished S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1975.

Theses in Progress

1. Genesereth, Michael R. "An Advisor for MACSYMA". Ph.D. Thesis, Harvard University, expected date of completion, May 1978.
2. Trager, Barry M. "Integration of Algebraic Functions", S.M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, May 1976.

Talks

1. Genesereth, Michael R. "An Advisor for MACSYMA." Computer Science Colloquium, Harvard University, Cambridge, Ma., May 1975.
2. Moses, Joel. "Algebraic Manipulation and Artificial Intelligence." Fourteen lectures at Electrical Technical Laboratory, Computer Science Department, Tokyo University, Tokyo, Japan; Electrical Engineering Department, Kyoto University, Kyoto, Japan, January, 1975.
3. Moses, Joel. "Capabilities of the MACSYMA System." Computer Science Colloquium, U.S. Naval Research and Development Center, Bethesda, Md., January 1975; ACM Southeastern Regional Conference, Atlanta, Ga., March 1975.
4. Moses, Joel. "The Personal Computer." Electrical Engineering Seminar on Computers and Communications, M.I.T., Cambridge, Ma., November 1975.
5. Moses, Joel. "Symbolic Integration." Two lectures. Workshop on Quadrature Algorithms, Los Alamos, N.M., May 1975.
6. Pless, Vera. "Mathematical Foundations of Interconnected J-K Flip-Flops." Annual winter meeting of the American Mathematical Society, Washington, D.C., January 1975.
7. Pless, Vera. "CAMAC." Sixth Southeastern Symposium on Combinatorics, Graph Theory and Computing, Florida Atlantic University, Boca Raton, Fl., February 1975.
8. Pless, Vera. "New Invariant Subcodes of the Symmetry Codes." M.I.T. Combinatorics Seminar, Cambridge, Ma., March 1975.
9. Pless, Vera. "Error-Correcting Codes: Practical Origins and Mathematical Implications." Miniconference on Combinatorial Designs, University of Pittsburgh, Pittsburgh, Pa., March 1975.

10. Pless, Vera. "Introduction to Error-Correcting Codes." Wayne State University, Detroit, Mi., May 1975.
11. Pless, Vera. "Error-Correcting Codes: Practical Origins and Mathematical Implications." Mathematics Colloquium, University of Illinois, Circle Campus, Chicago, Il., May 1975.

PROGRAMMING METHODOLOGY GROUP 77 PROGRAMMING METHODOLOGY GROUP

PROGRAMMING METHODOLOGY

Academic Staff

B. H. Liskov, Group Leader

Graduate Students

R. R. Atkinson
V. A. Berzins
T. Bloom
D. Kapur
M. S. Laventhal

J. E. Moss
R. N. Principato
J. C. Schaffert
L. A. Snyder
M. K. Srivas
S. N. Zilles

Undergraduate Students

G. L. Fulton
D. P. Gorgen
E. J. McCabe

R. W. Scheifler
K. Virgile

Support Staff

M. Nieuwkerk

A. L. Rubin

PRECEDING PAGE BLANK

PROGRAMMING METHODOLOGYA. INTRODUCTION

In the past year, work conducted in the Programming Methodology Group has emphasized the development of new programming methodologies using data abstractions. This work involves the study of tools and techniques to enhance the effectiveness of programmers in producing quality software--software that is reliable, has comprehensible structure, and is relatively easy to modify and maintain. Two major directions are being followed. A programming language/system, CLU, is under development. CLU enhances program quality by permitting direct expression of the kinds of program structures arising from promising design methodologies. In addition, a study of specification techniques well-suited to the program structures of CLU is under way; using these techniques, the programmer will be able to express and investigate properties of his program design in advance of actual implementation, and to prove the correctness of his implementation once it exists.

B. THE CLU LANGUAGE/SYSTEM

The motivation behind the development of the CLU programming language and system is discussed in [3,4]. Briefly, CLU is intended to simplify the design and implementation of quality software by providing linguistic constructs that allow the kinds of modules identified during design to be written naturally as CLU programs. The most important such construct, and the one that is original in CLU, is the cluster. The cluster permits a program module to be written that implements a data abstraction, or abstract data type, consisting of both a set of objects or values belonging to the type, and a set of operations that completely determine the behavior of the type's objects. Data abstractions are a particularly valuable sort of program module: they occur widely, since the manipulation of data is a primary concern of programming, and although much sharing of information and resources takes place within the data abstraction (particularly important is information about how objects of the type are represented in storage), this sharing is limited to the implementation of the data abstraction, and is not visible to the abstraction's users. The advantages of such an organization have been discussed in [1,2,5]; for example, the interface of a data abstraction module is very simple, and the hiding of information within the module means that the implementation of the abstraction can be changed without requiring recoding of the programs using the abstraction. One such change that might be made is to choose an alternative representation for the data abstraction's objects.

CLU differs from other languages in its emphasis on constraints, enforced by the language and its compiler, that guide the programmer's search for a good design by removing his or her freedom to violate certain precepts of good programming practice. A common constraint, found in almost all higher level languages, protects the local variables of a procedure from manipulation outside of the procedure's code. CLU extends this idea of constraints to permit a group of procedures to share a local environment; this is done through the cluster, which limits information about a type's

implementation to the operations belonging to the type. Conventional languages (e.g. FORTRAN, PL/I) provide no mechanism like the cluster. Even advanced extensible languages (e.g. ELI[6]), which provide data type extensions and even permit some operations to be defined along with the type, still do not constrain access to the type to just the operations, so the advantages mentioned above can be obtained by the programmer only by extra-language means. The issue here is not whether well-structured programs can be written, since they can be written even in assembly language. However, in languages other than CLU, such programs can be written only in spite of the language. Our goal is to simplify the writing of programs by having the language provide guidance about what constitutes good programming practice.

CLU is a language/system. A CLU program consists of a number of modules; each module implements an abstraction identified as useful during program design. The CLU system includes a description unit for each module, containing all in-computer information about the module. The description unit is created as soon as the module interface is known (before the module is implemented); formal specifications would also be entered at this stage. Modules are compiled separately; the CLU compiler makes use of the interface information to check that modules refer to other modules correctly. The CLU system is also used to control the loading and execution of programs.

Our major activity during the past year has been the implementation of a first version of CLU. The implementation is divided into three pieces: the CLU compiler, inter-module type-checking, and the CLU system. The CLU compiler was implemented first and has been running for several months; it translates CLU modules into a LISP-like language called MDL [7]. The type-checker has been implemented and debugged, and is awaiting integration with the CLU system (which contains the information about the type requirements of modules, and also establishes the meaning of types). The CLU system is currently being designed and implemented. A very interesting problem arising in the design of such a system concerns how to cope with multiple implementations of a data type. We are working on providing multiple implementations of a type in a very flexible way: different users can select different implementations, and even within the same program, different implementations can be used for different data objects of the same type.

The implementation of CLU was undertaken for two reasons: to establish the soundness of our design, and to permit us to gain experience in using CLU. No problems with the design of CLU were uncovered during the implementation; some minor modifications have been made to the language to make CLU programs easier to write. We have used CLU both for the CLU implementation, and to write many smaller programs. The results have been encouraging; programmer productivity is high, and the resulting programs have a good structure and are easy for others to understand. We have discovered that data abstractions are indeed very valuable for structuring programs, and that we design programs by identifying data abstractions, and then specifying the properties of their operations, in advance of any implementation. The transition from design to implementation is particularly simple, since each design unit

becomes a CLU program module.

The language being implemented is only an initial version of CLU, and we have continued to work on the design of CLU itself. One accomplishment of the past year has been the design of the structured exception handling mechanism described below.

C. STRUCTURED ERROR HANDLING

In designing the exception handling mechanism of CLU, our primary concern was the support of "robust" or "fault tolerant" programs, i.e., programs that are prepared to cope with the presence of errors by attempting various error recovery techniques. Note that it is the programs themselves that must recover from errors. We do not assume that a person helps in the error recovery (although this will sometimes happen), and therefore the mechanism need not facilitate person-computer interaction. In particular, the mechanism is not intended to support interactive debugging.

Successful handling of errors involves two separate activities. First the errors must be detected. After an error has been detected, it may then be possible to recover from the error. If it were always possible to recover from an error in the same local context in which the error was detected, no special error handling mechanism would be needed. However, often recovery must occur at a very different point in the program from where the error was detected. Thus the purpose of the mechanism is to permit information about errors to be communicated from one part of the program to another. Note that we are not concerned here with how error detection and recovery are accomplished (through redundancy) except to recognize that paths permitting communication of information about errors are required.

The use of the word "error" in the above discussion is somewhat misleading because what may appear as an error to one part of a program may be considered as reasonable behavior in another part. For example, an attempt to read from an empty file raises an "end-of-file" error; to the user of the read command, this merely means that all data has been read. Therefore, in the remainder of this section, we will use the more neutral term "exception" to refer to the occurrences of interest.

Our study of exception handling has led to the following analysis of how such a mechanism should behave:

1. Information about exceptions always results from a procedure invocation and flows from the called procedure to its caller. Whenever a procedure is invoked, it is invoked to perform a certain action or cause a certain effect. If the procedure is unable to do this, then it must notify its caller that something exceptional has occurred.

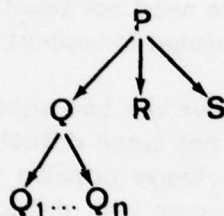
It is important to recognize that information about an exception results from an invocation even if the invocation does not actually occur. For example, the integer divide-check exception may result from the invocation of the integer

division operation. In CLU, as in most languages, division may be written as part of an expression:

$$y/x$$

and the code to do the division occurs in-line in the procedure containing this expression. Nevertheless, conceptually the exception is detected in a lower level procedure invocation, and the procedure containing the expression is notified of the exception.

2. The precepts of structured programming require that only the procedure performing an invocation can handle exceptions arising from that invocation, as the following discussion shows. CLU programs have a hierarchical structure. Consider, for example, the following graph:



Each node in this graph represents a CLU module, which implements an abstraction: P is programmed in terms of abstractions Q, R, and S; Q makes use of abstractions Q_1, \dots, Q_n ; and so on. The graph is a static view of the program structure; each module makes use of the modules connected to it by the outward pointing arcs. Dynamically, a module makes use of another module by invoking one of its procedures; invocations can only occur in the direction specified by the arc between two modules. Information about exceptions flows in the opposite direction; a procedure notifies procedures higher in the call chain of the existence of an exception.

Some rules about structured exception handling can be derived from considering the relationships between modules in the graph. An important precept of structured programming is that a module knows only about the abstractions it uses; it knows nothing about the abstractions used in implementing those abstractions. Thus P knows nothing about Q_1, \dots, Q_n . But if a module knows nothing about the implementation of a module it uses, it cannot possibly respond intelligently to exceptions detected by procedures called in the course of that implementation. The only module that can respond to the exceptions detected by a procedure is that procedure's caller e.g., only Q can respond to the errors detected by Q_1, \dots, Q_n ; P cannot.

3. A principle of modularity is that a module should be programmed to know nothing about the module using it, so that it can be used in many different places. Thus, a procedure knows nothing about its caller, and should not report an exception

by a mechanism that assumes something about the caller's environment. In particular, a jump to a non-local label (as in PL/I) is not a satisfactory mechanism, nor should information about the exception be communicated using non-local variables. Instead the mechanism should permit a procedure to communicate information about an exception to its caller without making any assumptions about who its caller is. Note that this is very similar to the way that procedures return control to their callers under normal conditions.

The use of a non-local goto as an exception mechanism is unsatisfactory for another reason. Even if a program is unable to recover from an error, it should restore its non-transient data to a consistent state; this is necessary to prevent an error from causing many other, unrelated errors later on. The exception mechanism must ensure that active programs, whose data may be inconsistent, will have the opportunity to respond to exceptions before becoming inactive. Thus, a mechanism that automatically terminates procedure activations is unsatisfactory.

4. Finally we come to the question of what actions the caller of a procedure can perform when notified about the occurrence of an exception. The simplest view, which is the one we take, is that one of two actions can occur: if the procedure is unable to recover from the exception, it may notify its caller that an exception (different from the one detected) has occurred; or, if the procedure is able to recover, it may continue its normal flow.

What is explicitly forbidden here is the ability for the calling procedure to resume processing in the procedure that detected the exception. Although we recognize that the ability to resume is sometimes convenient, we believe resuming is not necessary (provided the mechanism is sufficiently general, as ours is) and that resuming necessitates a much more complicated model of computation, in which the exception-reporting procedures act like coroutines.

The result of our analysis of exception handling is the following simple model of how the exception handling mechanism should behave: each procedure can terminate execution in one of several states; one of these states is the "normal" state, while the others represent exceptional conditions. Each state is given a symbolic name (the normal state is implicitly named "normal"). Finally, in each state, values may be returned to the calling procedure; these values can differ in type and number from one state to another. Allowing parameters to be returned eliminates the need for global variables to hold error information.

We believe the above model strengthens the abstraction power of the language. Each procedure is expected to be defined over all possible values of its input parameters and all possible actions of the procedures it calls. However, it is not expected to behave in the same way in all cases. Instead, it may respond appropriately in each case.

We consider that an abstraction is not meaningful unless all the exceptions that can arise from its use are identified. Thus, the specification of integers must indicate that divide-check can occur, and (in most languages) that integer-overflow can also occur; without this information, the user of integers will not fully understand their behavior. Similarly, stacks have associated overflow and underflow exceptions.

The requirement that an abstraction identify all exceptions applies whether the abstraction is language or user defined. For all the primitive types in CLU, exceptions have been identified and made part of the abstraction. The inventor of new abstractions (e.g. stacks) should do the same. As an added benefit, the process of identifying exceptions can be quite valuable during program design: an abstraction with many exceptions and special cases can probably be improved by redesign. CLU arrays reflect this concern with limiting exceptions; CLU arrays have an "index out of bounds" exception, but not the "undefined element" exception that arises in array abstractions in which space for array elements can exist in advance of values to store in the elements.

We have a partial design of how the exception handling mechanism is to be incorporated in the CLU language. The mechanism is completely defined as far as reporting exceptions is concerned.

1. The mechanism for reporting exceptions is the signal, which is a special type of return. Since the signal is a return, the activation of the signalling module will disappear; therefore the procedure must ensure that all its non-transient data objects are in consistent states before signalling. The best method of ensuring this is to detect exceptions before any objects are modified, but this is not always possible.

A signal always specifies a particular exception name; thus a procedure may have several exceptions associated with it. In addition, some values may also be returned by a signal. For example,

signal foo(x)

terminates execution of the procedure containing the signal statement with an indication that the "foo" exception has occurred; the current value of x is returned as a result.

2. All the exceptions to be reported by a procedure must be specified as part of the header of that procedure. For example,

pop = oper(s:stack)returns(int)
 signals(underflow)

This information is also included in the CLU system library as part of the description unit of the abstraction that the procedure implements. If the

exception returns values, the types of these values must always be specified, and these types must agree with the types of values actually being returned. For example, the operation to read the *n*th character of a string has interface description

cn = oper(s:string, n:int)returns(char)
signals(bounds(int))

The bounds exception returns the value of the out-of-bounds integer.

As was mentioned earlier, the calling procedure is required to respond to, or "catch," all exceptions arising from procedures it invokes. The hardest part of designing an exception handling mechanism, once the basic principles are worked out, is to provide good human engineering for catching exceptions. Flexibility in placement of the exception handlers is essential; otherwise the readability of programs will be compromised. Exceptions can arise from the evaluation of every expression, but requiring handlers inside of expressions would make programs unreadable. At present, we are investigating the semantic and syntactic issues that arise from the requirement of flexible placement of exception handlers within the calling procedure.

D. SPECIFICATION TECHNIQUES FOR DATA ABSTRACTIONS

One of the properties of data abstractions mentioned earlier is that their interface is particularly simple, since so much information is hidden within the module. Therefore, we can hope that data abstractions will have simple specifications, since it is precisely the interface that a specification must describe. A study of specification techniques for data abstractions was undertaken this year by B. Liskov and S. Zilles and is reported in [8]. Included here is a brief summary of this work.

A formal specification for a functional abstraction describes the effect of a single operation; a convenient way to do this is by an input/output specification. A specification for a data abstraction, which contains many operations, must describe the effects of all the operations, and also the behavior of the objects belonging to the type. New techniques are being defined for specifying the behavior of data abstractions. Using these techniques, the entire data abstraction is specified as a unit, with the advantage that a more minimal specification results, describing just the externally observable behavior.

The information contained in a specification of a data abstraction can be divided into a semantic part and a syntactic part. Information about the actual meaning or behavior of the data abstraction is described in the semantic part; the description is expressed using a vocabulary of terms or symbols defined by the syntactic part.

The first symbols that must be defined by the syntactic part of a specification identify the abstraction being defined and its domain or class of defined objects, and, in this case, it is conventional to use the same symbol to denote both the abstraction and

its class of objects. Thus, the objects belonging to the data abstraction, stack, are referred to as stacks.

The remaining symbols introduced by the syntactic part name the operations of the abstraction, and define their functionality--the domains of their input and output values. An example describing the functionality of the operations of the data abstraction, stack, is shown below.

```
CREATE: -> STACK  
PUSH: STACK X INTEGER -> STACK  
POP: STACK -> STACK  
TOP: STACK -> INTEGER
```

(TOP returns the value in the top of the stack without removing it, while POP removes the value without returning it.)

Note that more than one domain appears in the specification; this is true for almost all interesting data abstractions. Normally, only one of these (the domain of stacks in the example) is being defined; the remaining domains and their properties are assumed to be known. Given this distinction, the group of operations can be partitioned into three blocks. The first block, the primitive constructors, consists of those operations that have no operands in the domain being defined, but which yield results in the defined domain. This block includes the constants, represented as argumentless operations (for example, the CREATE operation for stacks). The second block, the combinatorial constructors, consists of those operations (PUSH and POP in the example) that have some of their operands in and yield their results in the defined domain. The third block consists of those operations (TOP for stacks) whose results are not in the defined domain.

The semantic part of the specification uses the symbols introduced in the syntactic part to express the meaning of the data abstraction. Two different approaches are used in capturing this meaning: either an abstract model is provided for the class of objects and the operations defined in terms of the model, or the class of objects is defined implicitly via assertions of properties of the operations.

In following the abstract model approach, the behavior is actually defined by giving an abstract implementation in terms of another data abstraction, one whose properties are well understood. The data abstraction being used as the model also has a number of operations, and these are used to define the operations of the new data type.

The approach of defining the objects implicitly via descriptions of the operations is much closer to the way mathematical theories are usually defined. Axioms are given that describe the behavior of the operations. The domain or class of objects is determined inductively. Usually it is the smallest set closed under the operations. Only those operations identified above as constructors are used in defining this closure.

The closure is the smallest set containing the results of the primitive constructors and the results of the combinational constructors when the appropriate operands are drawn from the set. For example, with stacks, the only primitive constructor is the constant operation CREATE, which yields the empty stack, and the class of stacks consists of the empty stack and all stacks that result from applying sequences of PUSH's and POP's to it. One difficulty with the implicit definition approach is that if the specifications are not sufficiently complete, in the sense that all the relationships among the operations are indicated, several distinct sets may be closed under the operations. The distinct sets result from different resolutions of the unspecified relationships.

In [8] a number of specification techniques for data abstractions were surveyed and compared. Two abstract model approaches were considered: use of a single fixed modelling domain (e.g., graphs or sets) and use of an arbitrary fixed modelling domain. When using a single fixed domain, the specifications are usually easily understood and easily constructed by someone familiar with the modelling domain, if they describe concepts within the range of applicability of the chosen domain. However, a fixed modelling domain usually has a somewhat limited range of applicability; only certain abstractions are expressed easily within the domain. Using such a technique is similar to writing programs in a programming language that provides a single data structuring method; although a single method can be powerful enough to implement all user-defined data structures, it does not follow that all data structures are implemented with equal facility. This limitation is somewhat mitigated by allowing the specifier to make use of an arbitrary fixed modelling domain. However, the number of domains available for use is not large, and, in addition, if a completely free choice of domains could be made, it is doubtful that the resulting specification would be comprehensible. Thus, in reality, the specifier must choose among a small number of domains. This situation is analogous to writing programs in a language providing several data structuring facilities; programming experience indicates that there will always be (problem oriented) abstractions that cannot be ideally represented by any of the data structuring methods. Thus, it appears unlikely that all data abstractions can be given minimal specifications by choosing among a small number of modelling domains.

Included among the implicit definition approaches are the state machine model approach of Parnas [9], and the algebraic approach of Zilles [10] and Guttag [11]. The state machine model approach as originally described by Parnas is not a formal technique: English is used to describe behavior when all else fails. Two techniques are being investigated to correct this: the hidden function approach at S.R.I. [12], and a new approach by Parnas [13]. The hidden function approach appears to introduce an abstract model to describe behavior, while the approach of Parnas uses axioms to express the behavior of the abstraction as a whole, and appears fairly close to the algebraic approach. Both approaches require more development before their properties will be known.

The algebraic approach of Zilles appears to be quite promising. From the syntactic part of the specification (the functionality of the operations) the set of legal, finitely constructible expressions in the operations can be defined; these expressions

are the words of a word algebra. Then axioms are given that specify when two words are equivalent; an example of such an axiom, for the stack example given earlier, is:

$$\text{pop}(\text{push}(s,i))=s$$

where s is a stack, and i is an integer. All words whose equivalence does not follow from the axioms are taken to be distinct.

The algebraic approach can be used to construct minimal specifications, containing no extraneous information, and there is no limit on the range of applicability. The main problem is that it may prove difficult to construct and comprehend these specifications, because they are so abstract. It is difficult to be certain that a set of axioms is complete and consistent. However, our experience in using the technique indicates that it is reasonably easy to use. In addition, tools can be devised to help the specifier determine the consistency, completeness, and meaning of these specifications [11].

As was explained in the introduction, the study of specification techniques was motivated by the desire to enhance the quality of software. Specifications are not only useful in proving the correctness of programs; they are a valuable aid during the process of system design. When a program is developed by stepwise refinement [14, 15], the problem of concern at a program level is solved by introducing abstractions that provide useful primitives for that problem domain. The original problem is solved in terms of the abstractions; each abstraction then becomes a new problem to be solved. Specifications provide a way to make this process precise; if each abstraction is defined completely by means of a specification, then we can be sure the implementor of a program using an abstraction and the implementor of the abstraction agree about the meaning of the abstraction. This is particularly important for large programs in which the abstractions may be implemented by different people.

In addition, we have found the actual writing of the specifications to be a valuable addition to the design process: if an abstraction has a complicated specification, often a better form of the abstraction with a simpler specification can be found. The provision of tools for examining properties of specifications in advance of implementation, which will be possible when specifications are added to the CLU system, also appears promising.

REFERENCES

1. Liskov, Barbara H. "A Design Methodology for Reliable Software Systems." AFIPS Conference Proceedings. Vol. 41, 1972, 191-199.
2. Parnas, David L. "On the Criteria to be Used in Decomposing Systems Into Modules." Communications of the ACM, Vol. 15, No. 12 (December 1972), 1053-1058.
3. Liskov, Barbara H., and Zilles, Stephen N. "Programming With Abstract Data Types." Proceedings of the ACM Conference of Very High Level Languages. SIGPLAN Notices, Vol. 9, April 1974, 50-59.
4. Liskov, Barbara H. A Note on CLU. M.I.T., Laboratory for Computer Science, Computation Structures Group Memo 112-1. Cambridge, Ma., November 1974.
5. Parnas, David L. "Information Distribution Aspects of Design Methodology." Proceedings IFIP Congress. August 1971, 340-344.
6. Wegbreit, Ben. "The Treatment of Data Types in EL1," Communications of the ACM, Vol. 17, No. 5 (May 1974), 251-264.
7. Galley, Stuart W., and Pfister, Greg. The MDL Language. M.I.T., Laboratory for Computer Science, Programming Technology Division Document SYS.11.D, Cambridge, Ma., in progress.
8. Liskov, Barbara H., and Zilles, Stephen N. "Specification Techniques for Data Abstractions." IEEE Transactions on Software Engineering, Vol. SE-1, No. 1 (March 1975), 7-19.
9. Parnas, David L. "A Technique for the Specification of Software Modules with Examples." Communications of the ACM, Vol. 15, No. 5 (May 1972), 330-336.
10. Zilles, Stephen N. Algebraic Specification of Data Types. M.I.T., Laboratory for Computer Science, Computation Structures Group Memo 119, Cambridge, Ma., March 1975.
11. Guttag, John V. The Specification and Application to Programming of Abstract Data Types. University of Toronto, Computer Systems Research Group, CSRG-59, Toronto, Canada, 1975.
12. Robinson, Lawrence; Levitt, Karl; Neumann, Peter; and Saxena, Ashok. "On Attaining Reliable Software for a Secure Operating System." Proceedings of the International Conference on Reliable Software. SIGPLAN Notices, Vol. 10, No. 6 (June 1975), 267-284.

PROGRAMMING METHODOLOGY GROUP 90 PROGRAMMING METHODOLOGY GROUP

13. Parnas, David L., and Handzel, G. More on Specification Techniques for Software Modules. Fachbereich Informatik, Technische Hochschule Darmstadt, Federal Republic of Germany, 1975.
14. Dijkstra, Edsger W. "Notes on Structured Programming." Structured Programming. APIC Studies in Data Processing No. 8. New York: Academic Press, 1972, 1-81.
15. Wirth, Niklaus. "Program Development by Stepwise Refinement." Communications of the ACM, Vol. 14, No. 4 (April 1971), 221-227.

PROGRAMMING METHODOLOGY GROUP 91 PROGRAMMING METHODOLOGY GROUP

Publications

1. Laventhal, Mark S. "Verifying Programs Which Operate on Data Structures." Proceedings of the International Conference on Reliable Software. SIGPLAN Notices, Vol. 10, No. 6 (June 1975).
2. Liskov, Barbara H. "Data Types and Program Correctness." Proceedings of the AFIPS 1975 National Computer Conference. May 1975.
3. Liskov, Barbara H. and Zilles, Stephen N. "Specification Techniques for Data Abstractions." IEEE Transactions on Software Engineering, Vol. SE-1, No. 1 (March 1975), 7-19; Also, Proceedings of the International Conference on Reliable Software. SIGPLAN Notices, Vol. 10, No. 6 (June 1975).
4. Schaffert, J. Craig; Synder, L. Alan; and Atkinson, Russell R. The CLU Reference Manual. M.I.T., Laboratory for Computer Science, CLU Design Note 39-1, Cambridge, Ma., June 1975.
5. Snyder, L. Alan. A Portable Compiler for the Language C. M.I.T., Laboratory for Computer Science, MIT/LCS/TR-149, Cambridge, Ma., 1975.
6. Zilles, Stephen N. Algebraic Specification of Data Types. M.I.T., Laboratory for Computer Science, Computation Structures Group Memo 119, Cambridge, Ma., March 1975.

Theses Completed

1. Fylstra, Daniel H. "Optimization of Arithmetic Expression." unpublished S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1975.
2. Henderson, D. Austin Jr. "The Binding Model: A Semantic Base for Modular Programming Systems." unpublished Ph.D. Thesis, M.I.T., Department of Electrical Engineering, February 1975.
3. Mui, Tony. "Features of Structured Programming Demonstrated by the Index Production Program." unpublished S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1975.
4. Sopelak, Alan B. "Compile Time Checking for Variable Initialization in CLU Programs." unpublished S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, June 1975.

PROGRAMMING METHODOLOGY GROUP 92 PROGRAMMING METHODOLOGY GROUP

Theses in Progress

1. Atkinson, Russell R. "Optimization Techniques for a Structured Programming Language." S. M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, 1976.
2. Fulton, Gordon L. "A Microprogrammed Instruction Set for a 32-bit Minicomputer." S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, 1976.
3. Gorgen, David P. "An Algorithm to Determine Mutability of Data Types in CLU." S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, 1976.
4. Isaman, David L. "Systems of Data-Structuring Operations for Parallel Processors." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, 1976.
5. McCabe, Edward J. "A Compactifying Garbage Collection Algorithm for a Typed Programming Language." S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, 1976.
6. Schaffert, J. Craig. "Specifying Meaning in Object Oriented Languages." S. M. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, 1976.
7. Scheifler, Robert W. "An Analysis of Inline Substitution for the CLU Programming Language." S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, 1976.
8. Virgile, Kenneth. "MEIL: A Macro Expandable Intermediate Language." S. B. Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, 1976.
9. Zilles, Stephen N. "Data Algebra: A Specification Technique for Data Structures." Ph.D Thesis, M.I.T., Department of Electrical Engineering and Computer Science, expected date of completion, 1977.

Talks

1. Laventhal, Mark S. "Verifying Programs Which Operate on Data Structures." International Conference on Reliable Software, Los Angeles, Ca., April 1975.
2. Liskov, Barbara H. "Programming with Abstract Data Types." IBM Research Center, Yorktown Heights, N.Y., January 1975.

PROGRAMMING METHODOLOGY GROUP 93 PROGRAMMING METHODOLOGY GROUP

3. Liskov, Barbara H. "Programming with Abstract Data Types." Intermetrics, Cambridge, Ma., March 1975.
4. Liskov, Barbara H. "Specification Techniques for Data Abstractions." International Conference on Reliable Software, Los Angeles, Ca., April 1975.
5. Liskov, Barbara H. "Practical Benefits of Program Verification." First National Conference on Reliable Software, Washington, D.C., September 1975.

PROGRAMMING TECHNOLOGYAcademic Staff

A. Vezza, Group Leader

J. C. R. Licklider

Research Staff

E. R. Banks
E. H. Black
M. F. Brescia
M. S. Broos
H. I. Badian

S. W. Galley
J. F. Haverty
P. D. Lebling
J. C. Michener
C. L. Reeve

Graduate Students

T. A. Anderson
S. E. Cutler

B. K. Daniels
G. D. McGath

Undergraduate Students

J. M. Berez
B. T. Berkowitz
M. Blank
K. W. Church
J. Connelly
D. L. Dill
J. J. Heeger
A. G. Jaffer
J. H. Morrison

J. R. Pollack
S. H. Soto
J. D. Sybalsky
G. A. Thompson
T. To
K. E. Van Sant
J. Westcott
C. K. Yap

Support Staff

S. B. Pitkin

PROGRAMMING TECHNOLOGYA. INTRODUCTION

Morse code is the major research and development effort of the Programming Technology Group. The work of the group is directed toward the development of a prototype computer system, the behavior of which is as similar as possible to that of a human Morse-code operator in a radio environment containing noise and interfering signals. Our approach requires the system to be rich in Morse-code-specific knowledge. Our results are embodied in a feasibility demonstration system called COMCO-1 (COMputerized MORse CODE Operator). Our effort is focused on design requirements, system design, techniques and algorithms to effectively supplement such a Morse-code operator.

Another component of the group is the development of a prototype computer message system that is based on a data-base management system.

The group's work is carried out on the Dynamic Modeling System, a programming environment based on a LISP-like language and an expanding library of programs.

B. EXPERT KNOWLEDGE APPLIED TO THE MORSE-CODE DOMAIN1. Transcription

Work on COMDEC, a transcriber of hand-sent Morse code, advanced considerably during the past year (Lebling, Haverty). The first part of the year was spent in design and implementation of a successor to a preliminary run-length-sequence transcriber [1].

Errors in hand-sent Morse code are of three basic types, if the problems associated with the radio domain are omitted. These three types are spacing errors, mark errors (a mark is a dot or a dash), and spelling errors (which may be treated as a special category of mark error).

A spacing error occurs when a sender does not keep to the proper ratios between inter-mark (within a letter), inter-letter (within a word), and inter-word spaces. The result is analogous to spoken language that is slurred or broken by arbitrary pauses. In ideal (machine-sent) code the ratio of space types is 1:3:7. That is, if an inter-mark space is 100 milliseconds long, an inter-letter space should be 300 milliseconds, and so on. If the ratios are not too far off the ideal, and do not vary too rapidly, the code can still be transcribed by a relatively simple moving-average procedure [2, 3]. The first module of COMDEC is such a procedure. However, COMDEC treats the transcription so produced as only a "suggestion," and can proceed from there to correct errors too severe to be handled by a moving-average transcriber.

A mark error occurs when a sender omits, adds or changes the sense of one or more of the marks (dots and dashes) making up a word. In practice, most mark errors are one of a small number of "simple" mark errors, which do not change the word as a whole sufficiently to make it unrecognizable to the average Morse-code operator.

The COMDEC control structure was designed to consist of any number of modules, each of which would be an "expert" on one aspect of transcription. Each module could add suggested transcriptions to a lattice of possible transcriptions. Successive modules would decide whether further error correction was necessary by examining the quality of existing suggested transcriptions. After a module has examined a section of code, and inserted its suggested transcriptions in the lattice, it passes that section of code to the next module in the chain, and so on, until each transcriber module has examined the entire message. A major part of the design was an N-dimensional metric for measuring trial transcriptions and a highly heuristic algorithm for comparing the measures.

Decoder modules are ordered approximately by the severity of the sending errors they are able to correct. Thus, spacing errors ("THE" sent as "T HE", or "IT IS" sent as "ITIS") are corrected first. Later, mark errors ("THE" sent as "THT") are corrected.

The first module to process a code sample is a moving-average transcriber, a hybrid of Selfridge's MAUDE [2] and Poehler's FRAUD [3]. It classifies marks and spaces into the appropriate types by comparison with continuously updated averages for each type and thresholds between types. More importantly, it associates with each mark and space a number which represents its confidence that the classification was correct. These assignments and confidences are used by later modules to select areas of the sample where errors appear to have occurred.

The spacing-error correction modules are able to correct errors in which:

- a. letters are run together (for example, "THE" sent as "6E"),
- b. letters are split apart ("THE" sent as "TIE"),
- c. words are run together ("IT IS" sent as "ITIS"), or
- d. words are split apart ("ORGANIZING" sent as "OR G AN I ZING").

Cases in which the preceding errors occur together or several times in one section of code are also handled.

The mark-error correction modules (Lebling, Haverty, Banks) are able to correct eight different classes of mark error, in words which contain at most one mark error. Statistically, the vast majority of mark errors are of the types COMDEC can correct, and occur only one to a word. The types of mark errors corrected by COMDEC are

- a. sending an extra dot,
- b. sending an extra dash,
- c. sending two extra dots,
- d. running two dots together as a dash,
- e. splitting a dash into two dots,
- f. dropping a dot,
- g. dropping a dash, and
- h. dropping two dots.

During the year, the size of COMDEC's dictionary of English words has been increased by more than a factor of three, from 1,300 to 4,200. Additionally, a morphology program embedded in the transcriber has been improved to handle more classes of word endings. Thus, a word usually appears in the dictionary only in its root form, and endings such as "-ing," "-ed," or "-s" are added as needed. In its current version COMDEC's morphology program understands the endings "-s," "-ed," "-ing," "-er," "-est," "-ly," "-tion," "-ment," and combinations of the preceding, such as "-ers." COMDEC has the knowledge of which words take each ending, and thus the effective size of the dictionary is several times the number of roots in it and begins to approach the size of the conversational vocabulary of the average English speaker (its size is approximately 18,000 words when all inflections are considered).

Due to improvements in dictionary lookup, mark-error correction, and primary-storage residency, the operational speed of the transcriber has improved to the point where it can usually transcribe a code sample in less than one-tenth the time it took the sender to transmit it. If the code is well sent, transcription is correspondingly faster.

An example of COMDEC's transcription abilities follows. The first section shows the result produced by the MAUDE-like moving average transcriber used by COMDEC as a first pass. The text consists of the first paragraph of the Declaration of Independence, with punctuation removed.

WE HOLD TH ESE TRUTHS TO BE SELF EVIDI N T THAT ALL MEN ARE CREATED E Q U AL
THAT THEY ARE ENDOWED BY THEIR CAEATOR W I T H CERTAIN UNALIENABLE RIGHTS
THAT AMONG TH ESE A R E LI FE LIBERTY AND THE PURSUIT OF HAPPINESS THAT TO
SECU RE TH ESE RIG H T S GOVERNMENTS ARE INSTITUTED AMONG MEN DER IVING
THEIR VU ST POWERS FROM THE CONSENT OF THE G OVERNED THAT EM HEN E VER
ANYFORM OF GOVERNMENT BEC OMES DE S TR U C TIVE OF THESE ENDS IT IS THE

RIGHT OF THE PEOPLE TO ALTER OR TO ABOLISH IT AND TO INSTITUTE NEW GOVERNMENT LAYING ITS FOUNDATION ON SUCH PRINCIPLES AND ORGANIZING UNDER POWERS IN SUCH FORM AS TO THEM SHALL SEEM MOST LIKELY TO PROTECT THEIR SAFETY AND HAPPINESS double-hyphen attention

This code sample was taken on our own equipment. The sender confessed that he was rusty, and the code shows it. The sample was supposed to end with the punctuation mark "end-of-message", but he had forgotten what it was and so sent two completely different punctuation marks. The sample was sent in just under six minutes.

The COMDEC transcription follows. Words whose transcription was found by correcting a mark error are enclosed in brackets <thus>.

WE HOLD THESE TRUTHS TO BE SELF <EVIDENT> THAT ALL MEN ARE CREATED EQUAL THAT THEY ARE ENDOWED BY THEIR <CREATOR> WITH CERTAIN UNALIENABLE RIGHTS THAT AMONG THESE ARE LIFE LIBERTY AND THE PURSUIT OF HAPPINESS THAT TO SECURE THESE RIGHTS GOVERNMENTS ARE INSTITUTED AMONG MEN DERIVING THEIR <VAST> POWERS FROM THE CONSENT OF THE GOVERNED THAT WHEN EVER ANY FORM OF GOVERNMENT BECOMES DESTRUCTIVE OF THESE ENDS IT IS THE RIGHT OF THE PEOPLE TO ALTER OR TO ABOLISH IT <AND> TO INSTITUTE NEW <GOVERNMENT> LAYING ITS FOUNDATION ON SUCH PRINCIPLES AND <ORGANIZING> <SAD> POWERS IN SUCH FORM AS TO THEM SHALL SEEM MOST <LIKELY> TO {TFFETGT} THEIR SAFETY AND HAPPINESS double-hyphen attention

COMDEC produced this transcription in approximately 40 seconds of processing time. There are three places in this sample where COMDEC did not produce the correct transcription. In all three the error was a type of mark error COMDEC is not designed to correct at present. Two of the three were words containing two or more mark errors. "JUST", which in Morse is (--- -- ... -), was sent as "VUST", which in Morse is (...- -- ... -); that is, two successive dashes were sent as dots. COMDEC corrected it, by assuming an extra dot had been sent, to "VAST". "EFFECT" was sent as "TFFETGT", because two separate dots were sent as dashes. COMDEC left that section of the message untranscribed, as indicated by the braces {} surrounding it. The third error is one which future improvements to COMDEC should eliminate. It occurred because the mark error (a dot sent as a dash), occurred in the ending ("-s") that was added to a root ("it"), to make the word. Thus "ITS" was converted to "SAD".

Future development of COMDEC will concentrate on adding knowledge about two separate domains: the radio domain and the English-language domain.

AD-A061 246

MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTE--ETC F/G 9/2
LABORATORY FOR COMPUTER SCIENCE (FORMERLY PROJECT MAC) PROGRESS--ETC(U)
AUG 78 M L DERTOUZOS

N00014-75-C-0661

UNCLASSIFIED

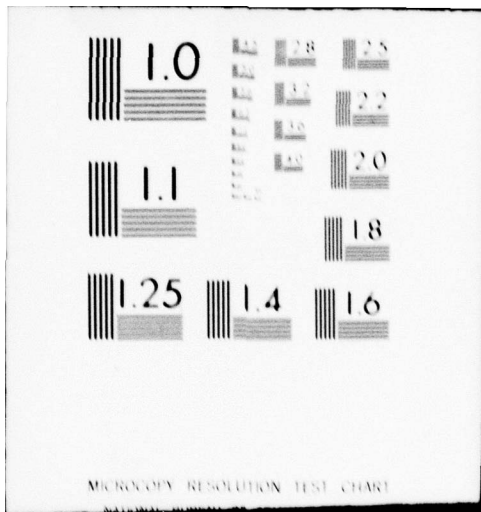
LCS-PR-13

NL

2 of 2
AD
A061 246



END
DATE
FILMED
1-79
DDC



2. Code Sampling

The MREAD program was written (Lebling) to take hand-sent Morse-code samples from a code key connected to a mini-computer (Imlac PDS-1D) and transmit that sample to a file on the DMS disk (Haverty, Soto). The program enabled us to take samples from local Morse-code senders. These samples could be used by the various transcription programs. MREAD allows the samples to be played back at any time and also incorporates a simple MAUDE-like transcriber which permits the person taking the sample to make a quick estimate of the quality of the sample. MREAD stores with the code sample such information as the date and time the sample was taken, which text sample was sent, and what sender was keying. This information is available to the transcriber programs. MREAD maintains both visual and aural displays of the progress of a sample, both when it is taken and when it is played back.

3. Cipher Group Transcriber

A transcriber was designed and implemented (Anderson) for n-letter cipher groups (such as "EQYPN QXTTL ..."). Since it is impossible to base such a transcriber on a dictionary, an approach quite different from that taken by COMDEC was needed. However, there is some context associated with cipher-group Morse, namely that the Morse is typically blocked into groups of four, five or six characters. All groups in a message are supposed to be of the same length, and, typically, alphabetic and numeric characters are not mixed in the same group. While this is not much context information, it is very important because operators do not usually differentiate distinctly the group boundaries, but, out of necessity, differentiate the letters. The transcriber of necessity assumes that its first pass (a MAUDE-like transcriber) has produced a relatively good transcription, which needs to be fixed in places. It therefore tries to find those places which look bad; when it reaches one, it tries a nearly exhaustive search to find a code group to fit the region of code. Ideally, most of the message will contain a few trivial errors, which can be corrected directly. This approach is usually successful; further work is needed to improve performance on code which is sent poorly.

In conjunction with this effort, a version of MAUDE (as opposed to the "MAUDE-like" transcribers generally used at LCS) was implemented and tested; it demonstrates principally that no one of the context-insensitive transcribers is better in all cases than any other one when used as a first pass for our context-sensitive transcribers. Some messages which fail miserably in MAUDE will do quite well in the "MAUDE-like" transcribers, and vice versa.

The n-letter transcriber was intended for use in research on sender characteristics: such text is entirely lacking in linguistic context, and therefore seemed like a good baseline from which to measure linguistic context-dependent variations in Morse-code messages.

4. Morse-Code Network Conventions

The design and implementation of programs was begun which perform Morse operator functions in an environment modeled closely after the conventions and procedures used in amateur radio "traffic" networks which handle "radiograms" throughout the United States (Haverty, Lebling, Church, Heeger, Vezza) [4].

The traffic networking activity is divided into two parts. One part concerns the activity conducted on the network's primary frequency, where many stations interact under the direction of a manager station called the Net Control Station, or NCS. The purpose of this activity is to permit the various participants to inform the NCS of the quantity, type, and destinations of messages which they wish to transmit, as well as the locations for which they can accept messages. The NCS is responsible for matching senders and receivers of messages appropriately, and dispatching them to side frequencies for actual transmission of the messages.

The second part of the activity concerns the interaction between two stations that have been directed to transfer messages. This involves the operations of establishing communication on the side frequency, negotiating some parameters of the interaction such as speed, transferring the message, obtaining repeats of sections garbled by noise, and confirming the receipt of the message. At that point the stations move back to the primary net frequency and report the results of their activity to the NCS.

Each of these parts involves interaction between stations which is carried out in Morse-code Q-sign and pro-sign language, usually referred to as Morse network chatter. The protocol that operators are supposed to use is well-defined although often ambiguous in meaning if taken out of context. The task is made even more difficult because the protocol is often violated. Thus, a system which must understand network chatter must be quite tolerant of inconsistencies and must continue to perform in a reasonable manner, even though it is not able to interpret the communications properly.

Three projects concerned with developing programs to understand aspects of the Morse network dialog were initiated. The programs per se developed for these initial projects are not expected to be pursued to their final form, but are instead intended to uncover some of the difficulties that will be encountered in the development of programs to understand Morse network chatter and model the network situation. The first project concerned the development of a program to monitor the communication occurring on the primary net frequency, and to create and update a data base which models the state of the net at all times (Church, Haverty, Vezza). Changes to this data base as activity occurs are printed to determine the accuracy of the program in interpreting the activity. The program is modelled as one of the participants in the net listening to the activity and reporting the progress of the net with time. As such, the program cannot assume that it can hear all other stations in the net, but it is capable of making hypotheses about the state of the net. The major purpose of this program is to maintain a dynamic model in

the computer of the net as it progresses. This data base would be used by other programs which interact with the net.

The second project concerned the development of a program to perform the functions of the NCS (Heeger) [5]. This involves understanding transmissions by the various net participants, maintaining a model of the state of the net, and generating appropriate responses for the NCS in each case.

The third project concerned the development of a program to operate in the situation where two stations are actually transferring a message (Dill) [6]. This involves interpretation of the conventions and formats used in the header of a message and where the body of a message begins and ends. (Headers are that portion of a message containing information such as address, originator, priority, etc.) The program is capable of handling messages perfectly when they adhere to the protocol, and it can perform very credibly at interpreting headers which depart from the strict protocol to a minor extent, by using contextual information and some heuristics based on the expected contents of the headers.

5. Morse-Code Laboratory

An experimental system was designed and constructed (Haverty, Cutler) during autumn and winter to be used in various phases of the Morse-understanding project. This system provides a real radio environment on a coaxial cable network, for use in experiments. Currently nine stations and an elaborate "processing station" with a computer interface are connected to the cable. The cable network can be used to create experimental conditions using human operators at the various stations, simulating a radio "universe" in which the nine stations interact, interfere, and otherwise behave as actual transmitters in a radio environment.

The experimental system permits parameters such as the quality of the transmitter and signal strength of various stations to be varied in a controlled manner. Each station transmitter is connected to the network through an accurate stepped attenuator to permit comparative measurements of performance. Individual transmitters may be simply modified as desired to introduce key clicks and chirp.

The processing station consists of equipment to acquire and process the Morse data on the cable network. It also provides a means to faithfully reproduce signals on the cable to provide for controlled experiments to test programs during development. The processing station consists of the following equipment:

- a. Collins 651S-1 receiver
- b. DEC PDP-11/10 computer with extended arithmetic unit, real-time programmable clock, and Computer Labs disk system

- c. Crown SX-824 audio tape recording system
- d. Heath SB-104 transceiver
- e. ADAC analog-to-digital converter
- f. Active analog filter

In operation, the cable input is supplied to the Collins receiver. This receiver is computer controllable and interfaced to the PDP-11. Programs are now under development to simulate the actions of a human operator in acquiring, tracking, and tuning the receiver to acquire a desired signal (Haverty).

The analog-to-digital converter on the PDP-11 provides raw digitized data from the audio output of the receiver. The receiver section of the Heath unit is used as a wide-band (3 kilohertz) monitor of the area around the signal being tracked. This data provides the program, through another channel of the analog-to-digital converter, with global information as to the immediate environment of the signal of interest, so that heuristics to adjust the Collins receiver for better reception can be used.

The transmitter section of the Heath unit is used to recreate particular experiments on the network cable to enable testing of the various signal-processing programs as they are developed. The Collins tape recorder is used to record the wide-band monitor output during a given experiment, thereby obtaining a permanent record. This audio record can be used at any time thereafter as "voice" input to the single-sideband transmitter. Because of the nature of single-sideband and Morse transmissions, this effectively recreates in the radio domain on the cable the conditions which existed when the tape was made.

Since much of the work being done involves creating programs which simulate an operator's actions under real-world conditions based on a model of actual situations encountered in the amateur radio domain, an antenna system was selected and installed to provide the capability for licensed amateur operators working on the various projects to observe (and participate in) actual radio situations, and to use that knowledge to develop the computer models. This system consists of a HY-gain TH6DXX beam antenna and 14AVQ vertical antenna to cover the frequencies of interest.

The PDP-11 has been interfaced to the DMS PDP-10 and the Collins receiver (Black). Subroutines have been written to transfer information to both devices and to obtain data from the ADAC analog-to-digital converter. The Collins unit can be both controlled and interrogated by the PDP-11.

Some facilities were implemented (Black, Haverty) for digital signal processing, including simulation of digital filters, a Chirp Z-transform and a sliding-window fast Fourier transform.

6. Signal Processing

In detection of Morse signals, the primary areas of interest in the incoming data are the transition points. These are the sections of the time signal where one of the transmitting stations is beginning or ending a mark. The time between each such transition for a station determines the length of each mark and space which defines the Morse element being transmitted. This information is the primary input to COMDEC for transcribing into characters.

The goal of a signal-processing module is to identify the transition points for the signal being copied, and to pass that data to the COMDEC phase of the transcriber. A procedure was designed (Haverty) this year, and is now being implemented (Jaffer) for testing, to attack the problem of extracting the mark-space information for a desired signal from a real radio environment.

The procedure involves acquisition of a "signature" for the upward and downward transitions of each signal of interest. In any short sequence under acceptable conditions, there is invariably a point where a desired signal makes a transition which does not overlap with any other signal's transition. The algorithm finds such a transition in the signal to be transcribed, and forms a model, or signature, of the transition involved.

Given a signature for the signal of interest, the processing procedure involves locating the various transitions as time progresses, and comparing the metric for each with the stored set of signatures, to develop a hypothesis concerning which signal owns the transitions involved.

C. MESSAGE SYSTEM

1. MSGDMS

Design and implementation of a message system on TENEX which incorporated the features of our ITS message system began (Broos, Vezza). Specifically, each user has a private relational data base [8] containing her or his messages, which can be searched to find messages of interest. These message-retrieval facilities are interfaced to the message-composition facilities to enable the user to load messages from his or her data base into composition buffers, and to dump the contents of the buffers into the data base. A working system was quickly made operational by using existing code, despite inefficiencies. The goals of this initial implementation were:

- a. to demonstrate meaningful and speedy transfer of technology from ITS to the TENEX operating system (both utility subroutines and various programming tools (such as our public program library) were transferred)
- b. to demonstrate at a TENEX site the functional power of a message system which was built on top of a data management system. (The ITS message system is built along these lines, but it is available only to registered MIT-DMS users.)

The new message system (called MSGIRS) was designed and an initial version was made available to a select group of test users in October. Various improvements were suggested by the test users, and a second version of the message system (now called MSGDMS) was produced and made available to a larger group of users in November. At that time, a first version of the MSGDMS user's manual and lesson plans were made available to the enlarged test group.

In January we started to rebuild MSGDMS from the ground up. The existing system, while very powerful, was too slow and inefficient to be useful to a general user community. Also, many users had found that the style of interaction with the system seemed unnatural to them. The redesign effort was therefore carried out along two fronts. The user interaction, conceptual framework, and command structure were reworked into an integrated user interface which retained all of the valuable functional capabilities of the old system. At the same time, a new internal architecture was conceived which eliminated the bottlenecks that had degraded the old system's performance.

The major architectural change to MSGDMS was the elimination of the old user-level data-base functions while retaining the lower-level facilities. By eliminating the middleman, a great many inefficiencies disappeared. The remaining data-base facilities were completely rewritten and some new packages created:

- a. The programmer-level disk-access package was rewritten completely. Several new features were added to complement the operating system, such as the ability to update an existing file using page-mapping.
- b. The disk-resident inversion manipulation package was rewritten to use the above disk-access package. A more compact data format was used, and several new capabilities (such as the use of "nand" and "nor") were introduced.
- c. A new package was created to provide the capability to build, maintain, search, store and retrieve lexicons or user-interface token-completion tables. The format of the tables was designed so they could be written to and read from disk files directly, without relocating pointers or using temporary storage.

- d. Another new package was created to provide the capability to create and modify messages using a "direct-copy" format, in the above sense. It was very important to be able to read in, display, modify, and write out individual messages quickly and directly. This package is fairly message-specific and rudimentary in terms of allowed data types and operations. A more general package, based on similar principles, is now being designed.

It should be pointed out that the new packages created or rewritten in the course of MSGDMS development are designed for general data-base management use, not just for MSGDMS or other message systems. A new relational data-base management system could eventually be constructed from these improved, low-level primitives.

A user interface for the MSGDMS search handler has been written (Black). The front end consists of a state-machine interpreter for the CALICO user interface which guides the user through a search request, attempting to obviate syntactic inconsistency by limiting the symbols available to those which are allowed. When a search command is obtained, requests are formulated for the data-base management system (Broos). These requests are optimized in the sense that linear searches are deferred as long as possible, and the order of processing is controlled by the size of the sets being processed. An example of a possible search command is "all messages dated after June 7 from VEZZA to BROOS or HAVERTY." In this case, every element typed (except for June 7) would have the token-completion facilities of CALICO available. The arguments to fields which are inverted ("from" and "to" are typical) are restricted to those values which are actually in the data base. If dates are not inverted in the data base (currently true), the date search would be performed last.

2. Message Protocol

Progress continued on efforts to develop a successor to the current ARPANET protocols for transmitting messages. The primary goal of this effort is to obtain a more machine-oriented protocol which will permit introduction of more powerful facilities into message-handling systems.

Several members of the division participated in meetings of the ARPANET Message Services Committee. A proposal [8] by Haverty and others for a new message-handling structure and protocols for use by cooperating message server programs was presented to the Committee in July 1975 for comments. This proposal was based largely on experience gained from the construction and development of the message facilities at MIT-DMS and the MSGDMS system implemented on TENEX.

Based on this proposal and comments received, a revised proposal was developed (Broos, Haverty, Vezza), and presented in December [9]. At the request of the Committee, the various general-purpose sections of the protocol are being extracted and presented to the ARPANET community for use.

3. Message Composer

During the year, the MIT-DMS message system was improved (Haverty, Lebling, Blank, Berez) to increase its efficiency and provide additional facilities, such as the ability to include notes to an individual addressee of a message. Other facilities, such as expiration dates, were implemented for intra-site messages, but cannot be directly used for inter-site messages until a more powerful protocol is adopted.

A message composer was implemented in MDL to provide the message composing power of the more sophisticated Reader/Composer but with a simpler user interface and a smaller load on system resources (Lebling, Blank). It was also intended to resemble the old ITS DDT :MAIL command. The new composer has a control structure that will enable it to be integrated with a message-display program.

4. CINCPAC Test

Design of a message system for the CINCPAC military message experiment began in January (Vezza, Broos, Haverty, Black). The design of military security controls has proceeded through a number of internal papers and meetings with other developers and all of the principals involved in the project. Work continues on the security and terminal designs.

D. DMS ACTIVITIES

The "Dynamic Modeling System" (whence our ARPANET identification MIT-DMS) comprises: (1) a large consistent set of tools that serve as a base for software development; (2) the high-level programming language MDL; (3) its coherent user interface CALICO; (4) the support for libraries of program modules, program abstracts, messages, etc.; and (5) a growing collection of programs in the program libraries.

The Programming Technology Group presented a full-day symposium in March on "Advanced Programming Techniques" under the auspices of the M. I. T. Industrial Liaison Program for more than 200 industry representatives. The program included talks (Professors Hammer and Liskov; Vezza, Licklider, Galley, Lebling) on programming languages, the Dynamic Modeling System, debugging, and program abstracts, and a question period (Galley, Haverty, Lebling, Reeve, Vezza).

1. CALICO

The transfer of the user interface CALICO from ITS to TENEX continued (Black). The version on TENEX now has the same capabilities as that on ITS, including system-dependent commands (display of file directories, creation of inferior processes, etc.). The help facilities were expanded (Black) and considerably improved (Broos) to allow descriptions of commands to be obtained using CALICO commands.

The most active users of CALICO are those of message systems, so improvements come mainly as the result of feedback from this area. Enhancements that were added (Black) were means for:

- a. reducing prompts on a permanent or per-command basis, allowing experienced users to proceed as rapidly as they desire
- b. tailoring the text-input package so that it appears to the user to be an integral part of CALICO
- c. informing application programs when the user tailors CALICO so that they can remember the user's "state" for future sessions.

2. MDL

The MDL interpreter and compiler (Reeve, Berkowitz, Daniels) both had features added this year.

The interpreter (Berkowitz, Reeve) has a new full-copy garbage collector; built-in functions to purify MDL objects and to dump them to secondary storage, to be reproduced exactly in MDL address space later; more flexibility in user-defined data types; improved pure-program storage; a faster function-call instruction; and new software interrupts. The MDL manual has been kept up-to-date with these changes (Galley).

The compiler's analysis phase was redesigned and implemented to extract data-type information from program context. It does a very thorough job of extracting such information. The analysis phase also includes an analysis of the life and death of variables. Using life-and-death analysis and other flow information, the compiler keeps values in fast registers in more cases where it is advantageous to do so and does not put them back into primary memory unless absolutely necessary. Special-case code can be produced for certain common code sequences; for example, `<REST .list <- <LENGTH .list> 1>>` (remove all but the last element of a list) is compiled to go down the list only once, and `<==? 3 <LENGTH .list>>` (does a list have exactly 3 elements?) goes only far enough down the list to test the predicate. Certain mathematical transformations are made on arithmetic statements to produce more efficient output code. A CASE statement is in the process of being introduced. A "peep-hole" optimizer examines generated code locally to remove redundant transfers of control or data.

The MDL subsystems facility was transferred to TENEX (Black). The facility allows application programs written in MDL to appear at top level, and it handles console interaction so that the programs can provide their own heralding and initialization.

3. Application Programs

A doctoral thesis (Cutler) deals with the solution of two major problems associated with distributed control systems. The problems are these:

- a. How should centralized control functions be handled in a distributed system? Which node should act as the "centralized controller?"
- b. How can arbitrary nodes of a distributed network communicate with each other even though individual nodes of a network possess no knowledge of the network's configuration?

The thesis discusses how a graph-structured distributed network can logically transform itself into one or more tree-structured networks. Once the network has been so transformed, the node at the root of the tree can act as the "centralized controller." Communications can be relayed through this master node, allowing arbitrary communications without routing tables.

Vehicular traffic control is used as an example of a distributed control system. It is shown how a totally extensible distributed traffic-control system can perform the same functions as a widely-used centralized system, while retaining all of the advantages of being a distributed system.

E. ENGLISH PARSER

EPARSE is an English parser, an English sentence analyzer. It will be used in both the Concept Extractor project and the Morse-code project. EPARSE goes beyond single-word disambiguation but does not attempt to go as far toward deep "understanding" of sentences as some of the parsers currently being developed in the field of artificial intelligence. The reason for limiting the effort to understand sentences deeply is that EPARSE must operate on unconstrained or very lightly constrained English, whereas, in research aimed at deep understanding, it is possible to constrain the universe of discourse as much as necessary--to the "blocks world", for example, or to children's stories.

EPARSE is actually to serve two purposes (for now). First, EPARSE is soon to be incorporated into the EXCON (EXtraction of key CONcepts) system for automatically indexing abstracts of computer programs. In this capacity EPARSE will facilitate the extraction of more appropriate index concepts rather than just keywords. The result should be both a higher fraction of retrieved abstracts relevant to the request ("precision") and a higher fraction of relevant abstracts actually retrieved ("recall"). For example, if the program abstract contained the sentence:

"This program relocates data items in memory."

and the search request was

"Find all programs for moving strings."

then both "relocates data item" and "move string" would have been modified to a common internal concept best represented, perhaps, as "move data." Thus a match of these items increases recall. Precision would be increased by being able to disambiguate among multiple meanings of a word such as "list." Thus if the sentence is about file-directory output, then "list" in the sense of "show" could be chosen as the key concept rather than, say, "list" in the sense of "sequence."

The second purpose of the parser is to assist the Morse-code project by determining which of several possible transcriptions is most English-like. An analysis of 29 samples of faulty transcriptions of hand-sent Morse code indicates that EPARSE should detect most of the faulty transcriptions and provide useful information back to the transcriber as to the locations of the faults. (The term "faulty transcription" refers to an output of the transcriber which may be incorrect due to faulty sending or incorrect transcription.) Frequently it will give a choice of fault locations. Some examples of easily detected, faulty transcriptions of portions of the Declaration of Independence follow (square brackets [] indicate where transcriber suspects problems, and angle brackets <> indicate words the transcriber selected from its dictionary in its effort to correct a mark error):

1. ... instituted among men <DERIVE> [<NO>] their just powers ...
2. ... to abolish it and to institute <DOG> over [<IN>] <MEN> laying its ...
3. ... that toe secure these rights governments are ...
4. ... that to <SENSED> the rights ...
5. ... to institute new [<MEMBER>] <BENT> laying its ...

EPARSE can detect a fault in the first example by noting that the noun group is overspecified: "no their just powers." The second example has "DOG" as a complete noun group with no adjective, article, specifier or other feature which would allow it. (Our dictionary contains noun-type information; in contrast to "dog", mass-nouns like "water" and abstract-nouns like "love" do not require such specification in the sentence.) The third example not only violates number agreement ("toe secure" instead of "toe secures") but also violates the semantic type of agent required for "secure". (Semantic type information is currently being added to the dictionary [Pollack].)

At present the parser is able to "get through" most sentences given it, despite our propensity to give it "weird" test sentences. When given faulty, difficult or ambiguous

sentences, it always is able to achieve at least partial results. For our purposes, concept extraction and Morse transcription, partial results, rather than understanding, are usually sufficient. Currently the weakest part of EPARSE is the processing of conjunctions.

The output from that part of EPARSE called the parser consists mostly of a collection of "groups"--the noun-groups, verb-groups, etc., within the sentence. These groups are presently interconnected only to the extent determinable by syntactic and positional information. Work has just begun on a case-frame module to further interconnect the groups using semantic information from the dictionary.

An unusual feature of EPARSE is its tolerance. It is tolerant of input which contains sentence fragments, cryptic abbreviations, unknown words, misspelled words, run-ons, jargon, ambiguous classifier constructions, metaphors and analogies, out-of-context references, comma splices, arbitrary usage of punctuation or capitalization, and so on. EPARSE was designed to keep running despite such input. This is consistent with the requirements of both concept extraction and the Morse-code project. In the former, programmer-written abstracts sometimes take liberties with the English language. In the latter, brevity and sentence fragments are the rule.

The MNEME relational data management system used for the dictionary saw three improvements this year. An ability to remove items from a data base was added (Westcott, Banks); a file salvager for cleaning up a data file was written; and password control of access to data files was added.

REFERENCES

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

1. Guditz, Ralph. "Computer Recognition of Hand-Sent Morse Code Using Properties of Natural Language." unpublished S. M. thesis, M.I.T., Department of Electrical Engineering and Computer Science, May 1975.
2. Selfridge, O. G.; Eisenstadt, B. M.; Gold, B.; Nelson, D. M.; and Pitcher, T. S. MAUDE. M.I.T., Lincoln Laboratory, Group Report No. 34-57, Lexington, Mass., 1959.
3. Poehler, Paul. "Computer Recognition of Hand-sent Morse Code." unpublished S. M. thesis, M.I.T., Department of Electrical Engineering and Computer Science, 1968.
4. American Radio Relay League. The Radio Amateur's Operating Manual. ARRL Publication #24, Newington, Conn., 1972.
5. Heeger, James J. "Morse Code Network Control Station Model." M.I.T., UROP Project Report, December 1975.
6. Dill, David L. "Analyzing Morse Code Headers." M.I.T., UROP Project Report, December 1975.
7. Broos, Michael. IRS -- MDL's Information Retrieval System. SYS.11.17, April 1975.
8. Haverty, Jack; Henderson, Austin; and Oestreicher, Don. "Proposed Specification of Inter-Site Message Protocol." unpublished proposal presented to ARPA Message Services Committee, July 8, 1975.
9. Broos, M.; Haverty, J.; and Vezza, A. "Message Services Protocol Proposal." unpublished proposal presented to ARPA Message Services Committee, December 4, 1975.
10. Salton, Gerald. Dynamic Information and Library Processing. Englewood Cliffs, N. J.: Prentice-Hall, 1975.
11. Fillmore, Charles J. "The Case for Case," Universals in Linguistic Theory. Bach and Harms, eds., New York: Holt, Rinehart and Winston, 1968.
12. Bruce, Bertram. "Case Systems for Natural Language." Artificial Intelligence, Vol. 6. No. 4 (1975), 327-360.

13. M.I.T., Laboratory for Computer Science. Progress Report XII. LCS/PR-XII,
Cambridge, Massachusetts, 1975.

PUBLICATIONS

115

PUBLICATIONS

LABORATORY FOR COMPUTER SCIENCE

PUBLICATIONS

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

TMs 1-9 were never issued

PRECEDING PAGE BLANK

PUBLICATIONS

118

PUBLICATIONS

- *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
- *TM-16** Bruere-Dawson, Gerard
Pseudo-Random Sequences
(Based on S.M. Thesis, EE Dept.,
June 1970)
October 1970
- *TM-17** Goodman, Leonard I.
Complexity Measures for Programming
Languages (Based on S.M. Thesis, EE Dept.,
September 1971)
September 1971
- *TM-18** Reprinted as TR-85
- *TM-19** Fenichel, Robert R.
A New List-Tracing Algorithm
October 1970
- *TM-20** Jones, Thomas L.
A Computer Model of Simple Forms
of Learning (Based on Ph.D. Thesis,
EE Dept., September 1970)
January 1971
- *TM-21** Goldstein, Robert, C.
The Substantive Use of Computers
for Intellectual Activities
April 1971

AD 720-761

AD 713-852

AD 729-011

AD 714-522

AD 720-337

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

PUBLICATIONS

120

PUBLICATIONS

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

AD 747-250

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

AD 749-365

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

AD 757-787

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

AD 759-999

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

AD 760-000

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

PB 221-742

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

AD 770-565

PUBLICATIONS

121

PUBLICATIONS

TM-36 Geiger, Steven P.

A User's Guide to the Macro Control Language
December 1973

AD 771-435

*TM-37 Schoenhage, A.

Real-Time Simulation of Multidimensional
Turing Machines by Storage Modification
Machines
December 1973

PB 226-103/AS

*TM-38 Meyer, Albert R.

Weak Monadic Second Order Theory of
Successor is not Elementary-Recursive
December 1973

PB 226-514/AS

TM-39 Meyer, Albert R.

Discrete Computation: Theory and Open
Problems
January 1974

PB 226-836/AS

TM-40 Paterson, Michael S., Michael J. Fischer
and Albert R. Meyer

An Improved Overlap Argument for On-Line
Multiplication
January 1974

AD 773-137

TM-41 Fischer, Michael J., and Michael S. Paterson
String-Matching and Other Products
January 1974

AD 773-138

TM-42 Rackoff, Charles

On the Complexity of the Theories of Weak
Direct Products
January 1974

PB 228-459/AS

PUBLICATIONS

122

PUBLICATIONS

TM-43 Fischer, Michael J., and Michael O. Rabin
Super-Exponential Complexity of Presburger
Arithmetic
February 1974

AD 775-004

TM-44 Pless, Vera
Symmetry Codes and their Invariant Subcodes
May 1974

AD 780-243

***TM-45** Fischer, Michael J., and Larry J. Stockmeyer
Fast On-Line Integer Multiplication
May 1974

AD 779-889

***TM-46** Kedem, Zvi M.
Combining Dimensionality and Rate of Growth
Arguments for Establishing Lower Bounds
on the Number of Multiplications
June 1974

PB 232-969/AS

TM-47 Pless, Vera
Mathematical Foundations of Flip-Flops
June 1974

AD 780-901

TM-48 Kedem, Zvi M.
The Reduction Method for Establishing
Lower Bounds on the Number of Additions
June 1974

PB 233-538/AS

TM-49 Pless, Vera
Complete Classification of (24,12) and (22,11)
Self-Dual Codes
June 1974

AD 781-335

PUBLICATIONS

123

PUBLICATIONS

TM-50 Benedict, G. Gordon
An Enciphering Module for Multics
S.B. Thesis, EE Dept.
July 1974

AD 782-658

TM-51 Aiello, Jack M.
An Investigation of Current Language Support for
the Data Requirements of Structured Programming
S.M. & E.E. Theses, EE Dept.
September 1974

PB 236-815/AS

TM-52 Lind, John C.
Computing in Logarithmic Space
September 1974

PB 236-167/AS

TM-53 Bengelloun, Safwan A.
MDC-Programmer: A Muddle-to Datalanguage
Translator for Information Retrieval
S.B. Thesis, EE Dept.
October 1974

AD 786-754

***TM-54** Meyer, Albert. R.
The Inherent Computation Complexity of Theories
of Ordered Sets: A Brief Survey
October 1974

PB 237-200/AS

TM-55 Hsieh, Wen N., Larry H. Harper and John E. Savage
A Class of Boolean Functions with Linear
Combinatorial Complexity
October 1974

PB 237-206/AS

TM-56 Gorry, G. Anthony
Research on Expert Systems
December 1974

TM-57 Levin, Michael
On Bateson's Logical Levels of Learning
February 1975

TM-58 Qualitz, Joseph E.
Decidability of Equivalence for a Class
of Data Flow Schemas
March 1975

PB 237-033/AS

*TM-59 Hack, Michel
Decision Problems for Petri Nets and Vector
Addition Systems
March 1975

PB 231-916/AS

TM-60 Weiss, Randell B.
CAMAC: Group Manipulation System
March 1975

PB 240-495/AS

TM-61 Dennis, Jack B.
First Version of a Data Flow Procedure Language
May 1975

TM-62 Patil, Suhas S.
An Asynchronous Logic Array
May 1975

TM-63 Pless, Vera
Encryption Schemes for Computer Confidentiality
May 1975

AD A010-217

*TM-64 Weiss, Randell B.
Finding Isomorph Classes for Combinatorial Structures
S.M. Thesis, EE Dept.
June 1975

TM-65 Fischer, Michael J.
The Complexity Negation-Limited Networks -
A Brief Survey
June 1975

- *TM-66 Leung, Clement
Formal Properties of Well-Formed Data
Flow Schemas
S.B., S.M. & E.E. Theses, EE Dept.
June 1975
- *TM-67 Cardoza, Edward E.
Computational Complexity of the Word Problem
for Commutative Semigroups
S.M. Thesis, EE & CS Dept.
October 1975
- TM-68 Weng, Kung-Song
Stream-Oriented Computation in Recursive Data Flow Schemas
S.M. Thesis, EE & CS Dept.
October 1975
- *TM-69 Bayer, Paul J.
Improved Bounds on the Costs of Optimal and
Balanced Binary Search Trees
S.M. Thesis, EE & CS Dept.
November 1975

TECHNICAL REPORTS

- *TR-1 Bobrow, Daniel G.
Natural Language Input for a Computer
Problem Solving System,
Ph.D. Thesis, Math. Dept.
September 1964
AD 604-730
- *TR-2 Raphael, Bertram
SIR: A Computer Program for Semantic
Information Retrieval,
Ph.D. Thesis, Math. Dept.
June 1964
AD 608-499
- *TR-3 Corbato, Fernando J.
System Requirements for Multiple-Access,
Time-Shared Computers
May 1964
AD 608-501
- *TR-4 Ross, Douglas T., and Clarence G. Feldman
Verbal and Graphical Language for the
AED System: A Progress Report
May 1964
AD 604-678
- *TR-6 Biggs, John M., and Robert D. Logcher
STRESS: A Problem-Oriented Language
for Structural Engineering
May 1964
AD 604-679

TRs 5, 9, 10, 15 were never issued

PUBLICATIONS

127

PUBLICATIONS

- *TR-7 Weizenbaum, Joseph
OPL-1: An Open Ended Programming
System within CTSS
April 1964
AD 604-680
- TR-8 Greenberger, Martin
The OPS-1 Manual
May 1964
AD 604-681
- *TR-11 Dennis, Jack B.
Program Structure in a Multi-Access
Computer
May 1964
AD 608-500
- TR-12 Fano, Robert M.
The MAC System: A Progress Report
October 1964
AD 609-296
- *TR-13 Greenberger, Martin
A New Methodology for Computer Simulation
October 1964
AD 609-288
- TR-14 Roos, Daniel
Use of CTSS in a Teaching Environment
November 1964
AD 661-807
- TR-16 Saltzer, Jerome H.
CTSS Technical Notes
March 1965
AD 612-702
- *TR-17 Samuel, Arthur L.
Time-Sharing on a Multiconsole Computer
March 1965
AD 462-158

***TR-18 Scherr, Allan Lee**

An Analysis of Time-Shared Computer Systems,
Ph.D. Thesis, EE Dept.
June 1965

AD 470-715**TR-19 Russo, Francis John**

A Heuristic Approach to Alternate Routing in a Job Shop,
S.B. & S.M. Theses, 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

PUBLICATIONS

129

PUBLICATIONS

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

PUBLICATIONS

130

PUBLICATIONS

- 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

131

PUBLICATIONS

- *TR-37 Guzman-Arenas, Adolfo
Some Aspects of Pattern Recognition
by Computer,
S.M. Thesis, EE Dept.
February 1967
- TR-38 Rosenberg, Ronald C., Daniel W. Kennedy
and Roger A. Humphrey
A Low-Cost Output Terminal For Time-
Shared Computers
March 1967
- *TR-39 Forte, Allen
Syntax-Based Analytic Reading of
Musical Scores
April 1967
- TR-40 Miller, James R.
On-Line Analysis for Social Scientists
May 1967
- *TR-41 Coons, Steven A.
Surfaces for Computer-Aided Design
of Space Forms
June 1967
- TR-42 Liu, Chung L., Gabriel D. Chang
and Richard E. Marks
Design and Implementation of a Table-
Driven Compiler System
July 1967
- TR-43 Wilde, Daniel U.
Program Analysis by Digital Computer,
Ph.D. Thesis, EE Dept.
August 1967

AD 656-041

AD 662-027

AD 661-806

AD 668-009

AD 663-504

AD 668-960

AD 662-224

PUBLICATIONS

132

PUBLICATIONS

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

AD 662-665

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

AD 663-502

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

AD 663-503

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

AD 662-666

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

AD 662-225

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

AD 667-602

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

AD 675-554

- *TR-51** Charniak, Eugene
CARPS, A Program which Solves
Calculus Word Problems,
S.M. Thesis, EE Dept.
July 1968
AD 673-670
- TR-52** Deitel, Harvey M.
Absentee Computations in a Multiple-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, Daniel E., Robert H. Stotz, Douglas T. Ross
and John E. Ward (ESL-R-356)
An Integrated Hardware-Software System
for Computer Graphics in Time-Sharing
December 1968
AD 685-202

PUBLICATIONS

134

PUBLICATIONS

- *TR-57 Morris, James H.**
Lambda-Calculus Models of Programming
Languages,
Ph.D. Thesis, Sloan School
December 1968

AD 683-394

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

AD 686-988

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

AD 692-200

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

AD 689-305

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

AD 690-887

TR-62 Tillman, Coyt C., Jr. (ESL-R-395)

EPS: An Interactive System for
Solving Elliptic Boundary-Value
Problems with Facilities for Data
Manipulation and General-Purpose
Computation
June 1969

AD 692-462**TR-63 Brackett, John W., Michael Hammer and Daniel
E. Thornhill**

Case Study in Interactive Graphics
Programming: A Circuit Drawing
and Editing Program for Use with
a Storage-Tube Display Terminal
October 1969

AD 699-930***TR-64 Rodriguez, Jorge E. (ESL-R-398)**

A Graph Model for Parallel Computations,
Sc.D. Thesis, EE Dept.
September 1969

AD 697-759***TR-65 DeRemer, Franklin L.**

Practical Translators for LR(k)
Languages,
Ph.D. Thesis, EE Dept.
October 1969

AD 699-501***TR-66 Beyer, Wendell T.**

Recognition of Topological Invariants
by Iterative Arrays,
Ph.D. Thesis, Math. Dept.
October 1969

AD 699-502***TR-67 Vanderbilt, Dean H.**

Controlled Information Sharing in
a Computer Utility,
Ph.D. Thesis, EE Dept.
October 1969

AD 699-503

*TR-68 Selwyn, Lee L.

Economies of Scale in Computer Use:
Initial Tests and Implications for
The Computer Utility,
Ph.D. Thesis, Sloan School
June 1970

AD 710-011

*TR-69 Gertz, Jeffrey L.

Hierarchical Associative Memories
for Parallel Computation,
Ph.D. Thesis, EE Dept.
June 1970

AD 711-091

*TR-70 Fillat, Andrew I., and Leslie A. Kraning

Generalized Organization of Large
Data-Bases: A Set-Theoretic
Approach to Relations,
S.B. & S.M. Theses, EE Dept.
June 1970

AD 711-060

*TR-71 Fiasconaro, James G.

A Computer-Controlled Graphical
Display Processor,
S.M. Thesis, EE Dept.
June 1970

AD 710-479

TR-72 Patil, Suhas S.

Coordination of Asynchronous Events,
Sc.D. Thesis, EE Dept.
June 1970

AD 711-763

*TR-73 Griffith, Arnold K.

Computer Recognition of Prismatic
Solids,
Ph.D. Thesis, Math. Dept.
August 1970

AD 712-069

- TR-74 Edelberg, Murray
Integral Convex Polyhedra and an
Approach to Integralization,
Ph.D. Thesis, EE Dept.
August 1970
- TR-75 Hebalkar, Prakash G.
Deadlock-Free Sharing of Resources
in Asynchronous Systems,
Sc.D. Thesis, EE Dept.
September 1970
- *TR-76 Winston, Patrick H.
Learning Structural Descriptions
from Examples,
Ph.D. Thesis, EE Dept.
September 1970
- TR-77 Haggerty, Joseph P.
Complexity Measures for Language
Recognition by Canonic Systems,
S.M. Thesis, EE Dept.
October 1970
- *TR-78 Madnick, Stuart E.
Design Strategies for File Systems,
S.M. Thesis, EE Dept. & Sloan School
October 1970
- 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 712-070

AD 713-139

AD 713-988

AD 715-134

AD 714-269

AD 717-336

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

AD 717-857

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

AD 717-951

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

AD 723-647

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

AD 720-098

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

AD 721-399

TR-85 Miller, Perry L.
Automatic Creation of a Code Generator
from a Machine Description,
E.E. Thesis, 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.B. & S.M. Theses, 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

PUBLICATIONS

140

PUBLICATIONS

- TR-92 Wang, Paul S.
Evaluation of Definite Integrals
by Symbolic Manipulation
Ph.D. Thesis, Math. Dept.
October 1971
AD 732-005
- TR-93 Greif, Irene Gloria
Induction in Proofs about Programs,
S.M. Thesis, EE Dept.
February 1972
AD 737-701
- TR-94 Hack, Michel Henri Theodore
Analysis of Production Schemata
by Petri Nets,
S.M. Thesis, EE Dept.
February 1972
AD 740-320
- TR-95 Fateman, Richard J.
Essays in Algebraic Simplification
(A revision of a Harvard Ph.D. Thesis)
April 1972
AD 740-132
- TR-96 Manning, Frank
Autonomous, Synchronous Counters Constructed Only of
J-K Flip-Flops,
S.M. Thesis, EE Dept.
May 1972
AD 744-030
- TR-97 Vilfan, Bostjan
The Complexity of Finite Functions
Ph.D. Thesis, EE Dept.
March 1972
AD 739-678
- TR-98 Stockmeyer, Larry Joseph
Bounds on Polynomial Evaluation Algorithms
S.M. Thesis, EE Dept.
April 1972
AD 740-328

PUBLICATIONS

141

PUBLICATIONS

- TR-99 Lynch, Nancy Ann
Relativization of the Theory of Computational Complexity
Ph.D. Thesis, Math. Dept.
June 1972
AD 744-032
- TR-100 Mandl, Robert
Further Results on Hierarchies of Canonic Systems
S.M. Thesis, EE Dept.
June 1972
AD 744-206
- TR-101 Dennis, Jack B.
On the Design and Specification of a Common Base Language
June 1972
AD 744-207
- TR-102 Hossley, Robert F.
Finite Tree Automata and ω -Automata
S.M. Thesis, EE Dept.
September 1972
AD 749-367
- *TR-103 Sekino, Akira
Performance Evaluation of Multiprogrammed Time-Shared
Computer Systems
Ph.D Thesis, EE Dept.
September 1972
AD 749-949
- TR-104 Schroeder, Michael D.
Cooperation of Mutually Suspicious Subsystems
in a Computer Utility
Ph.D. Thesis, EE Dept.
September 1972
AD 750-173
- TR-105 Smith, Burton J.
An Analysis of Sorting Networks
Sc.D. Thesis, EE Dept.
October 1972
AD 751-614

PUBLICATIONS

142

PUBLICATIONS

TR-106 Rackoff, Charles W.

The Emptiness and Complementation Problems
for Automata on Infinite Trees

S.M. Thesis, EE Dept.

January 1973

AD 756-248

TR-107 Madnick, Stuart E.

Storage Hierarchy Systems

Ph.D. Thesis, EE Dept.

April 1973

AD 760-001

TR-108 Wand, Mitchell

Mathematical Foundations of Formal Language Theory

Ph.D. Thesis, Math. Dept.

December 1973

TR-109 Johnson, David S.

Near-Optimal Bin Packing Algorithms

Ph.D. Thesis, Math. Dept.

June 1973

PB 222-090

TR-110 Moll, Robert

Complexity Classes of Recursive Functions

Ph.D. Thesis, Math. Dept.

June 1973

AD 767-730

TR-111 Linderman, John P.

Productivity in Parallel Computation Schemata

Ph.D. Thesis, EE Dept.

December 1973

PB 226-159/AS

TR-112 Hawryszkiewicz, Igor T.

Semantics of Data Base Systems

Ph.D. Thesis, EE Dept.

December 1973

PB 226-061/AS

PUBLICATIONS

143

PUBLICATIONS

- TR-113 Herrmann, Paul P.
On Reducibility Among Combinatorial Problems
S.M. Thesis, Math. Dept.
December 1973
PB 226-157/AS
- TR-114 Metcalfe, Robert M.
Packet Communication
Ph.D. Thesis, Applied Math., Harvard University
December 1973
AD 771-430
- TR-115 Rotenberg, Leo
Making Computers Keep Secrets
Ph.D Thesis, EE Dept.
February 1974
PB 229-352/AS
- TR-116 Stern, Jerry A.
Backup and Recovery of On-Line Information
in a Computer Utility
S.M. & E.E. Theses, EE Dept.
January 1974
AD 774-141
- TR-117 Clark, David D.
An Input/Output Architecture for
Virtual Memory Computer Systems
Ph.D. Thesis, EE Dept.
January 1974
AD 774-738
- TR-118 Briabrin, Victor
An Abstract Model of a Research Institute:
Simple Automatic Programming Approach
March 1974
PB 231-505/AS
- TR-119 Hammer, Michael M.
A New Grammatical Transformation into
Deterministic Top-Down Form
Ph.D. Thesis, EE Dept.
February 1974
AD 775-545

PUBLICATIONS

144

PUBLICATIONS

- TR-120 Ramchandani, Chander**
Analysis of Asynchronous Concurrent Systems
by Timed Petri Nets
Ph.D. Thesis, EE Dept.
February 1974
AD 775-618
- TR-121 Yao, Foong F.**
On Lower Bounds for Selection Problems
Ph.D. Thesis, Math. Dept.
March 1974
PB 230-950/AS
- TR-122 Scherf, John A.**
Computer and Data Security: A Comprehensive
Annotated Bibliography
S.M. Thesis, Sloan School
January 1974
AD 775-546
- TR-123 Introduction to Multics**
February 1974
AD 918-562
- TR-124 Laventhal, Mark S.**
Verification of Programs Operating on Structured Data
S.B. & S.M. Theses, EE Dept.
March 1974
PB 231-365/AS
- TR-125 Mark, William S.**
A Model-Debugging System
S.B. & S.M. Theses, EE Dept.
April 1974
AD 778-688
- TR-126 Altman, Vernon E.**
A Language Implementation System
S.B. & S.M. Theses, Sloan School
May 1974
AD 780-672

TR-127 Greenberg, Bernard S.
An Experimental Analysis of Program Reference
Patterns in the Multics Virtual Memory
S.M. Thesis, EE Dept.
May 1974

AD 780-407

TR-128 Frankston, Robert M.
The Computer Utility as a Marketplace for Computer
Services
S.M. & E.E. Theses, EE Dept.
May 1974

AD 780-436

TR-129 Weissberg, Richard W.
Using Interactive Graphics in Simulating the Hospital
Emergency Room
S.M. Thesis, EE Dept.
May 1974

AD 780-437

TR-130 Ruth, Gregory R.
Analysis of Algorithm Implementations
Ph.D. Thesis, EE Dept.
May 1974

AD 780-408

TR-131 Levin, Michael
Mathematical Logic for Computer Scientists
June 1974

TR-132 Janson, Philippe A.
Removing the Dynamic Linker from the Security
Kernel of a Computing Utility
S.M. Thesis, EE Dept.
June 1974

AD 781-305

TR-133 Stockmeyer, Larry J.
The Complexity of Decision Problems in
Automata Theory and Logic
Ph.D. Thesis, EE Dept.
July 1974

PB 235-283/AS

PUBLICATIONS

146

PUBLICATIONS

TR-134 Ellis, David J.

**Semantics of Data Structures and References
S.M. & E.E. Theses, EE Dept.
August 1974**

PB 236-594/AS

TR-135 Pfister, Gregory F.

**The Computer Control of Changing Pictures
Ph.D. Thesis, EE Dept.
September 1974**

AD 787-795

TR-136 Ward, Stephen A.

**Functional Domains of Applicative Languages
Ph.D. Thesis, EE Dept.
September 1974**

AD 787-796

TR-137 Seiferas, Joel I.

**Nondeterministic Time and Space Complexity
Classes
Ph.D Thesis, Math. Dept.
September 1974**

PB 236-777/AS

TR-138 Yun, David Y. Y.

**The Hensel Lemma in Algebraic Manipulation
Ph.D. Thesis, Math. Dept.
November 1974**

AD A002-737

TR-139 Ferrante, Jeanne

**Some Upper and Lower Bounds on Decision
Procedures in Logic
Ph.D. Thesis, Math. Dept.
November 1974**

PB 238-121/AS

TR-140 Redell, David D.

**Naming and Protection in Extendible
Operating Systems
Ph.D. Thesis, EE Dept.
November 1974**

AD A001-721

PUBLICATIONS

147

PUBLICATIONS

TR-141 Richards, Martin, A. Evans and R. Mabee
The BCPL Reference Manual
December 1974

AD A003-599

TR-142 Brown, Gretchen P.
Some Problems in German to English
Machine Translation
S.M. & E.E. Theses, EE Dept.
December 1974

AD A003-002

TR-143 Silverman, Howard
A Digitalis Therapy Advisor
S.M. Thesis, EE Dept.
January 1975

TR-144 Rackoff, Charles
The Computational Complexity of Some
Logical Theories
Ph.D. Thesis, EE Dept.
February 1975

*TR-145 Henderson, D. Austin
The Binding Model: A Semantic Base
for Modular Programming Systems
Ph.D. Thesis, EE Dept.
February 1975

AD A006-961

TR-146 Malhotra, Ashok
Design Criteria for a Knowledge-Based
English Language System for Management:
An Experimental Analysis
Ph.D. Thesis, EE Dept.
February 1975

TR-147 Van De Vanter, Michael L.
A Formalization and Correctness Proof
of the CGOL Language System
S.M. Thesis, EE Dept.
March 1975

PUBLICATIONS

148

PUBLICATIONS

TR-148 Johnson, Jerry
Program Restructuring for Virtual Memory Systems
Ph.D. Thesis, EE Dept.
March 1975

AD A009-218

TR-149 Snyder, Alan
A Portable Compiler for the Language C
S.B. & S.M. Theses, EE Dept.
May 1975

AD A010-218

TR-150 Rumbaugh, James E.
A Parallel Asynchronous Computer Architecture
for Data Flow Programs
Ph.D. Thesis, EE Dept.
May 1975

AD A010-918

TR-151 Manning, Frank B.
Automatic Test, Configuration, and Repair
of Cellular Arrays
Ph.D. Thesis, EE Dept.
June 1975

AD A012-822

TR-152 Qualitz, Joseph E.
Equivalence Problems for Monadic Schemas
Ph.D. Thesis, EE Dept.
June 1975

AD A012-823

TR-153 Miller, Peter B.
Strategy Selection in Medical Diagnosis
S.M. Thesis, EE & CS Dept.
September 1975

TR-154 Greif, Irene
Semantics of Communicating Parallel Processes
Ph.D. Thesis, EE & CS Dept.
September 1975

AD A016-302

PUBLICATIONS

149

PUBLICATIONS

- TR-155 Kahn, Kenneth M.**
Mechanization of Temporal Knowledge
S.M. Thesis, EE & CS Dept.
September 1975
- TR-156 Bratt, Richard G.**
Minimizing the Naming Facilities Requiring
Protection in a Computer Utility
S.M. Thesis, EE & CS Dept.
September 1975
- TR-157 Meldman, Jeffrey A.**
A Preliminary Study in Computer-Aided Legal Analysis
Ph.D. Thesis, EE & CS Dept.
November 1975

AD A018-997

PUBLICATIONS

150

PUBLICATIONS

PROGRESS REPORTS

***Project MAC Progress Report I
to July 1964**

AD 465-088

***Project MAC Progress Report II
July 1964-July 1965**

AD 629-494

***Project MAC Progress Report III
July 1965-July 1966**

AD 648-346

**Project MAC Progress Report IV
July 1966-July 1967**

AD 681-342

**Project MAC Progress Report V
July 1967-July 1968**

AD 687-770

**Project MAC Progress Report VI
July 1968-July 1969**

AD 705-434

**Project MAC Progress Report VII
July 1969-July 1970**

AD 732-767

**Project MAC Progress Report VIII
July 1970-July 1971**

AD 735-148

***Project MAC Progress Report IX
July 1971-July 1972**

AD 756-689

**Project MAC Progress Report X
July 1972-July 1973**

AD 771-428

PUBLICATIONS

151

PUBLICATIONS

**Project MAC Progress Report XI
July 1973-July 1974**

AD A004-966

**Laboratory for Computer Science Progress Report XII
July 1974-July 1975**

AD A024-527

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

* Out of Print reports may be obtained from NTIS if the AD number is supplied (see above). Out of Print reports without an AD or PB number are unobtainable.

OFFICIAL DISTRIBUTION LIST

Defense Documentation Center
Cameron Station
Alexandria, VA 22314
12 copies

Office of Naval Research
Information Systems Program
Code 437
Arlington, VA 22217
2 copies

Office of Naval Research
Branch Office/Boston
495 Summer Street
Boston, MA 02210
1 copy

Office of Naval Research
Branch Office/Chicago
536 South Clark Street
Chicago, IL 60605
1 copy

Office of Naval Research
Branch Office/Pasadena
1030 East Green Street
Pasadena, CA 91106
1 copy

New York Area Office
715 Broadway - 5th floor
New York, N. Y. 10003
1 copy

Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D. C. 20375
6 copies

Assistant Chief for Technology
Office of Naval Research
Code 200
Arlington, VA 22217
1 copy

Office of Naval Research
Code 455
Arlington, VA 22217
1 copy

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
(Code RD-1)
Washington, D. C. 20380
1 copy

Office of Naval Research
Code 458
Arlington, VA 22217
1 copy

Naval Electronics Lab Center
Advanced Software Technology
Division - Code 5200
San Diego, CA 92152
1 copy

Mr. E. H. Gleissner
Naval Ship Research & Development Center
Computation & Math Department
Bethesda, MD 20084
1 copy

Captain Grace M. Hopper
NAICOM/MIS Planning Branch
(OP-916D)
Office of Chief of Naval Operations
Washington, D. C. 20350
1 copy

Mr. Kin B. Thompson
Technical Director
Information Systems Division
(OP-91T)
Office of Chief of Naval Operations
Washington, D. C. 20350
1 copy

Captain Richard L. Martin, USN
Commanding Officer
USS Francis Marion (LPA-249)
FPO New York, N. Y. 09501
1 copy