2

Massachusetts Institute of Technology

Laboratory for Computer Science Progress Report 22

July 1984-
June 1985

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/PR-22	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION MIT Lab for Computer Science	6b. OFFICE SYMBOL *(If applicable)*
7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Dept. of Navy	
6c. ADDRESS *(City, State, and ZIP Code)* 545 Technology Square Cambridge, MA 02139	7b. ADDRESS *(City, State, and ZIP Code)* Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD	8b. OFFICE SYMBOL *(If applicable)*
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS *(City, State, and ZIP Code)* 1400 Wilson Blvd. Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*
MIT Laboratory for Computer Science Progress Report - 22

12. PERSONAL AUTHOR(S)
Dertouzos, M.L.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical /Progress	FROM 7/84 TO 6/85	June 1985	266

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Computer Architecture, Computer Science, Computer Systems, Electrical Engineering, Networks, Theory of Computers, Programming Languages

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*
Annual Report of Progress made at the MIT Laboratory for Computer Science Under Contracts:

a.) N00014-83k - 0125, Darpa Order 5602/2095
b.) N00014-84k - 0099, Darpa Order 4920

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Carol Nicolora	22b. TELEPHONE *(Include Area Code)* (617) 253-5894
22c. OFFICE SYMBOL	

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

*U.S. Government Printing Office: 1985—507-047

Unclassified

Massachusetts Institute of Technology

Laboratory for Computer Science Progress Report 22

July 1984-
June 1985

The work reported herein was carried out within the Laboratory for Computer Science, an MIT interdepartmental laboratory. During 1984-85 the principal financial support of the Laboratory has come from the Defense Advanced Research Projects Agency (DARPA). DARPA has been instrumental in supporting most of our research over the last 22 years and is gratefully acknowledged here. Our overall support has come from the following organizations:

- Defense Advanced Research Projects Agency;
- Department of Energy;
- National Institutes of Health, under National Library of Medicine;
- National Science Foundation;
- Office of Naval Research;
- United States Air Force;
- United States Army Research Office
- MIT controlled IBM funds under an IBM/MIT joint study contract;

Other support of a generally smaller level has come from Apple, Honeywell, Harris Corporation, NASA, and Siemens.

(A)

TABLE OF CONTENTS

[illegible]

B

(KR)

ADMINISTRATION

Academic Staff

M. Dertouzos	Director
M. Rivest	Associate Director

Administrative Staff

P. Anderegg	Assistant Administrative Officer
G. Brown	Facilities Officer
A. Chow	Fiscal Officer
J. Hynes	Administrative Officer
M. Jones	Fiscal Officer

Support Staff

L. Cavallaro	B. Pierce
R. Cinq-Mars	E. Profirio
R. Donahue	M. Sensale
A. Kekejian	D. Simmons
T. LoDuca	C. Stevens
J. Mullins	P. Vancini
T. Novak	

1

INTRODUCTION

The MIT Laboratory for Computer Science (LCS) is an interdepartmental laboratory whose principal goal is research in computer science and engineering.

Founded in 1963 as Project MAC (for Multiple Access Computer and Machine Aided Cognition), the Laboratory developed the Compatible Time Sharing System (CTSS), one of the first time shared systems in the world, and Multics -- an improved time shared system that introduced several new concepts. These two major developments stimulated research activities in the application of on-line computing to such diverse disciplines as engineering, architecture, mathematics, biology, medicine, library science and management. Since that time, the Laboratory's pursuits expanded, leading to pioneering research in Expert Systems, Computer Networks, and Public Cryptography. Today, the Laboratory's research spans a broad front of activities, grouped in four major areas.

The first such area entitled Knowledge Based Systems, involves making programs more intelligent by capturing, representing, and using knowledge which is specific to the problem domain. Examples are the use of expert medical knowledge for assistance in diagnosis carried out by the Clinical Decision Making Group; and the use of solid-state circuit design knowledge for an expert VLSI (very large scale integration) design system by the VLSI Design Project.

Research in the second and largest area entitled Machines, Languages, and Systems strives to discover and understand computing systems at both the hardware and software levels that open new application areas and/or effect sizable improvements in their ease of utilization and cost effectiveness. New research in this area includes the architecture of very large multiprocessor machines (which tackle a single task, e.g., speech understanding or weather analysis) by the Computation Structures, Functional Languages and Architectures, and Real Time Systems Research Groups. Continuing research includes the analysis and synthesis of languages and operating systems for use in large geographically distributed systems by the Programming Methodology and Real Time Systems Groups. Extended networks for such distributed environments as well as distributed file servers are studied by the Distributed Computer Systems Group. Finally, a key application involving the matching of news and other community information to individual needs, is pursued by the Imaginative Systems Group.

over

The Laboratory's third principal area of research, entitled Theory, involves exploration and development of theoretical foundations in computer science. For example, the Theory of Computation Group strives to understand ultimate limits in space and time associated with various classes of algorithms; the semantics of programming languages from both analytical and synthetic viewpoints; the logic of programs; the utility of randomness in computation; concurrent computation and the links between mathematics and the privacy/authentication of computer-to-computer messages. Other examples of theoretical work involve the study of distributed systems by the Theory of Distributed Computer Systems Group, and the development of effective algorithms for VLSI design.

The fourth area of research entitled Computers and People, entails societal as well as technical aspects of the interrelationships between people and machines. Examples include the use of computers in the educational process by the Educational Computing Group; the use of interconnected computers for planning; as well as the societal impact of computers carried out by the Societal Implications Research Group.

During 1984-1985 the Laboratory continued its ambitious project of constructing a Multiprocessor Emulation Facility consisting of 64 interconnected Lisp Machines, whose purpose is to analyze the behavior of larger (up to several thousand machines) multiprocessor systems. This facility, funded by the newly formed Strategic Computing Program of the Defense Advanced Research Projects Agency, will enable our experimenters to try out ideas before committing their proposed architectures to silicon circuits. Another related development during this period has been the continued successful development of the MultiLisp language for multiprocessor systems by Professor Robert Halstead of the Real Time Systems Group. This language and the Emulation Facility will be used in the coming year to assess the effectiveness of multiprocessors in carrying out a variety of new applications.

Another important growth activity has been the evolution of a research and development plan for the LCS Common System. Through this system, the Laboratory aspires to interconnect heterogeneous computer resources. Current distributed computer systems entail networked, generally identical computers which intercommunicate at the relatively low level of exchanging text and other computer files. The LCS Common System will deviate from this approach in two ways: First, it will admit dissimilar computers, such as Lisp Machines and Vaxes. Second, it will permit sharing data and procedures among these different environments so that our researchers may build on each other's work, as if they were using a single time shared computer. In addition, the system will enable access by every researcher to unique laboratory resources such as the Multiprocessor Emulation Facility and the Expert VLSI Design System. We expect that implementation of this major project will begin January 1986 and will last for at least three years.

During 1984-1985 the Laboratory has continued its successful Distinguished Lecturer Series with presentations by Steven Jobs, Chairman of Apple Computer; Professor Roger Schank of Yale University; Professor Stephen Cook of the University of Toronto; Bobby Inman, President of MCC; Professor Carver Mead of the California Institute of Technology; and Kenneth Thompson, a co-developer of UNIX from AT&T Bell Laboratories. Also during the same period the Laboratory employed twenty-four undergraduates through the "Hacker Heaven" project which strives to identify promising potential researchers in computer science.

During 1985, the following members of the Laboratory were honored with awards: David K. Gifford with the ITT Career Development Chair; Charles E. Leiserson with the NSF Presidential Young Investigator Award; Barbara H. Liskov as NEC Professor of Software Technology; and Sherry Turkle as *Ms. Magazine*'s Woman of the Year.

Other changes include the appointment of Dr. Karen Sollins as Head of Computer Resources, taking over from Dr. David Clark who will now lead the Laboratory's Common System research effort. Arrivals included Assistant Professors Nikhil and Weihl, and Visiting Scientist Richard Greenblatt. Departures during the same period were Research Associate Benjamin Kuipers of the Clinical Decision Making Group to the University of Texas, Mr. Albert Vezza, Head of the Programming Technology Group who became Chairman and Chief Executive Officer of Infocom Inc., Professor J.C.R. Licklider who is retiring, and Research Associates David Lebling and Christopher Reeve who have also joined Infocom, Inc.

Our Laboratory consisted of 350 members -- 42 faculty, and academic research staff, 35 visitors and visiting faculty, 63 professional and support staff, 110 graduate and 100 undergraduate students -- organized into 16 research groups. Laboratory research during 1984-85 was funded by 14 governmental and industrial organizations, of which the Defense Advanced Research Projects Agency of the Department of Defense provided over half of the total research funds.

Technical results of our research in 1984-85 were disseminated through publications in the technical literature, through Technical Reports (TR331-TR355), and through Technical Memoranda (TM279-TM290).

50 A

CLINICAL DECISION MAKING

Academic Staff

P. Szolovits, Group Leader
R. Patil

Collaborating Investigators

M. Criscitiello, M.D., Tufts-New England Medical Center Hospital
J. Dzierzanowski, Ph.D., Dept. of Orthopaedics, Harvard Medical School
R. Friedman, M.D., University Hospital, Boston University
W. Hardy, Ph.D., University Hospital, Boston University
J. Hollenberg, M.D., Tufts-New England Medical Center Hospital
J. Kassirer, M.D., Tufts-New England Medical Center Hospital
M. Klein, M.D., University Hospital, Boston University
S. Kurzrok, M.D., Tufts-New England Medical Center Hospital
J. Lau, M.D., Boston VA Medical Center
A. Levey, M.D., Tufts-New England Medical Center
A. Moskowitz, M.D., Tufts-New England Medical Center Hospital
S. Naimi, M.D., Tufts-New England Medical Center Hospital
S. Pauker, M.D., Tufts-New England Medical Center
W. Schwartz, M.D., Tufts-New England School of Medicine
L. Widman, M.D., Case Western Reserve University Medical School

Research Staff

G. Burke	W. Long
C. Eliot	T. McNerney
B. Kuipers	

Graduate Students

H. Goldberger	T. Russ
R. Granville	E. Sachs
D. Hirsch	D. Smit
P. Koton	M. Wellman
R. Kunstaetter	A. Yeh

Undergraduate Students

J. Fearnside	K-Y. Lai
S. Ferguson	T-Y. Leung
M. Harvey	D. Pachura
C. Kim	S. Rao

Support Staff

R. Hegg

Visitors

C. Dede	O. Gascuel
D. De Roulet	I. Kohane

1. INTRODUCTION

The three main goals of the Clinical Decision Making Group are as follows:

1) *cognition* -- to improve methods of analyzing the reasoning of human clinicians when they are faced with critical decision problems involving significant risk and uncertainty,

2) *explanation* -- to develop new methods that permit computer programs to explain their conclusions by arguing from basic medical facts and principles, and

3) *decision analysis* -- to enhance and disseminate micro-computer based tools that help perform clinical decision analyses.

This project is being carried out by an interdisciplinary and multi-institutional group of researchers at MIT and the Tufts-New England Medical Center (T/NEMC). The project corresponding to goal 1, above, was conducted principally by Drs. Jerome P. Kassirer and Alan J. Moskowitz at T/NEMC and Dr. Benjamin Kuipers, who was a Research Scientist at the MIT Laboratory for Computer Science (LCS) during the 1984-85 period. Project 2 has been conducted chiefly at LCS by Prof. Peter Szolovits, Prof. Ramesh Patil, Dr. William J. Long, and their graduate student research assistants. Project 3 was done mostly at T/NEMC by Dr. Stephen G. Pauker, Dr. Moskowitz and their associates and trainees in the clinical decision making program. Because of our ongoing close research relationship, the projects are not as distinct as this description implies. Frequent meetings among all the participants, as well as interactions through courses and seminars assures a close collaboration.

The intended beneficiaries of the results of this resource-related research project are the "AI in Medicine" (AIM) community, as chiefly represented by the SUMEX-AIM resource and other related research projects around the country. The techniques being developed, especially those related to goals 1 and 2, are critically needed components of most medical consulting and critiquing programs. Within our own research laboratory, results from both these efforts are already being applied in our project entitled "A Program for the Management of Heart Failure." The programs and methods developed toward goal 3 have been most extensively used in the clinical decision making service at T/NEMC both in research and in patient care, and have been distributed to over 100 other users.

In the coming year we plan to continue the general path adopted for goals 1 and 2. Our work on goal 3 has essentially been accomplished, and (as planned) will be terminated. Dr. Kuipers has accepted a position as Associate Professor in the Computer Science Department at the University of Texas at Austin. He will continue to work with us on this project, as he had started to do when he was on the faculty of the Tufts University Mathematics Department.

In the following section, we briefly review the major accomplishments of our project during the past year, and in subsequent sections we report in more detail on our progress and plans toward each of the three goals.

2. OVERVIEW OF RESEARCH PROGRESS

In the past year of work on the clinical cognition subproject, we have collected transcripts from eight expert physicians solving difficult clinical problems for which a formal decision analysis previously had been carried out. Four of these transcripts were analyzed in detail using referring phrase analysis and assertional analysis, and the data from some were incorporated into a frame structure. We also carried out an extensive comparison between the problem solving approaches evident in the transcripts and the formal decision analysis model. We presented these results at the meeting of the Society for Medical Decision Making in November 1984, and we have prepared a draft of a paper describing this research which we expect to submit to one of the cognitive science journals in the near future.

In studying means of providing better explanation capabilities to medical expert programs, we have pursued several lines of investigation all centered on the use of deeper and more explicit models of various kinds of knowledge in the program. Prof. Patil has begun to investigate the use of the NIKL knowledge representation formalism for medical knowledge. This system differs from former knowledge representation formalisms in that it has a very well-defined specific semantics and thus forces the person developing the representation of any piece of knowledge to be precise and explicit about all relationships in that knowledge. Ms. Phyllis Koton has completed a comparative study of two means of encoding expertise about the interpretation of genetics experiments, one empirical and the other based on more detailed models of the operation of genetic mechanisms. These results will be presented in a paper at the International Joint Conference on Artificial Intelligence (IJCAI) in August 1985. Mr. Elisha Sacks has described a new means of analyzing systems described in causal terms by an analysis of the mathematical form of their temporal behavior. In his Qualitative Mathematical Reasoning (QMR) program, a system is analyzed in two stages: first a mathematical description of each significant parameter is derived, and then the qualitative behavior of that form is deduced. This work will also be reported in a paper at the upcoming IJCAI conference. Mr. Michael Wellman has completed a Utility Reasoning Program (URP) that has an explicit encoding of a large number of theorems of utility theory and permits the investigation of partially-specified preference structures of individuals faced with value-laden decisions. Dr. Long has continued to develop means of showing a user the implications of given facts about a patient according to the constraints of a causal model. Both text and graphical presentations have been investigated as means of describing the results of analysis. A paper describing this approach was presented at the 1984 Computers in Cardiology conference.

The decision analysis subproject has met its objectives, to develop a useful and easy-to-use set of tools for clinical decision analysis. This two year developmental project proposed to improve a preliminary version of a decision tree analysis package sufficiently to permit its distribution to other investigators. The original program DECISION MAKER was developed on the Apple II computer and written in a dialect of UCSD Pascal. It required the user to purchase a language card and the UCSD operating system, which were substantial financial deterrents to dissemination. The IBM PC has now become the preeminent computer in the world personal computer market so we have directed our developmental efforts toward it and its compatibles. We found that the use of the UCSD operating system continues to be a deterrent to dissemination and have therefore converted to Turbo Pascal and now distribute object code. The current product operates under MS-DOS.

Over the past two years we have extended the basic tree folding program in an upward compatible way, adding a broad variety of new features that we found useful in our own unit. We have also incorporated requests from users in beta test sites. The current program exists in two versions: the more general version supports the IBM PC and its compatibles with all standard displays and printers; a more specific version supports only the PC with a color-graphics card and an Epson printer, but has more powerful graphics display options and somewhat easier tree input. Both programs have compatible file storage formats and include a detailed manual, limited on-line help and a variety of new features such as cost-effectiveness analysis, Boolean control structure, subtrees, extensive context-dependent debugging facilities, and Markov processes. We have distributed some 100 copies of the programs, and it has been used by another 50 individuals working in and visiting our unit. Their suggestions have helped shape development of the programs, which have been demonstrated at a variety of national medical meetings and educational courses for physicians. A version of the program is being developed, under separate funding, for the Apple Macintosh computer. It is being incorporated into an undergraduate medical curriculum at several medical schools. The program enabled construction of the largest medical Markov model to date as the basis for NHLBI's consensus study on artificial hearts [5].

3. CRITICAL DECISION NODE PROJECT

The objective in this project, conducted principally by Drs. Kassirer and Moskowitz at T/NEMC and Dr. Kuipers at LCS, is to describe the knowledge representations and problem-solving methods employed by physicians making difficult management decisions, involving considerable risk and uncertainty. We investigate the nature of clinical expertise and attempt to characterize the development of probabilistic reasoning by comparing the knowledge and problem-solving methods used across experimental subjects differing substantially in expertise.

Our research method involves collecting actual reasoning protocols from a variety of subjects and then carefully analyzing the transcript of such protocols.

3.1. Protocol Acquisition

We have collected eight protocols over the last year, which will provide material for testing our hypotheses about clinical problem-solving and for further characterizing the behavior of expert physicians. We interviewed four academic pulmonary subspecialists, and tape recorded their responses to two case histories. The interviews lasted from two to three hours each. The clinical case histories were written and presented to the physician-subjects on cards, one paragraph at time. The first case involved the management of an elderly man with acute pulmonary infiltrates and underlying preleukemia; the second case involved the management of a man with an acute myocardial infarction and an expanding abdominal aortic aneurysm. The subjects were asked to read each paragraph and to do all of their thinking aloud. After the subjects completed their spontaneous discussion about the case, the interviewer posed questions to clarify the subjects' thoughts on the case material and to probe the reasoning process. Each interview was transcribed verbatim.

3.2. Transcript Analysis

Our transcript analysis work has involved an extensive analysis of the behavior of three expert physicians (pulmonologists) making management decisions for a desperately ill patient with preleukemia, chest pain, fever, and an acute pulmonary infiltrative process. The problem involved choosing between empiric treatment, no treatment, and employing potentially dangerous testing to guide treatment of this patient. The uncertainty in the etiology of the underlying disease process, the high risk associated with gathering further information and the urgent need for immediate treatment of this immunocompromised patient required probabilistic reasoning and careful attention to several potential morbidities and competing mortality risks.

The most recent draft of our manuscript, "The Critical Decision Node,"[2] summarizes the methodology that we have developed for protocol analysis and delineates the insights that we derived from analyzing the protocols mentioned here.

Research in transcript analysis proceeded in three steps: segmenting the transcript, determining the framework of the transcribed reasoning process, and then identifying the set of assertions being made by the subject.

Segmenting the Transcript: Each verbatim transcript was segmented into lines and paragraphs which facilitated the subsequent analysis. The process involved breaking up sentences so that only one or two pieces of a complex concept was contained on a line, which frequently spread a sentence over many lines. Paragraphs were delineated as coherent chunks of narrative. Paragraph breaks were inserted with each change of topic or style of reasoning. Each of the three transcripts were segmented into roughly 1000 lines.

Determining the Conceptual Framework of the Reasoning Process:

Referring Phrase Analysis: Referring phrase analysis identifies the set of referring phrases in a protocol excerpt and defines a small natural universe of underlying conceptual objects which can be the referents of those phrases. This universe constitutes an ontology for the domain being discussed. The referring phrase analysis that we performed involved three steps. 1) Each line in the transcript excerpts was reviewed and the object phrases identified. These phrases are distinct from the wording used to refer to them. 2) The object phrases were then characterized by their role in the problem formulation. We specifically focused on phrases that had anything to do with likelihoods, probabilities, possibilities or alternatives. Table three in "The Critical Decision Node" [2], page 15, shows several lines of a segmented protocol extract with the referring phrases underlined. The righthand column in that table gives the classification of each phrase, which is explained in table three (page 16) of the same manuscript. Each identified phrase was classified into one or more of a small set of categories, according to the type of description of likelihood asserted. 3) The classification system was tested by its ability to account cleanly and completely for the set of referring phrases identified.

Our conclusions from the referring phrase analysis can be summarized as a set of constraints for specifying a knowledge representation of the concept of likelihood. Derived from observations of expert humans, such constraints are useful in the search for an accurate model of human reasoning under conditions of uncertainty. Following the human model may also be the best way to construct useful knowledge representations for effective expert problem solving systems. The constraints are as follows: All likelihood descriptions should be symbolic, as our analysis did not reveal evidence of the storage of high resolution numerical values, or the cognitive uses of arithmetic operators. There should also be at least two distinct representations for probabilities; our analysis revealed categorical references, "fuzzy" values on an absolute scale and ordinal references, relative value terms for pairs of probabilities, (e.g., greater than, less than, equal to). Numerical values would only be used to specify interval anchor points or focal values, for purposes of locating neighboring likelihoods. These constraints can be combined with domain-specific and computational constraints which will considerably aid our search for an accurate model of human reasoning under uncertainty.

Determining The Content of The Knowledge Being Reasoned About: Assertional Analysis: Assertional analysis identifies the set of assertions being made in the excerpt about the objects identified by referring phrase analysis. A set of relations on objects and connectives and operators on sentences are then defined to express the content of the assertions. This constitutes an epistemology for the domain being discussed.

We chose to represent the objects and their relations in a frame system, a data structure frequently employed in expert system development. Figure 2-1 is an example of a frame from our analysis of one of the transcripts. This structure has the advantage of permitting the knowledge to be concisely stated and completely specified. It also makes it possible to ascertain that the associations encoded work correctly. Assertions are also encoded in decision rules. Figure 2-2 is an example of one such rule. Features of frames and rules facilitate the specification of a computer simulation of the reasoning process, which is one measure of the completeness of the knowledge representation.

PROCEDURE (general)	
PURPOSE:	Dx, Rx or both
PATIENT:	recipient of procedure
SITE:	anatomical location
OUTCOME:	(event, likelihood)
PREREQUISITES:	set of conditions
INVASIVENESS:	low...high
INFORMATION-YIELD:	(description, likelihood)
SPECIMEN-YIELD:	fragment or large

PROCEDURE (transbronchoscopic)	
IS-A:	procedure
PURPOSE:	Dx
SITE:	lung
OUTCOME:	(mortality, likelihood-b)
PREREQUISITES:	(clot-status patient normal)
	(general-health-status patient reasonable)
INVASIVENESS:	low
INFORMATION-YIELD:	(diagnostic-tissue, likelihood-d)

On the top is the general notion of a procedure, with slots for each procedure attribute. The slot name is on the left, and the possible slot values are on the right. The slots represent those attributes of procedures that the physician referenced in this transcript. The bottom of the figure shows the transbronchoscopic frame, a specific instance of a procedure. The slots associated with this procedure are the attributes that the physician associated with this biopsy and are a subset of the slots seen above. Note that there are attributes for the organ of interest, the risks of the procedure and the information content of the result.

Figure 2-1: Procedure Frame

3.3. Comparison to Decision Analysis

We have compared the decision process seen in the protocols to a normative model of decision making under uncertainty, namely decision analysis. We found the reasoning process exhibited in the transcript to closely resemble the conceptual framework of

IF	Patient(Chest.x-ray)	= patchy-infiltrate
	Disease(caused-by)	= microorganism
	Microorganism (identified-by)	= morphology
THEN	Procedure(diagnostic)	= lung-bx

The above decision rule can interpreted as follows: if the patient has a patchy pulmonary infiltrate and the disease is caused by a microorganism and the microorganism can be identified by its morphologic characteristics, then a transbronchial biopsy is recommended. While this rule is out of context and somewhat simplified, the transcripts contain many such examples of decision rules.

Figure 2-2: Decision Rule

decision analysis but to employ a significantly different processing algorithm. A detailed discussion of this comparison was presented at the Sixth Annual Meeting of the Society of Medical Decision Making, November 1984, in Washington DC; an abstract of the discussion will be published in *Medical Decision Making* Vol. 4, No. 3 [4].

3.4. Application of Results to the Teaching of Clinical Medicine

In addition, we intend to exploit the concepts generated from this research to develop new teaching tools. Last year we reported on a teaching method based on the iterative hypothesis-based approach to clinical problem solving used by expert physicians [1]. The basis for this teaching method was our work on diagnosis, using a research methodology that involved interviews and transcript analysis. In the past year we have extended this concept in a series of published reports in the journal, *Hospital Practice.* In the coming year we intend to apply the principles learned in our studies of the critical decision node to additional publications in this journal. We selected this journal as the site of such teaching cases because it has an exceptionally large audience that includes academic physicians, house officers, and medical students, and because it publishes teaching materials of exceedingly high quality. We have agreed to publish one teaching case each month.

3.5. Goals for 1985-86

Our goals for the coming year are twofold. We intend to continue protocol analysis so that we may refine our model of reasoning under uncertainty and further characterize clinical expertise. We will be looking at previously collected protocols of clinical subspecialists solving problems within and outside their area of expertise, in hope of delineating general purpose and domain specific problem-solving techniques. Because our technique of protocol analysis is highly dependent on our obtaining fairly detailed exposes of the reasoning of our subjects, the collection of additional protocols may be necessary. In these analyses we plan to focus on the progression and outcome of the decision process, distinguishing the differences of agenda and conclusions among the subjects. To accomplish this, we will be refining the technique of script analysis [2], a technique designed to reveal the goal structure of the problem-solving process and the subject's explanation strategy.

Secondly, in conjunction with our collaborator, Dr. Kuipers, we intend to work toward specifying an implementation of a computational model of the reasoning process seen in one of the transcripts. Our goal is to have the computational model reach the same decision that the human expert did and to observe the intermediary states of the decision process for comparison to observations made about the transcript. Because most physician subjects verbalize only some of the knowledge that they use to reach their decision, achieving this goal may require a second interview.

4. EXPLANATION AND JUSTIFICATION BY EXPERT PROGRAMS

The objective of this project, conducted chiefly by Profs. Szolovits and Patil, Dr. Long, and their graduate student research assistants, is to develop a methodology for providing appropriate explanations to the user for the reasoning provided by a medical consultation system. These explanations should convey to the user the essential nature of the assessments and conclusions provided by the system, thus making the reasoning understandable. Our work began with the idea that by simply making explicit reasoning steps of any program, it would surely lead to understandable explanations. However, we have been drawn in this project to the conclusion that we must develop and investigate new means of knowledge representation and reasoning for the programs in order to make their behavior meaningful. As outlined in the overview Section 2, we have undertaken five different projects to explore various aspects of this problem. Each is discussed in the following sections.

4.1. New Representations of Basic Medical Knowledge

In most medical AI research projects, the particular form of knowledge representation has not been considered critical so long as it was adequate to express the knowledge that was essential to the operations of the program under study. As a result, many incompatible forms of knowledge representation have been developed, differing in many cases only for uninteresting, incidental reasons. In addition, and because the representation formalism was not itself the object of study in these projects, the medical representations have often lacked the integrity and clarity of state-of-the-art research systems for knowledge representation. This has made the task of systematically explaining the content and methods of such a knowledge base unnecessarily difficult.

To overcome these problems, we have begun to investigate the design of a schema for representing common medical knowledge and to develop a knowledge base of medical facts to be shared by various programs and projects in (ultimately) several medical domains. Such an approach should avoid wasteful duplication of efforts needed in developing knowledge bases for individual projects, and it should ease the problems of integrating programs in different medical sub-specialties. Key requirements for successful implementation of this approach are that the knowledge representation language should have a well specified semantics and that the knowledge encoded in such a system be organized according to a well-developed epistemology.

We have chosen to use the NIKL knowledge representation system (a successor of KL-ONE) for exploring these issues for the following reasons: First, it is one of the best known and widely published knowledge representation systems, thus providing a considerable advantage in communicating our ideas and sharing our knowledge base with researchers in other centers. Second, it provides well specified and widely researched semantics allowing us to encode medical knowledge with precision. Finally, it allows for the organization of a knowledge base on multiple intertwined hierarchies and provides an algorithm for automatically classifying any concept or query against the lattice of concepts present in the knowledge base, making additions of new concepts to the knowledge base as well as pattern matching against existing concepts in the knowledge base efficient.

We have begun by reimplementing a small portion of the existing knowledge base of the ABEL program. This knowledge base consists of the following components:

1) knowledge of the anatomy and physiology of the human body, including knowledge of body fluids, their compositions and regulation mechanisms;

2) knowledge of pathophysiologies, classification of primary disease etiologies and temporal courses of disease progression;

3) knowledge of diseases and syndromes organized hierarchically based on their anatomical site of involvement, their pathophysiology, their primary etiology and temporal patterns; and

4) associations between diseases and their clinical symptoms and causal relations between different diseases, expressed at multiple levels of explicit detail.

We will illustrate the organization of medical knowledge in the knowledge base with the help of a few selected examples. The first (Figure 2-3) shows a partial hierarchy of the anatomical parts of the kidney. It shows, for example, that the nephron is a part of the kidney, that the tubule is a part of the nephron and that the proximal-tubule is a part of the tubule. The anatomical entities organized in such a hierarchy are then used as anchor points for describing additional anatomical relations, as illustrated in Figure 2-4, which shows the pathway of renal filtrate as it moves from the nephron to the collecting duct. This information is itself organized hierarchically at varying levels of detail. This ability to express and reason with medical knowledge at varying levels of detail is a key to high performance AIM systems, which must contain large volumes of basic medical knowledge and yet perform efficiently at the clinical level. This can only be achieved if the knowledge base is organized in the above fashion, allowing the programs to bypass the detailed knowledge in all but those areas where such detail is relevant to the case at hand.

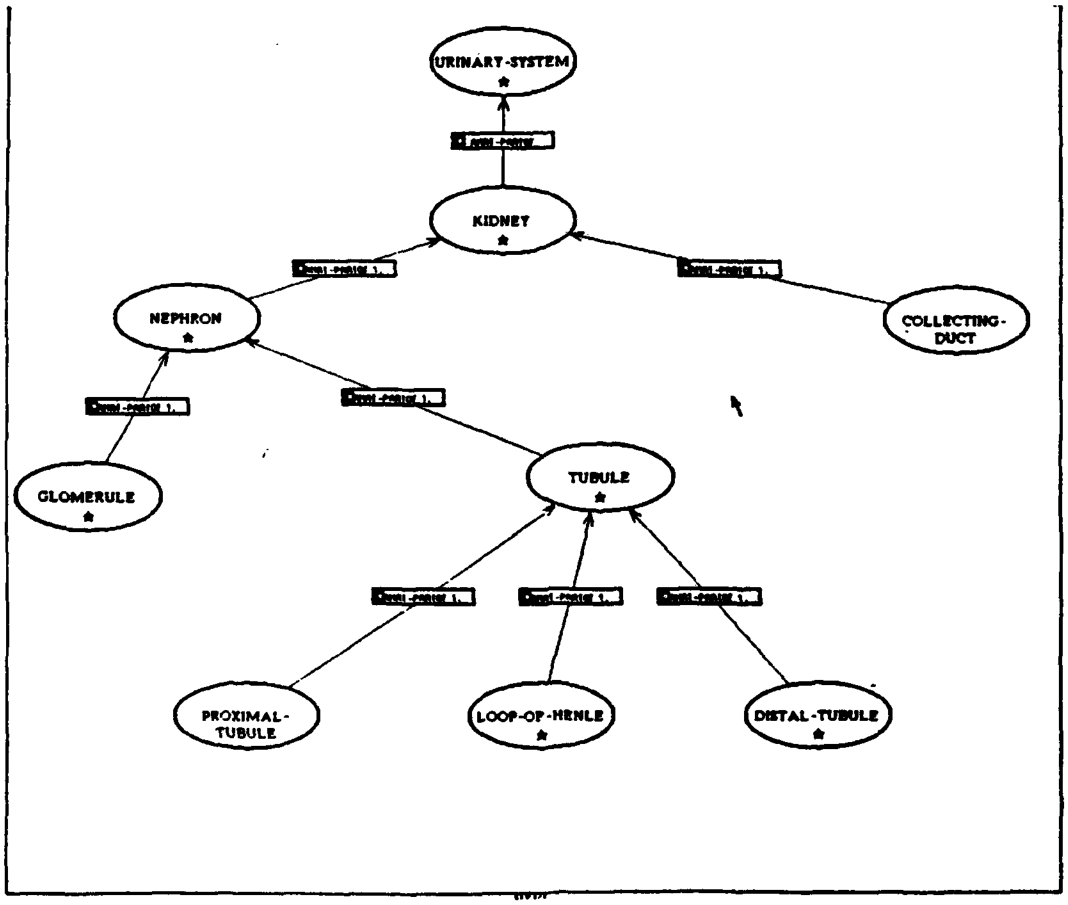


Figure 2-3: An Anatomical Taxonomy for Parts of the Kidney

Diseases can be organized in a hierarchy based on a number of different organizing themes. One organizing principle used commonly is based on the organ system involved in a disease. Thus a disease could be classified as being "renal disease" which could be further divided into diseases of the nephron, interstitium, etc. Another common organizing principle is based on the primary etiology of disease. Thus diseases are often classified as being infectious or degenerative, and the infectious diseases can be further subclassified as being viral, bacterial, etc. In our present work, we are exploring the use of a number of these dimensions simultaneously in organizing a heterarchy of diseases. Figure 2-5 shows a small fragment of such a disease taxonomy. Renal disease is defined to be a disease whose site of involvement is the urinary system. Next, interstitial-renal-

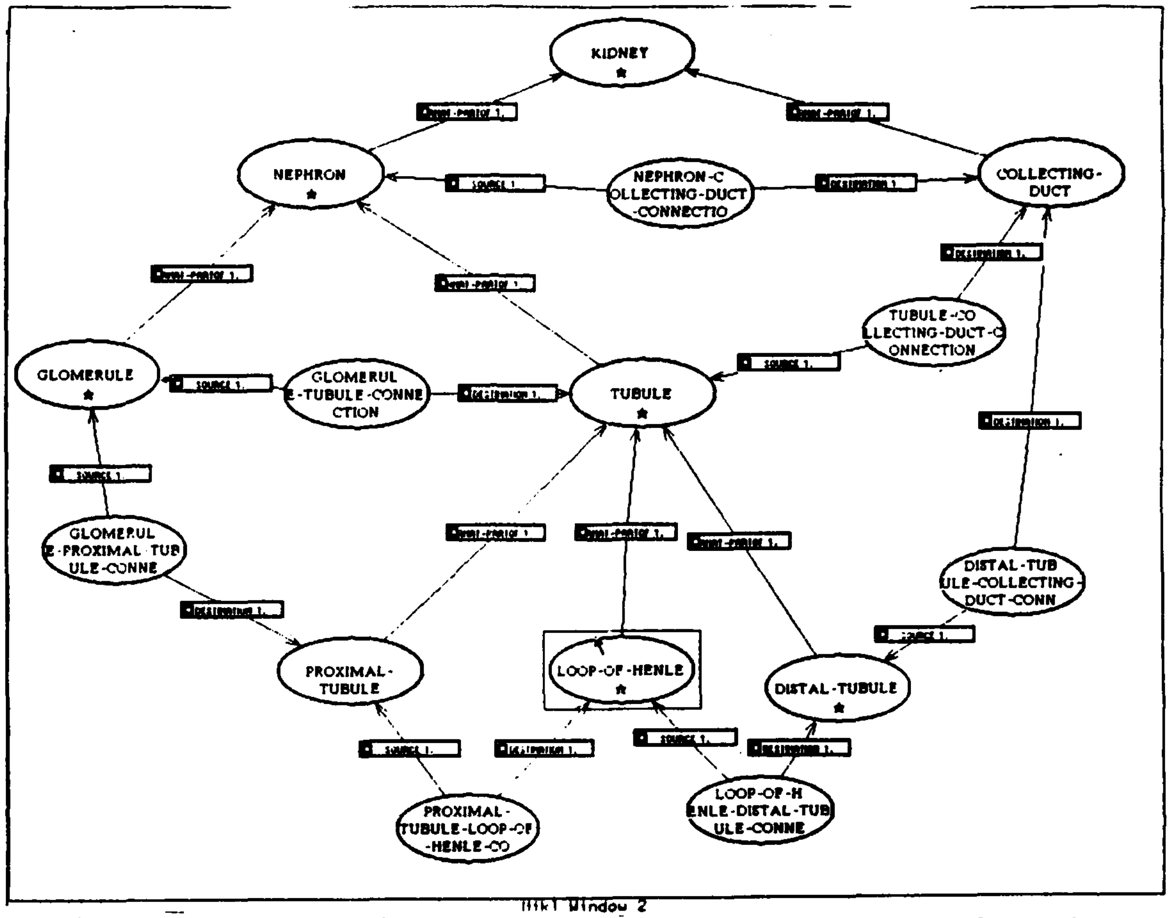

KIDNEY
NEPHRON
NEPHRON-COLLECTING-DUCT-CONNECTIO
COLLECTING-DUCT
TUBULE-COLLECTING-DUCT-CONNECTION
GLOMERULE
GLOMERULE-TUBULE-CONNECTION
TUBULE
GLOMERULE-PROXIMAL-TUBULE-CONNE
DISTAL-TUBULE-COLLECTING-DUCT-CONN
PROXIMAL-TUBULE
LOOP-OF-HENLE
DISTAL-TUBULE
PROXIMAL-TUBULE-LOOP-OF-HENLE-CO
LOOP-OF-HENLE-DISTAL-TUBULE-CONNE
source 1.
destination 1.
Window 2

Figure 2-4: The Pathway of Renal Filtrate at Several Levels of Detail

disease is defined to be a disease whose site of involvement is the renal-interstitium. The system then can automatically classify interstitial-renal-disease to be a specialization of renal disease because the renal-interstitium is a part of the kidney, which is a part of the urinary-system. Similarly, we define infectious disease to be a disease whose etiology is infectious. Next, when infectious-renal-disease is defined to be a specialization of infectious disease and renal disease, it automatically inherits its anatomical site of involvement as being the urinary-system and its etiology as being infectious. Finally, when we enter an infectious renal interstitial disease such as pyelonephritis, it is correctly placed into the hierarchy, as shown. Returning to the example of infectious-renal-disease, we note that by its placement in the heterarchy, we are able to identify diseases more general or more specific to it along any of the represented dimensions. We believe that such an organization will allow us to develop powerful reasoning strategies for effectively aggregating and refining disease hypotheses. It should also help to focus on a small number of aspects of a disease and thus to aid the formulation of small and effective differentiation problems for diagnostic inquiry.

Plans for 1985-86: We plan to continue to investigate the applicability of

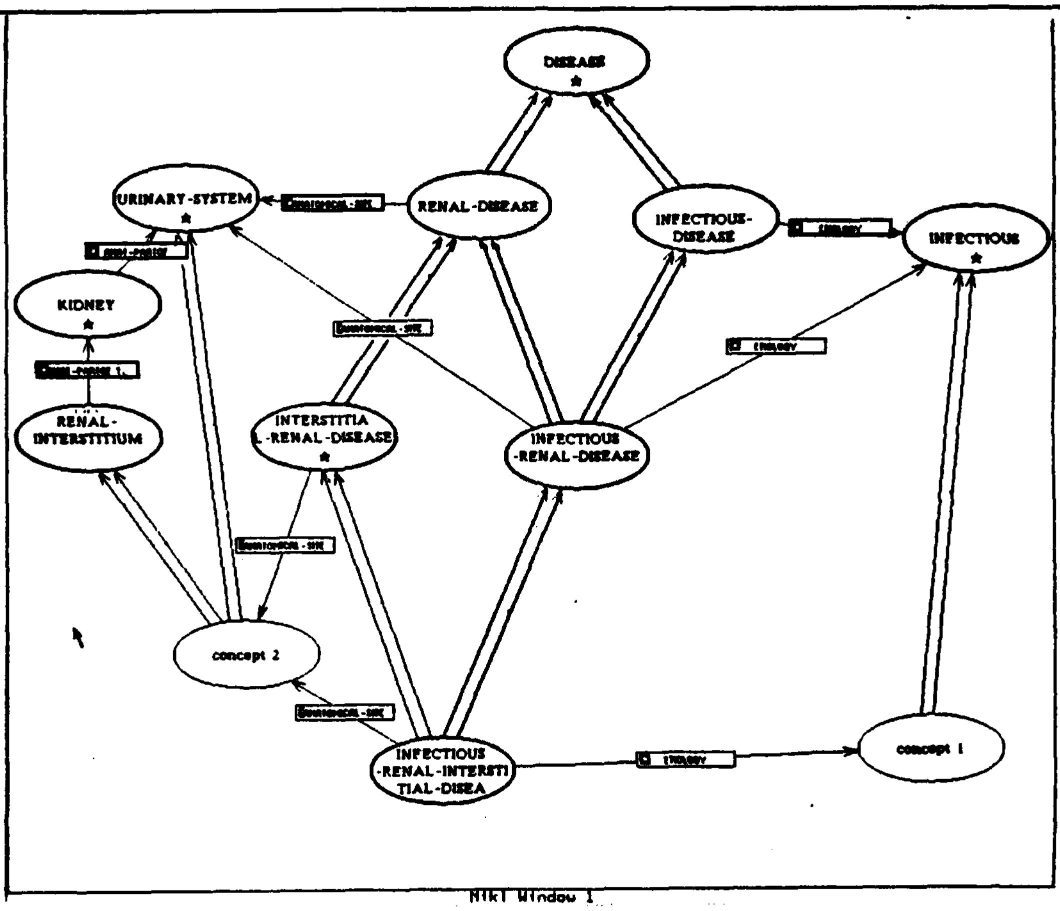


Figure 2-5: Fragment of a Heterarchic Organization of Diseases

knowledge-representation methods, such as those employed in NIKL, to represent detailed medical knowledge. Our plan for the coming year is to broaden the initial steps described here to represent more of the knowledge of the ABEL program in a principled manner. In addition, we have begun conversations with other researchers in the Laboratory for Computer Science who are interested in the development and application of parallel computing techniques, and this is likely to lead to a study of the use of a NIKL-like language on a parallel computer. If this occurs, we would expect to use the knowledge base being experimentally constructed in this project to serve as a study case for that other project.

4.2. Problem-Solving Systems for Molecular Genetics

As we have stressed, the ability of a system to explain its knowledge and conclusions is strongly influenced by the degree of cleanliness with which its knowledge is internally expressed. We have conducted a study that gives support to this view in comparing two versions of a program whose task is to explain a set of data about the expression of a genetic mechanism. Both programs contain knowledge about how to determine the malfunctions of the basic mechanism of transcription and translation, according to the operon theory of procaryotic genetics.

The first program, GENEX I, used a procedural encoding of how to trace down malfunction, essentially implementing a large flowchart. Even with this relatively implicit approach, it was able to answer correctly the relevant examination questions from an undergraduate MIT genetics course. A sample of its operation on a problem from that examination is given below, with the problem statement in bold face and comments interspersed in italics.

The enzyme *uncine synthetase* (USase) of the hypothetical bacterium *Altacoccus profundii* synthesizes the essential amino acid uncine. When no uncine is present in the growth medium USase is made at a high level, allowing cells to grow. When uncine is added to the medium, however, very little USase is made (the cells use the added uncine to grow).

This information can be represented to the program as follows. Let the operon be called **unc**. *The gene coding for USase is labeled* **unc-gene**.

```
unc
(UNC-GENE USASE)
```

This is what we the program has been told about USase:

```
(plist 'usase)
(PURPOSE SYNTHESIS END-PRODUCT UNCINE INV-PROPORTIONAL-TO
         (UNCINE))
```

`things-present` *is a list of the substances present in the medium at the time of interest. The substances besides uncine are acting as red herrings.*

```
things-present
(LACTOSE TRYP UNCINE ARABINOSE)

(initialize unc)
is UNC-GENE product made? NO
```

The gene product, USase, is not made in the presence of uncine.

```
is there a mutation in UNC-GENE? (yes, no, unknown) NO
is there an unlinked mutation? (yes, no, unknown) NO
was diploid constructed? NO
DONE
```

The program now has all the information stated in the above paragraph.

Outline briefly, at the molecular level, two simple but different models that would explain why very little USase is made in the presence of uncine. In each, be sure to indicate the role of uncine.

```
(genex unc)
assuming no misfunction at UNC-GENE promoter
is UNC-GENE a regulatory gene? NO
assuming UNC-GENE can be repressed
assuming UNC-GENE repressor is repressible
assuming UNCINE is corepressor for UNC-GENE--
        UNC-GENE is inactive in presence of UNCINE
```

GENEX I *has found one model for the behavior of the* **unc** *operon--that it is negatively controlled and corepressed by uncine. It used a rule which states that genes which code for biosynthetic pathways are repressible by their end-products.*

```
assuming no misfunction at UNC-GENE operator
checking positive control of UNC-GENE
assuming UNCINE is corepressor for UNC-GENE--
        UNC-GENE is inactive in presence of UNCINE
done checking positive control of UNC-GENE
```

A second model has been found, that **unc** *is positively controlled and the activator is inactivated by the presence of the co-repressor, uncine.*

```
message possibly attenuated due to presence of UNCINE
NIL
```

In fact, GENEX I *has found a third model for the behavior of* **unc**, *that uncine acts as an attenuator on the transcription of the* **unc** *genes. It used a rule which states that genes which code for the synthesis of amino acids can be attenuated.*

In contrast to the procedural representation of GENEX I, the successor program, GENEX II, contains explicit representations of the following concepts: operon, promoter, operator, terminator, binding-site, regulatory-site, gene, structural-gene, regulatory-gene, nucleotide, nucleotide-sequence, codon, DNA, RNA, amino-acid, amino-acid-sequence, protein, structural-protein, regulatory-protein, sugar, repressor, activator, inducer, corepressor, enzyme, and polymerase. Its representations of fundamental operations of molecular genetics are in terms of small operation models expressed in Prolog, such as the following model of the BIND operation:

```
bind(Mol, Site) :- binding-site(Mol, Site),
                   complementary-conform(Mol,Site),
                   \+ bound(X,Site),
                   free-to-bind(Mol),
                   assert(bound(Mol,Site)).
```

The representation of the *lac* operon is given in Figure 2-6.

```
operon(lac-operon).

part-of(lac-operon, lac-operon-promoter).
promoter(lac-operon-promoter).

part-of(lac-operon, lac-operon-operator).
operator(lac-operon-operator).

part-of(lac-operon, lac-operon-reg-gene).
reg-gene(lac-operon-reg-gene).

part-of(lac-operon, b-galactosidase-gene).
struct-gene(lac-operon, b-galactosidase-gene).

part-of(lac-operon, b-galactoside-permease-gene).
struct-gene(lac-operon, b-galactoside-permease-gene).

part-of(lac-operon, b-galactoside-transacetylase-gene).
struct-gene(lac-operon, b-galactoside-transacetylase-gene).

part-of(P,overlap-region(P,O)) :- promoter(P),operator(O).
part-of(O,overlap-region(P,O)) :- operator(O),promoter(P).
next-to(lac-operon-promoter,b-galactosidase-gene).
next-to(b-galactosidase-gene,b-galactoside-permease-gene).
next-to(b-galactoside-permease-gene,
        b-galactoside-transacetylase-gene).

product(b-galactosidase-gene, b-galactosidase).
enzyme(b-galactosidase).
substrate(b-galactosidase, lactose).
purpose(b-galactosidase, catabolic-proc).

...etc
```

Figure 2-6: Representation of the *lac* operon in GENEX II.

These representations give GENEX II an actual model of how genetics operates. Thus, it is able to work through all possible interpretations of the facts at hand that are consistent with its understanding of genetics, and therefore to identify all sets of conditions that can account for the known facts. With such explicit representations, GENEX II is able to deal with examples in which multiple mutations occur -- examples in which the more rigid heuristics of GENEX I would have failed.

Details of this project may be found in the report by Ms. Phyllis Koton, "Towards a Problem Solving System for Molecular Genetics," MIT Laboratory for Computer Science Technical Report TR-338, May 1985.

Future Plans: We have no current plans to continue this specific work on genetic modeling. We have, however, begun to examine the application of the same fundamental modeling techniques to the problems of finding commonalities among the descriptions of different patient cases. Although this idea is not yet well-developed, our examination of the problem-solving process employed by physicians suggests that on occasion they use detailed recollections of specific former cases to guide their current thinking. We plan to investigate ideas based on this observation, in the domain of patients with heart failure.

4.3. Qualitative Mathematical Reasoning

In seeking deeper models of understanding of human health and illness, we often develop formal models of how the interactions among parts of the body and various disease entities generate the observable behavior of the overall system. Although the sciences have developed powerful mathematical tools for analyzing the behaviors of complex systems, classical approaches such as modeling with differential equations demand a level of precision of knowledge that is often unachievable in the current state of understanding of medicine. Nevertheless, human clinicians are able to reason about the *qualitative* behavior of such systems, often relying on descriptions much weaker than differential equations. Instead, for example, they may know simply that a rise in one quantity invariably leads to a rise in a second, or that one measurable value is the time integral of another (e.g., a serum drug level is the integral of infusion rate minus excretion and metabolic breakdown) even if neither value can be precisely characterized.

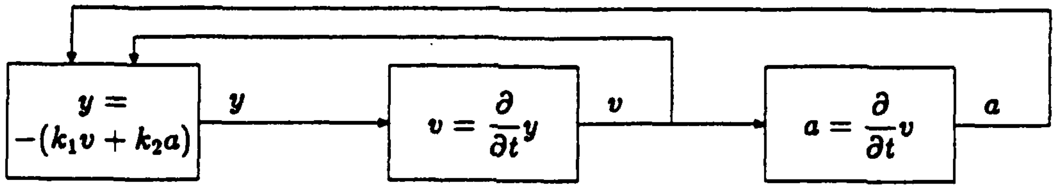
Both in medical and non-medical projects in AI, various suggestions have been made of how to achieve such reasoning within a program. Most of these have centered on the idea that even partial descriptions of physical or physiological systems may be symbolically simulated, to yield a trace of the expected future behavior of the system. These approaches have been moderately successful at predicting the behavior of simple systems (such as an electric buzzer, for example), but they fail to exploit any of the mathematical sophistication that has accumulated over the centuries in analyzing complex systems not by simulating them but by reasoning about the forms of the functions that describe their behavior.

In this project, we have applied such classical mathematical analysis methods to the sorts of qualitative (or imprecisely-specified) problems that have been the recent subject of study in the AI qualitative simulation field. The result has been a program called the "Qualitative Mathematical Reasoner," whose operation rests on two fundamental insights:

1) Imprecision may often be modeled by introducing symbolic parameters in conventional mathematical descriptions of a system; and

2) if it is possible to find a closed-form solution to such a parameterized system, then classical methods of analyzing the behavior of such functions can provide definitive and useful analyses of the function's asymptotic behavior, aperiodicity, values at interesting distinguished points, etc.

As an illustration of the power of this technique, consider the case of a damped spring, described by the following network:


$y = -(k_1 v + k_2 a)$
y
$v = \frac{\partial}{\partial t} y$
v
$a = \frac{\partial}{\partial t} v$
a

It obeys the equation of motion

$$y(t) = -k_1 v(t) - k_2 a(t) \text{ with } \begin{cases} k_1 > 0 \\ k_2 > 0 \end{cases}$$

which can be derived from the network, and this is solved to yield

$$y(t) = 2y_0\sqrt{k_2}\Delta e^{-\frac{k_1}{k_2}t} \cos\left[\frac{t}{\beta} - \arctan(k_1\Delta)\right] \text{ with } \begin{cases} \Delta = \frac{1}{\sqrt{4k_2 - k_1^2}} \\ \beta = 2k_2\Delta \end{cases}$$

The program then produces the following description of the points of interest and the behavior of the system in the regions between these:

There is a unique qualitative description of Y.
Time point 1: 0
Y's value is y_0
Between points 1 and 2:
Y decreases.
It accelerates between 0 and $\beta\,[\arctan k_1\Delta + \arctan(2k_2 - k_1)\Delta]$
It decelerates between $\beta\,[\arctan k_1\Delta + \arctan(2k_2 - k_1)\Delta]$ and $\pi\beta$
$\beta\,[\arctan k_1\Delta + \arctan(2k_2 - k_1)\Delta]$ is an inflection point
Y goes through the significant value 0
Time point 2: $\pi\beta$
It is significant because Y reaches a minimum.
Y's value is $-y_0 e^{-\pi k_1 \Delta}$
Between points 2 and 3:
Y increases.
It decelerates between $\pi\beta$ and $\beta\,[\arctan k_1\Delta + \arctan(4k_2 - k_1)\Delta]$
It accelerates between $\beta\,[\arctan k_1\Delta + \arctan(4k_2 - k_1)\Delta]$ and $2\pi\beta$
$\beta\,[\arctan k_1\Delta + \arctan(4k_2 - k_1)\Delta]$ is an inflection point
Y goes through the significant value 0
Time point 3: $2\pi\beta$
Y's value is $y_0 e^{-2\pi k_1 \Delta}$

Y repeats the qualitative behavior from $[0, 2\pi\beta]$
on $[2n\pi\beta, 2(n+1)\pi\beta]$ for $n = 1$ to ∞.
Its magnitude decreases by a factor of $e^{-2\pi k_1 \Delta}$ on each interval.
The limit at ∞ is 0.

This is in sharp contrast with the approach taken by programs based on simulation, which essentially produce only a sequence of descriptions of successive time points and region descriptions, and cannot recognize the long-term trends in the system's behavior such as approaching an asymptote or oscillating.

This work is described in detail in the report by Ms. Elisha Sacks, "Qualitative Mathematical Reasoning," MIT Laboratory for Computer Science Technical Report TR-329, May 1985.

Plans for 1985-86: The methods explored in this project are only applicable when it is possible to find a closed-form solution to the system being modeled. Unfortunately, this is only rarely the case for nonlinear systems, and is even unusual for large linear systems. Our attention is turning, therefore, to studying methods of deriving symbolic *approximate* solutions to such systems, by the application of well-known mathematical techniques such as series expansions, approximation by piecewise-linear functions, etc. In this, we plan to retain the basic approach developed to date. We have also begun to collect interesting medical models in which such analysis methods are applicable. Thus far, a model for the regulation of calcium metabolism looks most promising, and others in the fields of circulation and acid-base buffering are also being examined.

4.4. Reasoning About Preference Structures

A different aspect of medical knowledge, one that plays a particularly important role in making choices under uncertainty when there are risks involved, is to be able to compare preferences for different possible outcomes. In classical decision analysis, this is accomplished by using a utility function, which assigns a numerical value to each outcome. Unfortunately, it is very difficult (if not impossible), to ask a patient, for example, to quantify in this way his or her evaluations of all possible outcomes of a planned series of diagnostic and therapeutic interventions. This difficulty arises from two factors:

1) An individual may be able to state certain general preferences (e.g., living a longer life is better than a shorter one) and general results of utility theory suggest that nearly everyone is risk-averse, translating such qualitative notions into precise numerical assignments introduces a level of precision that far exceeds its accuracy;

2) Many decisions involve evaluation not just along a single scale but in terms of multiple attributes. Thus, it may not only be length of life but also quality of life, short-term pain, money, etc. that influence an individual's decisions.

In an effort to get around the difficulties with classical utility assignments, we have developed a set of programs called the "Reasoning Utility Package" (URP), that allow a program to reason with incompletely-specified utility functions. This program provides two important capabilities:

1) It can apply knowledge of mathematical relationships that can determine whether certain alternatives dominate others even without a precise knowledge of their value.

2) It includes a large body of theorems of multi-attribute utility theory and a theorem-prover that can apply that knowledge, to analyze independence conditions when multiple attributes are present and to suggest appropriate forms of the multi-attribute utility function based on what is known about the relationships among the different attributes.

The first of these has been demonstrated in analyzing a published decision problem from the T/NEMC Division of Clinical Decision Making, showing that many possible decisions can be eliminated even without assuming such specific utility structures as had been employed in the original analysis. The case involves a 72 year-old male with a large abdominal aortic aneurysm and a history of coronary artery disease and cerebrovascular disease. The decision problem is whether to operate to repair the aneurysm in the face of grave operative risks from the other disorders, or to repair the other problems first, at the risk of a ruptured aneurysm.

URP was able to show that, of the seven options for diagnostic and therapeutic interventions considered, four could be eliminated assuming only that preference is monotonic for life-expectancy (i.e., longer life is better). Dominance among the others depended explicitly on the patient's risk aversion (i.e., how much long-term life expectancy he was willing to trade for a smaller chance of immediate mortality). As tighter assumptions were introduced, URP naturally reproduces the original analysis, but it is able to give a more precise understanding of which assumptions about the patient's preferences have what consequences for the analysis.

Plans for 1985-86: The intent of this work on preference modeling is to provide more sophisticated capabilities for decision analysis programs to reason about utilities. We plan to continue this line of research, but will not be able to incorporate its results into a practical decision analysis tool because that portion of the project is now terminating. However, we are currently seeking funding for a new effort to merge techniques of decision analysis and artificial intelligence into a tool for clinical decision making, and our work on preference modeling should be a key beginning for that effort.

5. PROGRAM FOR THE MANAGEMENT OF HEART FAILURE

5.1. Objectives

The objective of this project is to develop a methodology for assisting the physician in reasoning about the diagnosis and management of disease, particularly when causality, physiological relationships, and changes over time are important to the reasoning process. The context for this research is the diagnosis and management of heart failure, a problem which requires a thorough understanding of the hemodynamic and physiological relationships of the cardiovascular system for effective management. The emphasis of the first two years of the project is on developing an adequate representation for the physiological knowledge, developing the appropriate physiological model for the domain, and beginning to develop the algorithms for reasoning about the diagnostic and management problems of the patient.

5.2. Progress Report

Our strategy in the past year has been to look at a representative part of the heart failure domain to understand the pertinent reasoning processes and use that subdomain to uncover requirements for the representation. By concentrating on a subdomain first, it is possible to experiment with representational ideas and variations of the physiological model to explore the advantages and limitations of alternative structures without a large expenditure of effort. The subdomain we have focused on is the management of angina in the patient with heart failure. This domain emphasizes the patient management aspects of the reasoning, thus anticipating the reasoning problems existing in heart failure cases with uncorrectable etiology.

During the past year, we have constructed a physiological model containing the important factors for the management of angina and have used it to address some of the reasoning issues. The model consists of approximately 70 nodes representing physiological parameter values, primary causes, and therapies. Thirty nodes represent physiological parameter values, including `excess myocardial oxygen consumption`, `pulmonary venous congestion`, and `inadequate coronary flow`. Twenty nodes, designated `primary`, represent states whose causes are outside of the model, such as `coronary artery disease`, `anemia`, and `fever`. The remaining 20 nodes represent factors that can be controlled by the physician, including therapy for the angina or heart failure, therapies that interact with the angina management, such as broncodilators, and maneuvers to correct some of the primary states, such as aortic valve surgery to correct stenosis. The nodes were chosen to include those factors that should be considered in the management of patients.

First versions of the input module, diagnostic module, and therapy module were built around the angina model to provide a functioning test environment. The program and the problem domain have been useful in understanding and investigating a number of problems including the classification of causality, setting node values from input data, assessing the completeness of a diagnosis, finding data to refine a diagnosis, finding possible therapies, and determining possible outcomes of therapy.

By addressing a domain where most of the primary causes are uncorrectable, it became clear that some of the most important information about the state of the patient is the existence of treatable conditions that worsen the problem even though they may not cause it. To facilitate reasoning about such conditions, we enhanced the causal representation to include five types of relationships between cause and effect: definite causes, possible causes, worsening factors, possible corrections, and definite corrections. The worsening factors are those conditions that can precipitate or worsen a problem, but are insufficient in themselves account for the problem. For example, a patient with coronary artery disease may experience angina when a fever increases the demand on the heart.

The input data must be interpreted to assign appropriate node values before reasoning can take place. We made an initial pass at this problem by providing a system that allows the user to enter data in a paragraph style, add more data as desired, or change information already entered. A typical example of input, entered by completing phrases, is as follows:

```
The patient data is as follows:

The patient history includes coronary artery disease.  The
patient exhibits angina.  The patient is experiencing
fatigue and shortness of breath.  The patient is treated
with nothing.
```

The input data is mapped to the nodes for which the data provides evidence. Each node for which there is new evidence is evaluated according to simple rules that assign a number from -1 to 1 to the evidence. From this range thresholds are used to determine those nodes with sufficient evidence to assert truth or falsity, those that are unknown, and those for which there is some evidence but the user should decide whether it is sufficient. The mechanism of asking the user about less strong evidence brings the user into the reasoning at a place where familiarity with the patient may be very useful, as well as allowing the reasoning mechanisms to be built incrementally.

With the ability to incrementally change the input, the system proved useful for exploring diagnosis refinement. From the reasoning in angina management, it became clear that identifying the correctable conditions contributing to the patient's problem is essential to the diagnosis. With a patient suffering from coronary artery disease, the ultimate cause of angina is clear but, if there is a rapid heart rate and high metabolic demand from a fever, treating the fever is likely to provide some relief from the angina. The model representation is proving effective for finding those conditions that might be worsening the situation that have not yet been considered. The program can examine the current state of the diagnosis and identify the possible causes and worsening factors that remain unknown. It can then use the evidence mappings to suggest further information that would clarify the diagnosis. For example, in the angina patient it would suggest information to assess any increase in cardiac demand, such as temperature, use of broncodilator therapy, existence of hyperthyroidism, and so forth.

The angina model has also been useful for understanding the reasoning process in selecting therapy. The links to possible and definite corrections provide the connections from the diagnosis to the therapies that might treat the problem. Thus, finding candidate therapies requires examining the diagnostic nodes for such links. There is no guarantee that the therapy is appropriate since there may be effects besides the desired effect. Determining the possible effects of the therapy is a process of examining the paths forward from the therapy to see what changes might result. We have written simple algorithms for doing this in the angina domain. The results have been encouraging, indicating that there is much useful information to be had from simple strategies. However, it is also clear that the relative strengths of relations as well as the overall effects of the several feedback loops are important for pruning the possible effects.

The angina model has proved an effective base for exploring these issues and has led to a number of ideas for enhancements to the representation which we will be trying in the coming months. The exploration also resulted in a paper presented at a Computers in Cardiology conference. [3].

5.3. Objectives

Our research plan for the coming year is to explore strategies for overcoming some of the inadequacies in the representation identified over the past year, extending the experimental angina model to cover aspects of the heart failure problem that have other kinds of reasoning requirements, and use actual cases to critique our algorithms.

The representational inadequacies that we plan to address include the following:

1) At present there is not a one-to-one correspondence between nodes in the model and possible findings from the input. Some findings might be produced by more than one node. As a result, the reasoning needed to use the findings as evidence for nodes has many of the same features as the reasoning within the model. Thus, we will be experimenting with strategies for doing much of the evidence reasoning within the physiological model. The evidence reasoning that will remain outside the model will be that relating to measurement errors, such as variation in experience levels and physical conditions that change the reliability of the measurements.

2) The `possible` and `definite` distinction for causes is sufficient to drive the logic substrate of the program but is insufficient to order the diagnostic reasoning tasks. To allow the program to focus attention on the most likely causes and explanations for the findings, we will add a coarse probability measure to the causal relationships. This information should also prove useful for deciding when the diagnosis is sufficiently complete to allow reasoning about the possible therapies.

3) The issues of feedback and relative changes, for reasoning about the possible effects of therapy and the meaning of observed effects, will be addressed in the coming months. From our understanding of the problem so far, it appears that including very coarse strength relations between cause and effect in the representation will be sufficient to approximate the information in relationships such as the Frank-Starling curves. In addition, it appears advantageous to explicitly identify the feedback loops and characterize their overall behavior under known conditions. We will determine whether this additional information is sufficient to drive algorithms to give the user the desired reasoning behavior.

4) The issues of time of causation were partially addressed earlier in the project, but need to be further developed and incorporated into the same representational framework.

5) Finally, we plan to look at the issue of reasoning about the changes that take place between sessions. This is a problem of obvious importance that depends on a solid foundation of diagnostic and therapeutic reasoning, but we should have a sufficient foundation to begin this investigation during the next year.

The angina model covers many of the issues of importance in the heart failure domain, but there is little in it that requires reasoning about a significant delay between cause and effect. To provide a fruitful environment for incorporating the work on reasoning about causation in time, we will be extending the model to include the reasoning about kidney function and fluid retention. It may also be useful to extend the model to cover one of the valvular diseases causing heart failure to test the interaction between reasoning strategies used by physicians for different problems.

Finally, we will be using data from cases to critique the reasoning possible from the model. The cases will be used to generate a testbed for comparing versions of the program. The abstraction of case data as well as work on the model extensions will be done by research fellow Dr. Steve Kurzrok, a fully trained cardiologist who joined the project in October.

6. TOOLS FOR CLINICAL DECISION ANALYSIS

This project was carried out mostly at T/NEMC by Drs. Pauker, and Moskowitz, and their associates. Some five years ago we first proposed to extend a decision tree analysis package, developed on the Apple II computer, for dissemination and to monitor its use and application. When that proposal was funded the evaluation and dissemination portions of the project were largely eliminated and we were funded only for development to the stage that dissemination could be supported. We have proceeded along that path, experimenting with a variety of developmental environments before settling on our current product. Although we have not been able to monitor program use at the some 100 sites to whom we have disseminated, the developing products have been extensively used in our own units and we have expanded the products as dictated by their extensive applications there. The programs are used on some 10 IBM PC computers in our units and we estimate that they see, on average, 100 man-hours of use each week.

6.1. Systems Configurations

The original program was developed on the Apple II computer, using a 6502 chip. It required the language card, extending the machine to 64K and providing the Apple version of the UCSD Pascal operating system. It was next extended to the Apple III computer, a now defunct machine which was relatively upwards compatible with the Apple II and which was designed around the UCSD Operating System as its kernel. The lack of commercial success of the Apple III (and hence a lack of users -- we distributed only two Apple III versions of our program to the medical community) led us to consider other processors. We developed versions for the Z80 processor on the Heath/Zenith Z-89 computer. That program was inherently slower than its 6502 counterpart and was severely limited by the 64K memory limit of that family of computers. We explored the use of other Pascal compilers, particularly ones that could produce object code, rather than the interpreted P-code used by the UCSD P-system, hoping to gain additional speed. We extensively explored the Pascal MT+ compiler, but eventually were forced to abandon that attempt because the object code version of the program took so much of the 64K RAM that realistic decision trees could not be created.

We eventually shifted to the IBM PC because sufficient market share and program availability made that machine a viable alternative. Our initial development in that environment was limited to the UCSD P-system because of its compatibility with our prior program and because of its support of the overlay structure necessary for a program of this size and complexity. (The IBM Pascal compiler and the Microsoft Pascal compiler did not, and we believe still do not, support overlay segments in a manner needed by our program). We have recently been using a far better compiler, the Turbo Pascal system, which has become the *de facto* industry standard. This compiler supports overlays and produces very fast object code. Furthermore, the object code can be used under the standard MS-DOS and PC-DOS operating systems, so users are no longer required to purchase the UCSD Pascal P-System to use our programs.

We have also explored another MS-DOS version written in the IQLISP programming environment. Unfortunately that system requires the purchase of IQLISP. That program is not as fast as the Turbo Pascal versions, but is able to perform common expression analysis. It also can produce BASIC statements which can then be passed to the IBM BASIC compiler to produce very fast object code which represents a particular decision tree. That program is not suitable for tree development and debugging, because the edit/compiler/basic-compiler/run cycle is very cumbersome. However, once an application has been developed, the code produced by that system allows very rapid sensitivity analyses.

6.2. Extensions to Decision Analysis

The central way that our system differs from previous decision tree systems is the use of sub-trees. Prior to the development of this approach, decision trees were required to be entered in extensive form, with similar portions of the tree being entered many times. In our system, such similar regions are entered only once and other occurrences are simply linked to that subtree with the specification of appropriate parameters (i.e., arguments) and with the subtree returning an appropriate expected value which is then passed further up the tree. The mechanism we established for passing arguments is a binding stack, which is a push down stack that allows multiple copies of the same variable but access only to the last or top copy. The binding stack is linked to tree evaluation, so that pointers in the decision tree place marks in the binding stack. As the tree is folded back, when a pointer is reached, the binding stack is popped, returning the context of evaluation (the set of variable bindings) to a prior value. The mechanism essentially provides local variables in the same manner as early LISP implementations.

One of the major time constraints we found in tree evaluation is the necessity to fold a tree back many times with different utility structures. The most common situation in which this occurs is when the analyst is performing a cost effectiveness analysis: in that case, the two utilities are typically dollar or resource cost and effectiveness -- typically quality adjusted life expectancy. We have found numerous other problems, however, in which dual utility structures are needed, e.g., calculating abortions induced per malformed child avoided, complications per year of vision saved, etc. The problems with folding the decision tree back separately for each utility are several: If there are any embedded decision nodes, then the analyst must be able to ensure that those embedded switches flip the same way for each fold back. In general, that will not be true because the maximization criterion at any node will depend on the utility used. Furthermore, most of the effort required in the foldback is duplicated because the calculation of probabilities, bindings and the creation of tree structure is shared. This led us to create a DUAL utility mechanism, whereby two separate utility structures are calculated and maintained, if desired, in each foldback. In term of calculation time, this mechanism performs dual foldbacks in 60%-70% of the time required for two separate analyses. We also provide the automatic calculation of average and marginal cost-effectiveness ratios for those situations when the two utilities are, in fact, cost and effectiveness.

The next major extension of decision analysis was the creation of the boolean node -- a new node type beyond the traditional three (decision, chance, and terminal). The boolean node is a logical switch which is controlled by the truth status of a variable or an expression. This extension allows the structure of a decision tree to be modified during evaluation, allowing a far more powerful set of processes to be modeled. One obvious use of this mechanism is to allow the modeling of recursive processes, with a control variable or flag being incremented or set to limit the depth of recursion. We use this mechanism to deal with problems involving the order of testing or therapy or if an alternative therapy is available after primary therapy has failed. Boolean nodes also allow the representation of time with a control variable being incremented with each pass through the decision tree. This approach has allowed us to model both Markov and semi-Markov processes and to make hypersimplified decision trees.

A major advance occurred with the implementation of Markov processes using a tree-like presentations which we called Markov cycle trees. This notation used chance nodes to model pathways within a Markov process and used terminal nodes as collectors for states. Control of the Markov process (i.e., when it terminates) is provided by alternative criteria -- the lack of gain of incremental utility, a maximum number of cycles, or some combination of these criteria. The program has automatic half cycle corrections for the first and last cycles and allows the tail expression to be the name of a tree so that Markov processes can be patched into an ordinary decision tree. Of course, dual utilities are supported so that Markov cost-effectiveness analyses can be performed.

6.3. Alternative versions

In addition to the basic machine independent Turbo version (DECISION MAKER), a second version has been developed and tested (SMALLTREE) which provides a great deal more screen support and graphic tree entry and editing facilities. That program was also developed in Turbo Pascal but requires more specific hardware. Both of these versions now partially pre-compile expressions and use address pointers into the binding stack (rather than variable name lookup). These techniques, along with the shift from the interpreted P-system, have produced a 20 to 50 fold increase in execution speed. At this point all use of the SMALLTREE system has been within our own unit; all beta sites have been using the DECISION MAKER system.

This group has also developed a more cumbersome decision package that uses the IQLISP system on the IBM PC. It can compile specific decision trees into basic code which is that compiled by the IBM basic compiler into very fast, but inflexible, object code. The development of that system was necessitated by the relatively slow UCSD Pascal interpreted version of DECISION MAKER which was not well suited for repetitive extensive sensitivity analyses. With the development of the two Turbo versions, the IQLISP program has seen far less use in this group.

6.4. Dissemination

The program is now available as object code running under MS-DOS/PC-DOS for the IBM PC. We have distributed copies to some 100 colleagues at other institutions. Although we have not made any formal evaluation of use, we find three basic levels of success: For naive users who have never constructed decision trees nor received formal training in clinical decision analysis, we have found few successful users, although many continue to dabble with the program. For settings in which physicians and decision analysts are routinely performing analyses, we believe our program is being used extensively, although some of these experienced analysts have some minor disagreements with our style of analysis. A third class of users is constituted by physicians who have spent time in our unit, from one day to many months. These individuals use the program far more extensively and are, in general, happier with its style.

We have developed a library of sample analyses, that have been tested by many visitors to our institution. We have also developed written and on-line manuals for several versions of our program. There is also a limited amount of on-line help screens available.

We plan to continue to expand and modify these programs as the need dictates.

References

1. Kassirer J. P. "Teaching Clinical Medicine by Iterative Hypothesis Testing. Let's Preach What We Practice," *New England Journal of Medicine,* 309 (1983), 921.

2. Kuipers, B., Moskowitz, A. and Kassirer, J.P. "The Critical Decision Node," to appear.

3. Long, W.J., Naimi, S., Criscitiello, M.G., Pauker, S.G. and Szolovits, P. "An Aid to Physiological Reasoning in the Management of Cardiovascular Disease," *Proceedings of IEEE Computers in Cardiology Conference,* Salt Lake City, UT, September 18-21, 1984, 3-6.

4. Moskowitz, A.J., Kassirer, J.P. and Kuipers, B.J. "Clinical Reasoning versus Decision Analysis," abstract of discussion presented at Sixth Annual Meeting of the Society of Medical Decision Making, National Institutes of Health, Bethesda, MD, November 1984, to appear in *Medical Decision Making,* 43, (1985).

5. National Heart, Lung and Blood Institute (NHLBI). "Artificial Heart and Assist Devices: Directions, Needs, Costs, Societal and Ethical Issues," Report of the Working Group on Mechanical Circulatory Support of NHLBI, to appear.

Publications

1. Beck, J.R., Letarte, A.L. and Pauker, S.G. "Clinical Decision Analysis Using Decision Maker," 85-03, Department of Pathology, Dartmouth Medical School, Hanover, NH, 1985. To appear in *Proceedings of 1985 Symposium on Computer Applications in Medical Care.*

2. Granville, R.A. "Controlling Lexical Substitution in Computer Text Generation", *Proceedings of the 10th International Conference on Computational Linguistics and 22nd Annual Meeting of the Association for Computational Linguistics (COLING-84),* Stanford University, Stanford, CA, July 2-6, 1984, 381-384.

3. Hollenberg, J.P. "Markov Cycle Trees: A New Representation for Complex Markov Processes," abstract of a discussion presented at *The Sixth Annual Meeting of the Society for Medical Decision Making,* National Institutes of Health, Bethesda, MD, November 1984. To appear in *Medical Decision Making.*

4. Hollenberg, J.P. "The Decision Tree Builder: An Expert System to Simulate

Medical Prognosis and Management," abstract of a discussion presented at *The Sixth Annual Meeting of the Society for Medical Decision Making,* National Institutes of Health, Bethesda, MD, November 1984. To appear in *Medical Decision Making.*

5. Koton, P. "Towards a Problem Solving System for Molecular Genetics," MIT/LCS/TR-338, MIT Laboratory for Computer Science, Cambridge, MA, May 1985.

6. Kuipers, B. "Qualitative Simulation of Mechanisms." MIT/LCS/TM-274, MIT Laboratory for Computer Science, Cambridge, MA, Apıil 1985.

7. Kuipers, B. "Commonsense Reasoning About Causality: Deriving Behavior From Structure," *Artificial Intelligence,* 24 (1984), 169-203. Reprinted in *Qualitative Reasoning about Physical Systems,* Bobrow, D. G. (Ed.), North-Holland Press, New York, 1984. Paperback publication by MIT Press, Cambridge, MA, 1985.

8. Kuipers, B. and Kassirer, J.P. "Causal Reasoning in Medicine: Analysis of a Protocol," *Cognitive Science,* 8 (1984), 363-385.

9. Kuipers, B. "The Cognitive Map Overlaps the Environmental Frame, the Situation, and the Real-World Formulary." Commentary on an article by Feldman. To appear in *The Behavioral and Brain Sciences,* 8,2 (June 1985).

10. Kuipers, B. "Is the Theory a Competence Theory or a Performance Theory?" Commentary on an article by Nashner and McCollum. *The Behavioral and Brain Sciences,* 8, 1 (March 1985).

11. Lau, J., Levey, A.S., Kassirer, J.P. and Pauker, S.G. "Using a Semi-Markov Process to Estimate Prognosis of Patients with End-Stage Renal Disease," abstract of a discussion presented at *The Sixth Annual Meeting of the Society for Medical Decision Making,* National Institutes of Health, Bethesda, MD, November 1984. To appear in *Medical Decision Making.*

12. Long, W.J., Naimi, S., Criscitiello, M.G., Pauker, S.G. and Szolovits, P. "An Aid to Physiological Reasoning in the Management of Cardiovascular Disease," *Proceedings of IEEE Computers in Cardiology Conference,* Salt Lake City, UT, September 18-21, 1984, 3-6.

13. Moskowitz, A.J., Dunn, V.H., Lau, J. and Pauker, S.G. "Can 'Hypersimplified' Decision Trees be Used Instead of Markov Models?" abstract of a discussion presented at *The Sixth Annual Meeting of the*

Society for Medical Decision Making, National Institutes of Health, Bethesda, MD, November 1984. To appear in *Medical Decision Making.*

14. Moskowitz, A.J., Kassirer, J.P., and Kuipers, B.J. "Clinical Reasoning versus Decision Analysis," abstract of discussion presented at Sixth Annual Meeting of the Society of Medical Decision Making, National Institutes of Health, Bethesda, MD, November 1984. To appear in *Medical Decision Making,* 4:3, (1985).

15. Patil, R.S. "Is Artificial Intelligence Just Another Pretty Face? An Explanation of the Technology," *Proceedings of the NEHA/AHA First Annual Conference on Hospital Information Systems*, Boston, MA, March 1984.

16. Patil, R. S., Szolovits, P. and Schwartz, W.B. "Causal Understanding of Patient Illness in Medical Diagnosis," in *Readings in Medical Artificial Intelligence,* Clancey, W. J. and Shortliffe, E. H. (Eds.), Addison Wesley, Reading, MA, 1984, 339-360.

17. Sacks, E. "Qualitative Mathematical Reasoning," MIT/LCS/TR-329, MIT Laboratory for Computer Science, Cambridge, MA, May 1985.

18. Szolovits, P. and Pauker, S.G. "Categorical and Probalistic Reasoning in Medical Diagnosis," *Readings in Medical Artificial Intelligence*, Clancey, W. J. and Shortliffe, E. H. (Eds.), Addison Wesley, Reading, MA, 1984, 210-240.

19. Wellman, M. "Bayesian Revision Using Probability Distributions" abstract presented at the *Sixth Annual Meeting of the Society for Medical Decision Making,* National Institutes of Health, Bethesda, MD, November 1984.

20. Wellman, M. "Reasoning About Preference Models," MIT/LCS/TR-340, MIT Laboratory for Computer Science, Cambridge, MA, May 1985.

Theses Completed

1. Sacks, E. "Qualitative Mathematical Reasoning," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, November 1984.

2. Wellman, M. "Reasoning About Preference Models," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

Talks

1. Kuipers, B. "Causal Reasoning and Explanation," *1985 Annual Meeting of the American Educational Research Association*, Chicago, IL, March 1985.

2. Kuipers, B. (Panel member.) "Expert Causal Reasoning and Explanation," presented as part of a symposium entitled "Expert Systems and Cognitive Psychology: Implications for Medicine," *American Educational Research Association Annual Meeting*, Chicago, IL, March 1985.

3. Kuipers, B., Long, W., and Patil, R. (Panel members.) "Deep Models, Qualitative Reasoning, Compiling From Deep Models, Anatomical and Physiological Reasoning," *Artificial Intelligence in Medicine Workshop*, Ohio State University, Colombus, OH, July 1984.

4. Kuipers, B. (Panel member.) "Cognitive Psychology and AIM," *Artificial Intelligence in Medicine Workshop*, Ohio State University, Colombus, OH, July 1984.

5. Kuipers, B. "Qualitative Causal Reasoning in Diagnosis,"
   Revolving Seminar, MIT Artificial Intelligence Laboratory, Cambridge, MA, May 1985;
   Microelectronics and Computer Technology Corporation, Austin, TX, April 1985.

6. Kuipers, B. "Causal Reasoning and Qualitative Simulation," *Artificial Intelligence Symposium*, Department of Computer Science, Columbia University, New York, NY, April 1985.

7. Kuipers, B. "Qualitative Simulation of Mechanisms," *Computer Science Colloquium*, University of Texas, Austin, TX, September 1984.

8. Long, W. "Modeling of Heart Failure," *Artificial Intelligence in Medicine Workshop,* Ohio State University, Colombus, OH, July 1984.

9. Long, W. Panel member. "Diagnostic Reasoning Strategies," *Artificial Intelligence in Medicine Workshop*, Ohio State University, Colombus, OH, July 1984.

10. Long, W. "Artificial Intelligence Techniques Applied to the Cardiovascular System," *Devices and Technology Branch Contractors Meeting for Division of Heart and Vascular Diseases, National Heart, Lung, and Blood Institute,* Rockville, MD, December 1984.

11. Patil, R. (Panel member.) "Computing Environments for AIM Research/End User," *Artificial Intelligence in Medicine Workshop,* Ohio State University, Colombus, OH, July 1985.

12. Patil, R. "An Explanation of Expert System Technology and It's Potentials," Greater Boston Chapter of Data Processing Management Association, Boston, MA, November 1984.

13. Patil, R. "How ABEL really works," seminar at Heuristic Programming Project, Stanford University, Stanford, CA, January 1985.

14. Patil, R. "Combining the Use of Experiential and Fundamental Knowledge through Multi-Level Reasoning," seminar at USC/ISI, Los Angeles, CA, January 1985.

15. Patil, R. "Deep and Shallow Reasoning in Patient Management," *Clinical Decision Making Colloquium,* Tufts-New England Medical Center, Boston, MA, January 1985.

16. Russ, T. "The Arrhythmia Advisor," Medical Systems Division of Hewlett-Packard, Waltham, MA, October 1984.

17. Russ, T. "The Time Dependent Control Structure," Speech Processing Group at Bell Labs, Murray Hill, NJ, November 1984.

18. Szolovits, P. "Types of Knowledge for Reasoning in Medical AI Programs," *National Conference on Medical Informatics*, Marseilles, France, December 1984.

19. Szolovits, P. "Overview of US Medical AI Research Approaches," *National Conference on Medical Informatics*, Marseilles, France, December 1984.

20. Szolovits, P. "Knowledge in Medical Expert Systems," *Islamorada Workshop on Knowledge Base Management Systems*, Islamorada, FL, February 1985.

21. Szolovits, P. "Artificial Intelligence Methods for Medical Decision Making," Harvard Medical School, Cambridge, MA, April 1985.

22. Wellman, M. "Preference Models for Decision Making Programs," Air Force Institute of Technology, Dayton, OH, October 1984.

COMPUTATION STRUCTURES

Academic Staff

J. B. Dennis, Group Leader

Research Staff

W. B. Ackerman
G. A. Boughton
W. Y-P. Lim

Graduate Students

T-A. Chu
G-R. Gao
B. Guharoy
S. Jagannathan
B. Kuszmaul
S. Markowitz
K. Theobald
E. Waldin III
T. Wanuga

Undergraduate Students

C. Goldman
E. Gornish
W. Hamdy
D. Kravitz
B-H. Lim
E. Lyons
D. Marcovitz

Support Staff

N. Tarbet

Visitor

J. E. Stoy

1. INTRODUCTION

The Computation Structures Group is involved in two projects concerned with the design of computer systems using concepts of dataflow program execution and functional programming languages. In our work on static dataflow architecture, the goal is machines capable of high performance, low cost, and improved programmability in the number crunching domain. In the VIM project, the objective is to build an experimental computer system that demonstrates the merit of these ideas for general purpose information processing by a user community.

2. STATIC DATAFLOW ARCHITECTURE

The group's effort on static dataflow architecture has been devoted to developing the design of a practical machine for high performance scientific applications. With the availability of commercial, pipelined floating point chips for 64-bit addition and multiplication, we envision a dataflow processing element capable of performing 10 million floating point operations per second. A dataflow multiprocessor utilizing these elements could comprise 64 processing elements yielding a total peak performance of 640 megaflops. Each processing element would include a large memory for holding array values generated during computation, and would be interconnected with other processing elements by a packet routing network.

3. PROCESSING ELEMENT DESIGN

The major work done in the past year has been the evolution of our architecture for a dataflow processing element into an efficient and practical scheme. The participants include graduate students T.-A. Chu, G.-R. Gao, T. Hegg, W. Lim, K. Theobald, T. Wanuga, staff members W. A. Ackerman, G. A. Boughton and W. Lim, and several undergraduates.

An important change has been a revision of the program execution model for a static dataflow processor. The new model, called "argument fetching," is logically equivalent to the old "token flow" model, but achieves better utilization of memory cycles and space, especially when used with presently available high performance floating point processing chips. In the token flow model, an instruction sent each of its successor instructions a message containing the result. In the argument fetch model, the messages only give an indication to the successor instructions that they should fire. The messages do not contain the result. The successor instructions fetch the result from predetermined locations in a random access memory. This model permits greater separation of the data paths from the instruction control mechanism and allows tighter coupling of arithmetic processors to the memory.

Our current vision of the processing element is shown in Figure 3-1. It is composed of memories, arithmetic units, a Signal System, and an instruction execution control box. While many components of the processing element correspond to components of a

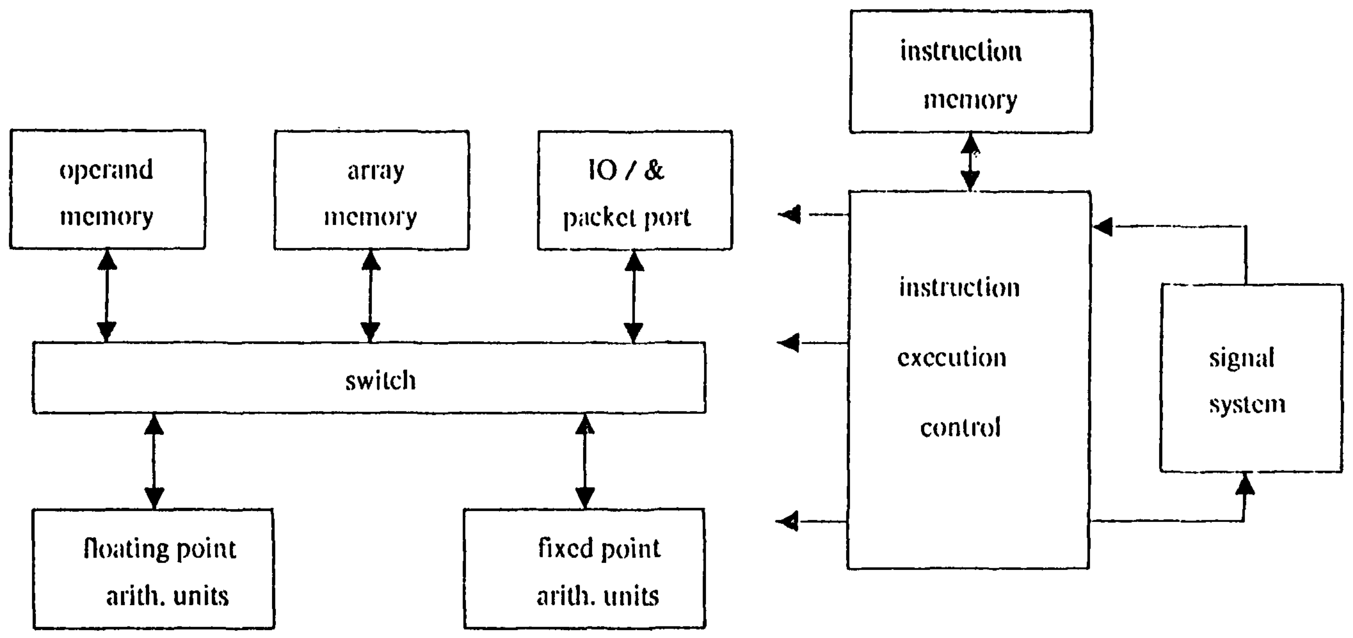


Figure 3-1: Structure of a Dataflow Processing Element.

conventional processor, the Signal System is new. This system handles the bookkeeping required to determine when all operands for a dataflow instruction are available and the instruction is ready for execution.

We have studied possible designs for each of the major components, keeping in mind the approaches that might be used for their fabrication -- gate arrays, standard cell, full custom VLSI, and commercially available chips.

The set of arithmetic processors will include commercially available chips. This is especially attractive due to the availability of pipelined adders and multipliers using the IEEE floating point standard that offer as much as 10 megaflops of performance. For the fixed point units, commercial units may be used, but it is is not certain that these fit well into our design, thus specialized implementations may be required.

The heart of the Signal System is the Enable Memory that records the number of times each instruction has been signalled by other instructions, and determines which instructions have received needed signals and are therefore "enabled" for execution. The Enable Memory will be implemented using custom VLSI; an experimental prototype has been fabricated. The Signal System also contains a signal list memory which holds, for each instruction, the list of instructions to be signalled once the instruction has been executed. One proposal for the format of the signal lists and the functionality of the controller has been developed by D. Marcovitz.

A crucial element is the collection of data paths that form the interconnection of the arithmetic units and the primary memories. The need to support a very high data transfer rate implies that many connections will be needed, and a bit-sliced structure of the data path appears most appropriate.

Several approaches to implementing the instruction execution control are being considered. To minimize the number of overhead instructions that perform no arithmetic, a powerful set of modes for fetching operands and results has been proposed, including modes for entering and removing elements of FIFO queues and for exchanging data packets with other processing elements. In the implementation, the instruction execution control must have access to a pointer memory in which pointers and other types of control information are stored. Since each instruction must go through several stages of processing -- access of control information, address computation, operand fetch, operation, result store -- pipelined operation is essential to achieve a high throughput. Alternative structures for the instruction execution control have been proposed but many details need further work. A semi-custom VLSI implementation is anticipated.

4. THE ENABLE MEMORY

The Enable Memory (EM) is a key component of the static dataflow machine responsible for the sequencing of instructions. It keeps a record of how many signals each instruction cell has received and marks cells that have received the required number of signals. These marked cells are said to be "enabled". One task of the Signal System is to select enabled cells one at a time and forward their numbers to the Instruction Execution Control. It is important for smooth operation of the dataflow machine that each enabled instruction be executed promptly -- that it not happen that other enabled instructions keep jumping ahead, causing a long delay for one instruction. G.-R. Gao and K. Theobald have designed a prototype Enable Memory chip using MOSIS CMOS 3-micron technology. It has about 4000 transistors and a capacity of 128 instruction cells. This prototype is expected to run at about 10 MHz, handling signals from completed instructions at the rate of 2.5 MHz. The chip has been fabricated and is awaiting testing and evaluation.

5. A SELF-TIMED DESIGN METHODOLOGY

T.-A. Chu is conducting doctoral research on a self-timed design methodology for VLSI systems. This methodology is applicable to VLSI and capable of exploiting concurrency. The methodology consists of a high-level organization principle, which was first proposed by J. Dennis, and implementation techniques for self-timed circuits. A graph model called the Signal Transition Graph (STG) has been developed by T.-A. Chu for the specification and direct synthesis of asynchronous self-timed control circuits. A STG is a directed graph with vertices representing the signal transitions at nodes in the corresponding circuit module, and arcs representing the precedence constraints between signal transitions. The signal nodes are the input and output terminals of logic components of the circuit module. The STG can be thought of as a specialization of the Petri Net model in the following way: A Petri Net is a bipartite directed graph consisting of two types of vertices, places and transitions. It is capable of modeling concurrent systems in general. Finite State Machines represent one extreme of the model by using only places and no transitions; hence, it has very limited capability for describing asynchronous concurrent circuits. Our STG model differs from Petri Nets in two important ways: First, it uses only transitions and no places; this type of graph has been known as marked graphs. Secondly, a pair of transitions {+,-} is associated with every node of a circuit, instead of a single transition as used in Petri Nets. These specializations allow one to use STG to specify many types of control circuits easily and also allow direct and efficient synthesis of the hardware modules. STG can model all self-timed control circuits, those data-dependent and data-independent, those with conflicts, and with concurrency.

Two self-timed VLSI chips have been designed using the methodology. One is a CMOS self-timed Two-by-Two Packet Router that works at around 10 MHz, the second is a NMOS FIFO chip that works at around 4 MHz. The router is the constituent component of a communication network for the proposed dataflow computer that our group is developing. A large routing network can be put together using the self-timed two-by-two routers. The advantages of using self-timed components are the modularity of construction and the absence of a global clock. The FIFO chip has a novel organization called a Ring Buffer using distributed control circuits. This organization permits the trade-off between area, latency, and throughput rate. It is believed that these are two of the first truly asynchronous, large-scale systems successfully implemented as single chips.

6. FAULT DETECTION AND RECOVERY STRATEGIES FOR A STATIC DATAFLOW MACHINE

K. Theobald has begun an analysis of fault-detection and fault-tolerance strategies for the static dataflow machine. The goal of the research is the development of general strategies for adding redundancy to detect, and in some cases correct, errors resulting from hardware failures. Since the dataflow machine itself is still in the design phase, the strategies are to be sufficiently flexible to adapt to design changes. Therefore, the research does not extend to the level of circuit or logic design; circuit and logic diagrams are only used when illustrating a general strategy by example. However, the research is comprehensive, ranging from hardware analysis to compilation techniques.

The preliminary phase of the project consisted of determining how the choice of strategies is constrained by the nature of the machine and its intended use. For example, since the goal of the machine is high-speed computing, temporal redundancy (running the same program several times on the same machine, and voting on the results) is unacceptable because the throughput of the machine is reduced two-fold or worse. On the other hand, the machine is not intended for real-time applications, where a momentary delay in a computation might be disastrous (e.g., aircraft control). It may use dynamic redundancy (selective re-execution of only those program steps that were executed erroneously).

The next phase involved reviewing current VLSI technology to determine the major sources of hardware failures in such systems. From this analysis, low-level fault models have been devised and used as primitive failure models to see how hardware failures are manifested at the logical and program levels.

The bulk of the research has been devoted to deciding how and where to add redundancy selectively to the hardware to detect (and possibly correct) errors caused by hardware failures. The system was divided into two parts: the processing elements and the routing network. The processing element was further divided into the functional execution unit and the sequencing control system, whose failure modes are different. Fault-detection and correction strategies have been devised for these subsystems. The only major portion remaining is the perfection of a time-out system to detect a certain class of errors in the Enable Memory (part of the sequencing control subsystem). This effort continues and builds on the work of Leung.

7. THE INTERACTION OF ROUTING NETWORK TRAFFIC AND DATAFLOW INSTRUCTION SCHEDULING

A simulation model of a packet switched routing network for the static dataflow machine, in the form of an indirect binary n-cube composed of two-by-two routers, has been expanded and transposed from a previous model and developed in the simulation language Simula. The new simulation model operates as a pseudo event-driven simulator written in Pascal, using a more detailed model of the two-by-two router. The expanded model of the two-by-two router models the timing of the actions of its individual components (buffers, multiplexors, demultiplexors, input and output pins, etc.).

The expected performance of the routing network, in terms of throughput, latency, blocking of network inputs, and buffer utilization, is being examined with this simulator. Studies have been made concerning the placement of buffers at various places within the router, e.g., on the inputs, internal to, or on the outputs of the router. Also examined were the effects of different patterns of communication across the network, e.g., does each processing module talk to only a few other modules or many other modules. Currently, the effects of different chip sizes and packet sizes, i.e., how many bytes per packet are being investigated.

Studies so far have shown that the placement of buffers internal to the routers (as opposed to on the router inputs or outputs) can greatly improve several aspects of the network performance. Different patterns of communication across the network have been shown to produce widely varying levels of network performance, depending on which processing modules communicate with each other. A simple method for assigning code blocks to processing modules, such that the network always performs quite well, has been developed. The effect of increasing packet size seems to be only a slight degradation in network performance (less than 15% in throughput and latency for a doubling of the packet size).

8. THE PROGRAM TRANSFORMING COMPILER

A compiler that produces efficient dataflow machine code from programs written in the applicative language VAL is crucial to the success of the static architecture in large scale scientific applications. To keep the architecture simple and efficient for computations involving large arrays of numerical data, most of the decisions about resource assignment (allocation of data structures to space in array memory and of dataflow instructions to processing elements) have been entrusted to the compiler.

To accomplish an effective resource allocation, the compiler must transform the program so that its structure is a good match to the processing power and memory space of the target machine. Hence, the number of processing elements and the sizes of the instruction and array memories will be parameters of the compiler. Global program transformations are needed because the several sections of a large program must be capable of operating together in a way that allows full utilization of performance without requiring inordinately large amounts of memory for intermediate results. It is reasonable to attempt such global program transformations because VAL is a functional or applicative language, and therefore interactions among program parts occur only at points that are evident from the syntactic structure of the program -- the impossibility of "side effects" removes the major difficulty that inhibits use of global optimizations in compilers for conventional languages.

The conceptual basis for such a compiler has been laid down in the doctoral thesis of Ackerman, which develops loop unfolding and array interleaving optimizing transformations for dataflow programs, and in the graduate research of Gao which explores the concept of *pipe-structured* programs that support the pipelined execution of dataflow machine instructions. Accordingly, compilation of a large program will proceed by the following steps:

1) Syntax analysis; type inference and checking; static semantic checks: Converts program into abstract syntax tree.

2) Graph generator: Converts program from abstract tree into "dataflow graph" representation.

3) Simple machine independent optimizations: removal of records; constant folding; common subexpressions.

4) Linker: Combines modules to form complete program for a task to be performed by the machine.

5) Analysis I: develop space/time estimate for presentation to user—for the complete program if possible, otherwise, by modules with indication of where the analysis is incomplete and why.

6) User interaction: The user augments the data generated by the compiler's analysis and confirms the transformations to be applied.

7) Transform I: Array interleaving and loop unfolding: This implements the basic space/time tradeoff involved in matching a problem to the machine.

8) Transform II: Pipelining: This implements sections of the program as "maximally pipelined" code blocks wherever profitable.

9) Analysis II: Prediction of code performance and memory requirements on the basis of the selected program structure (from analysis and advice).

10) Code generation: generate code for the target machine.

The program transforming compiler will accept valid VAL programs and produce target code for static dataflow computers. It will attempt to generate *pipe-structured* object code whenever the source code allows. For some programs optimal code of this form can be produced algorithmically, given parameters characterizing the machine. In these cases, nearly full performance of the machine will be realized for any program that allows sufficient concurrency without exhausting the machine's memory. For other programs, program transformations will be used to convert the code into a form that can make better use of the target machine. The choice of transformations to be applied will be guided by the compiler's structural analysis of the source program, and the programmer's advice given in response to the compiler's analysis. The advice will concern the degree of interleaving to be used for arrays and the degree of unfolding of loops to be used by the compiler. This advice is needed for programs in which the compiler's analysis is incomplete due to the absence of such information as: the number of cycles performed by loops (termination test depends on a computed value); size of arrays (index range is not a compile-time constant); the frequency of selecting arms of a conditional (when the computations are of very different form for the different arms). We envision that the advice file will have the form of a tree representation of the program structure with annotations that constitute the advice.

W. Ackerman has begun implementation of optimizing transformations for dataflow programs. At this writing, the global unfolding of record structures has been accomplished. This sort of global optimization would be considerably more difficult to implement for a language that was not applicative.

9. THE VIM PROJECT

The VIM project aims to develop an experimental computer system based on principles of data-driven instruction execution and functional programming languages. The project is unique in striving for a system that will serve multiple users with a degree of semantic coherence well beyond what contemporary computer systems are able to offer. It also differs from other efforts to build systems to support functional programming in that the issues of efficient execution of functional programs over a hierarchical memory are addressed and solutions developed. The system uses a base language that is a form of acyclic dataflow graph, and a user language VIMVAL that is an extension and revision of the functional language VAL.

9.1. VIMVAL

Effort on refining the design of VIMVAL during the past year has focused on the role of modules and the means of binding free names in modules to form executable program units. In the VIM system, program modules are written in VIMVAL and are compiled separately. The programmer creates a runnable program by using the *bind* command of the VIM system shell to associate values with the free names of some module. The resulting *closed* module defines a value which may be bound to a free name of yet another module. In this way, a runnable program unit that is a tree (actually a directed acyclic graph) of linked modules can be constructed. The absence of cycles in the linked program is guaranteed by requiring that only closed modules be bound to free names.

Usually the value defined by a (closed) module will be a function, but we extend the notion of module to include any value representable as VIMVAL text such as an integer, a string, or any data structure built of arrays and records. Thus, a module can define any constructible data object which has a VIMVAL type specification. In this way, traditional data objects and functions are treated as equals. This feature avoids a problem present in many programming environments: how to handle the global constants of a program. Instead of having to set up some kind of global environment for such values, they can be collected in a VIMVAL record that is the value of a closed module. Adjusting any of the "global" parameters is simply a matter of recompiling the parameter module and rebinding the modules that use them.

The restriction to acyclic structures of closed modules does not rule out recursion and mutual recursion. Such functions may be expressed within a single VIMVAL module as a set of mutually recursive function definitions. Such a set should be conceived of as a unified entity. Viewed from outside, the recursive set is an just an ordinary function definition. This idea is one embodiment of the principle that modules are independently written program units that should be comprehensible without reference to the internal workings of other modules.

In the VIM system, we require that all code be type checked before execution, providing a guarantee that no type conflicts will occur during program execution.

Within a VIMVAL program module most type specifications are optional, since the compiler can infer the type of an object from its context. However, in certain situations the compiler may be unable to infer the type of an expression. In such cases the programmer must insert a type specification. At other points a programmer may wish to insert a type specification just to improve program readability.

However, omitting type specifications from the module header would be inconsistent with the principle that a user need not inspect the module body to determine its legitimate uses. Therefore, each module has a header that gives type information for each argument, each result, and each free name of the module. This information must be sufficiently complete that the user of the module may construct a module that uses the given module and know that no type conflicts will occur when any legal binding is performed to create a runnable program unit.

Since complete type information is not always needed in module headers to support this need, the VIM system provides for *partial type definitions*. For example, a programmer may wish to specify that an argument or result is an array, but not the type of the array elements, because their type is immaterial. Then the type specification *array[t]* may be used where no type specification is given for *t* indicating that it is a *type variable.* One use of partial type specifications is to constrain the range of types so the compiler may produce more efficient code. Another use is to specify polymorphism, as in the header of a sort program

```
    function Sort ( Data: array[value], Keys: array[tag], Test:
        function (x,y: tag) returns(boolean))
        returns(array[value], array[tag])
```

where *value* and *tag* are type variables.

Various language constructs for expressing the computation of array objects have been investigated. These computations can normally be expressed using simple array operations, such as *append,* in a tail recursive function definition. However, in many cases where the computation is expressed as a tail recursive function definition, it is difficult for the compiler to generate code that uses storage efficiently. In addition, the programmer may find it awkward or inconvenient to write the computation in this form. We have found it desirable to include a special language construct for these cases that permits easy generation of good code and that provides the programmer with a simple, semantically clean language construct.

A related area of investigation involves general recursion in VIMVAL. General recursion allows recursive equations to define data structures. Some array computations can be considered as special cases of general recursion. For example, if there is a recurrence relationship among the elements of an array, then the entire array may be easily defined by a set of equations that specify this recurrence relationship. However, several difficulties are raised by the inclusion of general recursion in the language, among them the capability for the programmer to express circular data structures. A problem of current interest is finding an efficient implementation of general recursion within the framework of the VIM system.

9.2. Operational Models for VIM

Two projects concerning the implementation of large data structures and the backup of information in VIM are being completed as master's theses. So that these studies may be done in terms of a precise understanding of the program execution model of VIM, formal operational models for VIM have been formulated by B. Guharoy and S. Jagannathan. The first model, **L1**, specifies the semantics of the dataflow graphs that represent programs executed by VIM. In **L1**, there is no explicit modeling of hardware elements of the implementation -- the model is abstract and suitable for understanding the correctness of a compiler from the VIMVAL language into dataflow graphs. This model, and the other models to be mentioned, consist of a set of system states and an interpret function that defines a set of non-deterministic transitions between states. In **L1**, a system state has three components: a set of function activations; a heap which contains, among other things, structure values and function templates; and the set of enabled instructions (enabled nodes of the dataflow graph). This latter set contains those instructions in the activations that have received all operands and signals necessary to enable execution. There is no concept of memory in this model; it treats structure types as sets and structure values are simply elements in these sets. The interpreter is a state transition function mapping from a state and an enabled instruction to a set of possible next states. The rationale for developing this model is two-fold: First, it provides a simple but precise mathematical description of system operation as a formal interpreter of dataflow program graphs. Second, this model serves as a suitable basis, through augmentation and refinement, for rigorously defining aspects of system implementation, as described below.

For the study of the implementation of data structures in the VIM memory hierarchy, B. Guharoy has developed a second operational model, **L2**. This model includes a storage model that reflects a physical storage hierarchy consisting of a main store and disk. Here the program representation is the same as in **L1**, but the heap is modeled by a collection of fixed-size memory *chunks* which are either *accessible* (meaning they reside in fast (main) memory) or *inaccessible* (meaning that the chunk is held only on the disk memory). The model includes the reference count scheme by which chunks are identified as unreachable and therefore the space occupied is available for reallocation.

In his master's thesis, B. Guharoy establishes correctness of a set of algorithms for an implementation of general array operation using chunk allocation in a hierarchical memory. This is achieved by showing that **L2** and **L1** have equivalent behavior using the proof method developed by D. Berry and C. McGowan.

9.3. Backup and Recovery in VIM

In his master's degree research, S. Jagannathan has developed the design of a backup and recovery system for the VIM computer system. The goal is to design procedures to guarantee the security of *all* accessible information in the system against loss or corruption as a result of hardware failure. The mechanisms to achieve data security on conventional systems are based on the existence of a file system that uses memory space independent of that used by the basic program execution facility, and exploit the fact that updates of files take place relatively infrequently and usually replace the file in its entirety. This implementation has the unfortunate consequence that all activity performed since the last update action is lost when the system fails. The backup and recovery system proposed for VIM takes advantage of the novel aspects of VIM -- its powerful applicative base language and the data-driven architecture of the system.

In VIM, long-lived objects are bound to names in an *environment* associated with each user. The backup system preserves the environment by copying values onto the backup store as they are defined in response to user commands. The information placed in backup store by this mechanism is called *quiescent data.* The backup system also maintains an *activation log* of all active computations in the system. When the system fails, recovery is accomplished as follows: First, the recovery system uses the quiescent backup data to restore a past system state; then interpreting the activation log will bring the system forward to the its state immediately before the failure. The activation log is a directed tree in which nodes correspond to function activations, and arcs represent caller/callee relationships. The thesis shows how the execution rules for certain VIM instructions may be augmented to incorporate the necessary building of the activation log. It is also shown how the recomputation of values can be avoided by storing pointers to computed results in the activation log, thus preventing long recovery times for long running commands.

The backup store consists of a *stable storage* device used to hold the activation log, and a large mass storage device, such as tape. Information on stable storage is immune to the effects of hardware failure and is easily accessed and updated. Stable storage is presumed to be too expensive to hold the entire backup state, so a cheaper storage device, such as redundant tapes, is needed to hold quiescent data.

S. Jagannathan has designed the backup system to handle the creation of stream elements produced in a demand-driven fashion and has included optimizations to handle tail recursive functions.

10. FUNCTIONAL LANGUAGE IMPLEMENTATION ON THE CONNECTION MACHINE

Programs written in an applicative language exhibit a large degree of parallelism, and the connection machine is a highly parallel machine. In his master's thesis, B. Kuszmaul has hypothesized that the connection machine will perform well running such programs, and that there is much to be learned about applicative programming from a high performance testbed. He is studying several implementation schemes for applicative languages through simulation on the connection machine. These include static dataflow, VIM-style dynamic dataflow, and combinator reduction. For this study, Kuszmaul has developed a new programming model for the connection machine well suited to writing highly parallel interpreters for applicative languages. The objective is to evaluate the suitability of the connection machine for each of the implementation schemes mentioned above. VIM is an experimental computer system based on principles of data-driven instruction execution and functional programming languages. The project is unique in striving for a system that will serve multiple users with a degree of conceptual coherence well beyond what conventional computer systems are able to offer. It is also distinguished from other efforts to build systems to support functional programming in that the issues of efficient execution of functional programs over a hierarchical memory are addressed and solutions are developed. The system uses a base language that is a form of acyclic dataflow graph, and a user language VimVal that is an extension and revision of the functional language VAL.

Publications

1. Ackerman, W.B. "How to Design Simulatable CMOS Integrated Circuits," VLSI Memo No. 85-237, MIT,Department of Electrical Engineering and Computer Science, Cambridge, MA, March 1985.

2. Chu, T.-A. "Design of a CMos Self-timed Two-by-Two Packet Router," Computation Structures Group Memo 242, MIT Laboratory for Computer Science, Cambridge, MA, December 1984.

3. Dennis, J.B. "Modeling the Weather with a Data Flow Computer," *IEEE Transactions on Computer,* (July 1984).

4. Dennis, J.B. "Data Flow Computation," *Control and Data Flow:Concepts of Distributed Programming.* Manfred Broy, (Ed.) Springer-Verlag, Berlin, Heidelberg, New York, NY, 1985.

5. Gao, G-R. "Pipelined Mapping of Homogeneous Data Flow Programs," Proceedings of IEEE International Conference on Parallel Processing," Cambridge, MA, August 1984.

6. Lim, W. Y-P. "A Design Methodology for Stoppable Clock Systems," Computation Structures Group Memo 240, MIT Laboratory for Computer Science, Cambridge, MA, August 1984.

Theses Completed

1. Boughton, G. .. "Routing Networks for Packet Communication Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1985.

2. Guharoy, B. "Data Structure Management in a dataflow Computer System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

3. Markowitz, S. "VLOE: A Val Language-oriented Editor," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1984.

Theses in Progress

1. Gao, G-R. "A Pipeline Code Generation Scheme for Static Data Flow Computers," Ph.D. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1986.

2. Jagannathan, S. "Guaranteeing Data Security in a Dynamic Data Flow Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1985.

3. Theobald, K. "Adding Fault-Tolerance to a Static Data Flow Supercomputer," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1985.

4. Wanuga, T. "Routing Network Performance in a Static Data Flow Machine," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1985.

Talks

1. Dennis, J.B. "Data Flow Computation," six lectures at International Summer Course on Control Flow and Data Flow: Concepts of Distributed Programming, Marktoberdorf, Germany, August 1984.

2. Dennis, J.B. "Data Flow Computing" RIACS Workshop on Evaluation Study of dataflow Computation, Moffett Field, CA, September 1984.

3. Dennis, J.B. "On the Form and Use of Distributed Computer Systems," Keynote address given at IEEE Conference on Distributed Systems, Denver, CO, May 1985.

4. Gao, G-R. "Pipeline Mapping Scheme for Array Operations on Static Data Flow Computers," IBM, Yorktown Heights, NY, August 1984.

5. Gao, G-R. "Principals of Pipeline Code Generation Scheme for Static Data Flow Computers," RIACS Workshop on Evaluation Study of dataflow Computation, Moffett Field, CA, September 1984.

DISTRIBUTED COMPUTER SYSTEMS

Academic Staff

D. Clark, Group Leader

Research Staff

L. Allen	M. Lambert
M. Greenwald	E. Marten

Graduate Students

R. Baldwin	K. Sollins
J. Gibson	L. Zhang
T. Ng	

Undergraduate Students

J. Barba	R. Reuss
D. Bridgham	J. Romkey
J. Lunny	M. Rosenstein
C. Newman	C. Warack

Support Staff

E. Felix

1. INTRODUCTION

During the 1984-85 period, there has been considerable rearrangement of this group's activities, with the completion of a major project and the departure of several group members. Some new projects have been initiated as well; the various efforts are discussed in the sections below.

2. THE SWIFT OPERATING SYSTEM

The purpose of the SWIFT project was to demonstrate an operating system suitable for the efficient implementation of such programs as network protocols. The project started about three years ago and was essentially finished during this year. A running kernel for the system was demonstrated and several programs were implemented to run on it, such as a version of TCP/IP.

This project has been described in detail in previous progress reports, and the final results are reported in a paper submitted for publication [1].

A brief summary of the conclusions are as follows:

- The program structure demonstrated in SWIFT, in which lower level programs call back to programs invoking them from above, is an effective program structure;
- CLU is a reasonable language in which to program such constructs; the run time type-checking required in this case is not inconsistent with the general compile-time checking philosophy of CLU, and the efficiency is adequate;
- Using type-checking as a protection tool for a single shared address space is effective; and
- Use of monitors to control sharing of state among processes is reasonable.

The only current effort underway on the SWIFT system is the development and demonstration of a fault recovery mechanism. It can reconstruct the system after a failure of one or more processes and in many systems, storage associated with a failed process is reclaimed by discarding the address space of the process. Yet SWIFT operates with all processes in one shared address space, so this reclamation technique is ineffective. Instead, special tools are needed to identify and reclaim data objects associated with the failure and are being pursued by J. Gibson as a Master's thesis.

3. PROJECTS IN DISTRIBUTED COMPUTING

3.1. Non Real-Time Delivery Protocols

Most of the current network protocols are based on the idea that the two ends of a connection are in real-time communication with each other, exchanging information back and forth as part of an ongoing transaction. There is an alternative paradigm, which is that communication is not done in real time, but rather by messages that are sent from one end to the other (perhaps with substantial delay) as in computer mail. For personal computers, which are often disconnected from the network, this alternative communication paradigm seems very natural. Unfortunately, this staged delivery paradigm makes unusable most current algorithms for distributed synchronism, such as two-phased commit. For this reason, it is necessary to rethink the class of algorithms which can be utilized in this context.

T. Ng has been studying possible synchronization algorithms for use in staged delivery situations. He has developed a number of techniques for serialization and recovery, which depend on permitting the application to see intermediate states of the computation, and requiring the application to assist in rolling back the computation in unusual circumstances.

3.2. Distributed Mail

As part of the research in non real-time protocols and the personal computer, several members of the group have developed a protocol for allowing a human to process computer mail from an IBM PC. A central node, called the repository, holds the master copy of the user's mail, while the PC provides the user interface. In fact, the user can use more than one PC to read mail (perhaps one in the office, one at home and one in the briefcase). The repository keeps track of the current state of these various PCs, and attempts to synchronize the local copy of the mail database within each. Of course, since the machines are not always on the network running the mail software, this synchronism cannot be perfect; we have developed the protocols to deal with the resulting inconsistencies. This mail program is now in initial test by a few members of our group, and should be available for more general use during the summer.

3.3. Distributed Calendar Maintenance

Another non real-time protocol was developed by M. Rosenstein to synchronize several versions of an appointment calendar located on separate PC class machines. Again, the various machines were assumed to be available only intermittently. In this case, the synchronization was not done using a master copy; instead the various copies tried to keep each other in step by sending special mail messages to each other. A preliminary version of this application will be completed over the summer.

4. PROJECTS IN NETWORK PROTOCOL DESIGN

4.1. Unified Stream Protocol

D. Clark developed a protocol called Unified Stream Protocol, or USP, which is designed to facilitate interworking among different protocol families. It specifies a uniform set of semantics which a client at the application layer should assume, and shows how to provide that semantics using any of the popular transport level virtual circuit protocols. This proposal is now under consideration by various of the network communities.

4.2. Reliable Circular Multicast Protocol

J. Romkey developed a protocol called Reliable Circular Multicast Protocol, or RCMP, as an undergraduate thesis. Its purpose is to provide a logical connection between a number of machines in such a way that when any of the machines sends, all are guaranteed to receive. This protocol was intended for small messages and for a small number of hosts, such as a teleconference or a computer game. The first implementation done was for the IBM PC; it shows good performance, with 5 to 10 ms. delay in delivery per node in the connection.

4.3. Bulk Data Transfer

D. Clark developed a protocol for bulk data transfer, called NETBLT. NETBLT provides a reliable delivery of data, but differs from traditional reliable protocols such as TCP in a number of ways. In particular, its user interface and flow control differ and, instead of a window scheme for flow control, it uses an open loop rate controlled flow. Implementation of this protocol was done by D. Bridgham as an undergraduate thesis.

4.4. Congestion Control and Routing

L. Zhang continued her work on congestion control algorithms for network and internetwork management. Using a simulator, she has demonstrated that the current algorithms in the DARPA Internet suite do not work well. She has proposed some alternative mechanisms for congestion control, which have been incorporated into the NETBLT protocol mentioned above.

5. DISTRIBUTED NAME MANAGEMENT

[This work was initially done in the Computer Systems Structures group]

K. Sollins completed her doctoral thesis entitled "Distributed Name Management" during this past year. Since this work has been discussed in the previous two progress reports, it will only be summarized here.

The problem being addressed in this research is the design of a naming facility achieving the following goals: First, two functions on names must be supported--accessing a named object and acting as a place holder for the named object. Second, it must be possible to share those names. Third, communication of the names as well as communication by use of the names must be possible. In addition, feasibility of implementation is a goal. In this research a name is defined to be an object that can be associated with another object and has an equality operation defined on it. The assumed system model is a loosely coupled distributed system defined in previous reports as a *federation*.

The research addresses this problem with: (1) a detailed analysis of the naming problem and the nature of names themselves; (2) a proposal for a set of mechanisms that addresses the problem above, including the proposal of two new types of objects and the mechanisms for their use; and (3) two examples of uses of the model. The model consists of private views (known as *aggregates*) of shared, local namespaces (known as *contexts*) allowing shared use of names and supporting shared responsibility for management of the namespace. In addition, the model provides for the acceptance and deletion of names in stages within a context. A context is parameterized by both a definition of the set of stages through which a name will pass in the process of being accepted and procedures for acceptance and deletion. Joint management as exemplified in the procedures of acceptance and deletion form an important part of this research effort. The two specific domains to which the model is applied are electronic mail, in the form of an implementation, and programming support environments, in a detailed study.

The contributions of the research include an investigation into the nature of names, an analysis of naming as a social process especially recognizing both the joint management of names by the users of those names, the fact that acceptance and possibly deletion occur in degrees, and the proposal for a mechanism to address these issues.

6. EXTENDED REACH NETWORKS

[This work was originally done in the Computer Systems and Communication Group]

6.1. A High-Speed Packet-Switching Network for CATV Systems

D. Feldmeier is writing a Master's thesis that proposes a preliminary design for a high-speed, packet-switched network for residential CATV (Community Antenna Television) systems to provide digital data communication to the home. This network consists of stations that transmit data on an upstream cable channel to a network controller that repeats the data to all stations via a downstream channel. A design goal for this network is to change the existing cable system as little as possible, ideally adding only a network controller at the headend, and interface cards at the subscriber end. A related goal is complete compatibility with existing cable equipment, television sets, and cable channel allocation.

Professionals are beginning to use personal computers extensively for their work not only at the office, but in the home as well. For the computer-oriented professional, a personal computer at home offers many advantages: evening or weekend work is convenient and an office in the home allows a worker to avoid commuting and interruptions during the day. As the price of personal computers continues to drop, the office in the home will become more popular, but it becomes realistic only if the home computer can be effectively linked to the office computer system. To work efficiently, a professional working at home needs the same services that are available at the office. Services are provided to computers in the office by a local area network, which supplies high-speed communications over a limited geographical range at low cost. The network connects computers to each other and to services too expensive to provide to each individual, such as high-quality printers, mass storage, and high-speed data processors.

High-speed, low-cost communications must be brought to the home to support the services provided in a local area network environment. Communication from the home today is done with the telephone system, which provides low-speed communication and wastes telephone system resources. A CATV system can be the basis of high-speed, low-cost data communications from the home with few changes to the cable system itself.

Although all the aspects of the network will be discussed, the primary emphasis of the thesis is the upstream transmission method and the network access scheme. Because of the converging tree topology of a CATV system, the noise arriving at the CATV headend is the amplified sum of all the noise over the entire system; consequently this noise makes upstream transmission difficult. In addition, the frequencies used in the upstream direction are routinely used for short-wave communication and the CATV distribution plant make an effective antenna at these frequencies. To combat these noise problems, several upstream transmission techniques are discussed, including spread-spectrum modulation.

The access scheme for a CATV network is another difficult problem. Local area network access schemes such as Carrier Sense Multiple Access with Collision Detection (CSMA/CS) become inefficient with increased propagation delay, making them unsuitable for a CATV network with its large propagation delay. Satellite access schemes are more efficient, but have slow response times. A variation on a satellite access scheme is suggested as an alternative that would work well on a CATV network.

6.2. Inter-Organization Networks

D. Estrin is writing her doctoral thesis on the interconnection of computer networks across organization boundaries. *Inter-Organization Networks* (IONs) are used to support person-to-person communication; exchange of CAD/CAM data, software modules, or documents; input to an order/entry or accounting system; or use of shared computational resources. This inter-disciplinary research addresses usage control requirements and mechanisms for IONs, and the organization implications of the new medium.

When distinct organizations interconnect their internal computer networks, they encounter new access control requirements. Each organization would like to discriminate between external and internal users of its computer-based facilities. However, unlike some security requirements, the goal is not to prevent outside access altogether. One obvious solution is to increase access control on all internal facilities. However, controls tailored to restrict outsiders may interfere with internal communication and resource sharing. In addition, it is impractical to require modification of all internal systems. Finally, given the decentralized administration of many networks, it is difficult to assure conformance with new policies and interconnections. The obvious solution is objectionable and we conclude that administrators and users of strictly-internal resources should not be required to take action in order to be protected from external access.

A scheme has been developed that satisfies these constraints using non-discretionary control mechanisms in all entry and exit points to an organization's internal network (i.e., the Inter-Organization Network gateways). In order to accommodate different policy requirements, the non-discretionary control mechanisms employed differ in subtle ways from previous non-discretionary mechanisms. The approach presented has implications for network interconnection protocols because of the need to associate policy information with each packet, message, or connection. Alternatives to packet-level interconnection, and related implementation issues, are evaluated in this light.

Participating organizations can use the speed, incremental cost, range of capabilities, and automatic nature of computer-based communications to support new interchange patterns. A descriptive model has been developed which explains how the use of Inter-Organization Networks affects interchange in several domains (e.g., manufacturers-subcontractors, research, customer-supplier). The model describes the technical characteristics of IONs that underlie changes in the economics of inter-organization-communication; the opportunities for ION participants to adopt more efficient and effective communication and interchange patterns -- more intense communications of greater scope; the opportunities for ION participants to expand and enhance cross-boundary activities -- more cross-boundary activities and with a larger number of outside organizations; and the problematic effects of ION use on interchange and cross-boundary activities--more segmented and penetrating interchange, restricted sets of interchange partners, and more explicit administrative and technical controls on cross-boundary flows.

The model can be used to both explain and inform the design, deployment, management, and regulation of IONs. To evaluate the model, we conducted an empirical study of ION use among commercial and university research and development laboratories.

References

1. Clark, D. "The Structuring of Systems Using Upcalls," MIT Laboratory for Computer Science, Cambridge, MA, submitted for publication.

Publications

1. Estrin, D. and Sirbu M. "Standards," to appear in *International Encyclopedia of Communications,* Norwood, NJ, Ablex, 1986.

Talks

1. Allen, L. "The Use of Upcalls in SWIFT," *IBM/ACM SIGOPS Workshop on Operating Systems in Computer Networks*, Zurich, Switzerland, January 29, 1985.

2. Allen, L. "The SWIFT Operating System," The Computer Laboratory, Cambridge University, Cambridge, England, January 31, 1985.

3. Estrin, D. "Non-Discretionary Controls for Inter-Organization Networks," *1985 IEEE Symposium on Security Privacy*, Oakland, CA, April 1985.

EDUCATIONAL COMPUTING

Academic Staff

H. Abelson
A. diSessa, Group Leader
S. Papert
S. Weir

Research Staff

E. Lay

Graduate Students

M. Eisenberg
L. Morecroft
E. Long
F. Turbak

Undergraduate Students

J. Boyle
C. Humphreys
J. Roschelle
R. Oellette

Support Staff

P. Davis
M.Palmgren
J. Karaslaanian

Visitors

W. Higginson
W. McKay
M. Kliman

1. OVERVIEW

The commitment of the Educational Computing Group to pursue a broad approach to educational problems and innovation has continued this year. Most of our activities involve some combination of work in the areas of: (1) Technology: building and adapting the best that contemporary hardware, software, and computer science has to offer to education; (2) the Educational Context (Schools, Teachers, Students and Curriculum): maintaining contact with the educational system as it stands so that our innovation is responsive; and (3) Cognitive Studies: simultaneously attacking the long-term and deep issues of learning.

This year, the greatest emphasis was on technology. Our Boxer project, which aims to provide a very broad spectrum of capabilities to non-professional users, is the bread and butter of the Group. We continued to develop the implementation and specification of the system. In particular, we added a conceptually simple but broadly functional graphics facility. As we close in on settling the basic semantics and functionality of the system, we look forward to developing applications by ourselves and in cooperation with teachers and curriculum developers, and also to more serious studies of the learnability of the system.

In the area of educational context, our impact on the outside educational world will shortly be significantly enhanced by another major publication, S. Weir's books *Cultivating Minds: A Logo Casebook.* In addition, several small curriculum development projects progressed, and cooperative work with Papert's Laboratory is developing several major sites in which to experiment with innovations "in the real world."

As usual, cognitive studies were spread among other activities, but continued to focus on learnability of computational systems, understanding, and developing children's spatial and other intuitive knowledge.

2. TECHNOLOGY

2.1. Boxer

System Semantics -- During the year, steady progress has been made working out the basic computational semantics of the system, and gradually converting our experimental system toward the agreed-on specification. The current major implementation task is to rebuild the Boxer interpreter according to our latest specification, which is largely being carried out by E. Lay. Here, by way of example, we give an overview of data objects in Boxer.

One of our major decisions, to forego distinct atomic data types, has been implemented. This leaves only one kind of data object. Data boxes may contain any text or other Boxer structure, including data and procedure definitions. These may be accessed in any number of ways to achieve different functionalities. Data boxes may be accessed as a whole, by item number, by row, by row and column (matrix), or by named subpart (record). Data boxes that contain procedural definitions can by used with Boxer's remote procedure execution capability to achieve functionalities usually associated with actor for data abstraction capability.

Reference to data boxes normally works via an *information transmission* metaphor, that is, all references imply a (virtual) copy so that users do not see any sharing. It is our conviction that this is by far the simplest way for beginners to think about passing around and using data.

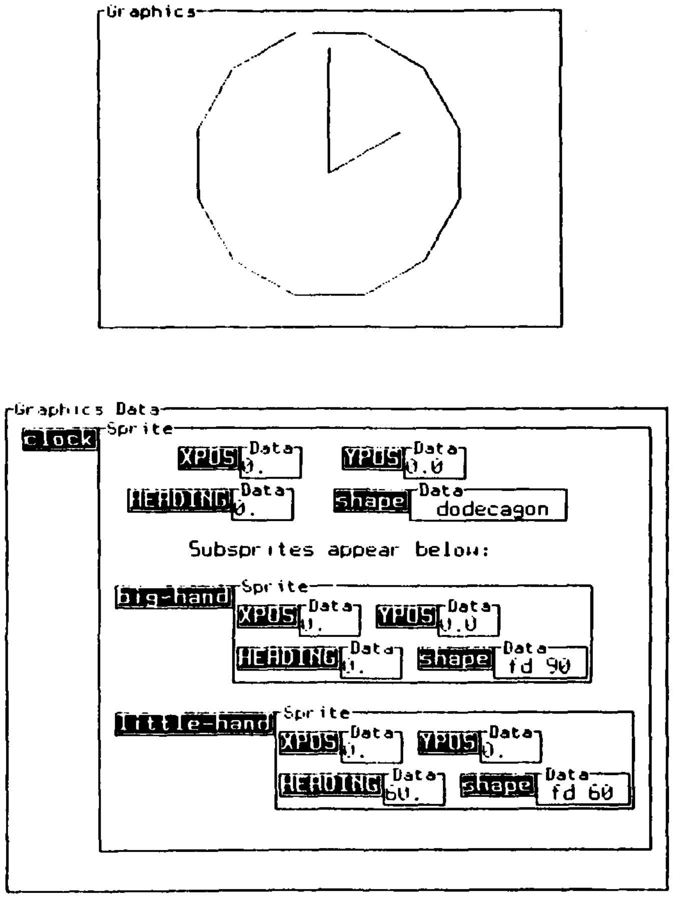
A port is a view on some data box, called the port's target, that lives in any arbitrary part of the user's spatially organized Boxer universe. Ports are kinds of boxes aimed at implementing sharing, for more advanced users. They are "sticky" in the sense that any reference to a port, e.g., by name or by selecting a subpart, also becomes a port reference to (part of) the target of the port. Thus, ports support the complementary reference metaphor of *object reference*, which conventional pointers also implement. This means that mutation to a port changes the data for every part of Boxer that has a port reference to that same data. If one needs to pass actors as data objects, one actually uses a port to the actor (data box) in order that any state change in the actor is seen by a part of the system that shares references to that actor.

Graphics -- As an example of increased functionality of Boxer achieved this year, we briefly describe a much expanded graphics sub-system implemented by J. Roschelle.

The graphics system seeks to extend the turtle graphics of Logo. It has generalized graphical objects called *sprites* that live inside of *graphics* boxes. (The reason that we have graphics isolated in a special kind of box is that all other parts of Boxer are manipulated in the same way, with the extended text/box editor. Graphical objects in their ordinary form cannot be manipulated in that way, hence we chose to isolate that different interaction mode spatially.) Like turtles, sprites can move and turn; their shape can be set to be that described by any turtle program and may include text (and in the future, shading); their size can also be adjusted separately.

Sprites can have sub-sprites, for example, the sprite in Figure 5-1 has two sub-sprites which are hands of the clock. The user can move the clock as a whole, TELL CLOCK FORWARD 100; or he can talk to the subparts, for example, TELL BIG-HAND RIGHT 6, to turn the hands of the clock. Sprites are sensitive to the mouse cursor in that they highlight themselves when the cursor moves over them. In that case, pressing mouse buttons causes user-defined procedures to be activated in the sprite. This means it is very easy with this graphics system to create information appliances that work using only the mouse, much as Macintosh and similar software does. Indeed, one of the aims of the graphics system is to support other user-definable modes of interaction defined by the Boxer editor. Special commands like FOLLOW-THE-MOUSE extend the functionality of mouse operations on sprites. Sprites can draw with a pen, like a turtle;

they can stamp their shape on background in their graphics box; and they are sensitive to other sprite's touch, which simplifies writing may kinds of programs involving interacting objects.



clock: XPOS Data 0. YPOS Data 0.0 HEADING Data 0. shape Data dodecagon

big-hand: XPOS Data 0. YPOS Data 0.0 HEADING Data 0. shape Data fd 90

little-hand: XPOS Data 0. YPOS Data 0. HEADING Data 60. shape Data fd 60

Figure 5-1

Graphics boxes actually have two forms. The primary form is purely graphical, as they are mostly intended to be used. But in order that every part of the system be visible and manipulable in the same way, graphics boxes can also be turned into *graphics data* boxes, in which case the sprites contained in the graphics box have the usual box structure that is Boxer's hallmark. In this form, graphics boxes and sprites can be named and manipulated with the editor in the same way as all other kinds of boxes. The second box in the figure is the same graphics box turned into its "data" form. Note the sub-sprites that appear inside the clock sprite. The spatial relation of containment means "part of" for graphical objects. In the data form, one can see the position coordinates of the sprites, and one can see other attributes (boxes) that are either built-in to all sprites (like HEADING) or have been added. Naturally, when a sprite is given a command like FORWARD 100, its position coordinates change on the screen if the graphics box is in its data form. And if one changes the coordinates directly with the editor, the position of the sprite in the graphics box changes. One will see that change as soon as one returns the graphics-data box back to its purely graphic form.

User Interface -- In order to explore other modes of interaction with Boxer, R. Ouellette attached a touch screen to our development machine, and implemented software features in Boxer that allow all editor functions and any user-defined function to be operated by poking the screen. A novel feature of the system is that it supports a mode in which the part of the screen around the touched point is expanded in size, so that the physical resolution of touching implement (e.g., finger) is no longer an issue. Single character resolution is very easy to achieve.

Videotape -- With the help of D. Smith of Umbrella Films, a second videotape of Boxer was completed this year. The tape concentrates on showing how Boxer can change the meaning of programming from an esoteric activity for professionals and experts to something that students, teachers and pedogogically (but not technologically) sophisticated developers can use to develop and adapt their own applications. The tape shows a personal journal, a Boxer environment for teaching Boxer, several "information appliances" (little programs that work like miniature spread-sheets or other concrete data processors) and a video book on physics which includes both text and a working optics simulator that can compute the view of an object through an arbitrary user-defined optical system.

Boxer video II was shown at the San Francisco meeting of ACM SIGCHI in 1985. Selections will appear in the conference proceedings video.

2.2. Understandability of Computation

Two master's theses exploring issues in building easier-to-understand computational systems are near completion.

M. Eisenberg has designed and implemented a version of Scheme, called Bochser, that uses some ideas imported from Boxer to visually represent much more of the system than can be seen in ordinary implementations of the language. His implementation, on a Symbolics Lisp Machine, takes particular care to make environment structure explicit. Preliminary use of the system by beginning Scheme programmers is quite encouraging.

F. Turbak has designed and partially implemented a system that begins with the same intent, to make computation more understandable by representing it visibly, but he has concentrated on the dynamic structure of the language. His assumption is that one can and should incorporate all the major aspects of a mechanistic model of the language's operation into its visual representation. By literally watching the system operate, users are much more likely to learn a coherent model of the system's operation. A particularly novel feature of the system called Grasp enables learners to begin with objects called activations, something like a function invocation, and only later come to abstractions of activations, the equivalent of function definitions. One of the attractive aspects of this design decision is that, in terms of activations, the entire history (and even possible futures!) of a program's execution can be leisurely inspected in a static structure. The system also implements reversibility: one can watch programs executed forward or backward in time.

3. EDUCATIONAL CONTEXT

Getting our ideas out to the world of education is a particularly important subtask for the group. As highlighted in the overview, S. Weir's about-to-be published book *Cultivating Minds: A Logo Casebook* makes a significant contribution to the group's already substantial output. In the book, some of the major and enduring conceptual issues of learning are addressed from the particulars of experiences with children learning with Logo. Individual styles and competences are treated with particular care and concern.

On other fronts, Dr. Weir is just completing a two-year project run cooperatively with the Carroll School for disabled youngsters. The project focused on analyzing and capitalizing on strengths in spatial reasoning shown by dyslexic children. Data is currently being analyzed.

Curriculum Projects: A number of small projects are in progress to develop and test materials for use with students in teaching a variety of subjects.

*L. Morecroft is nearly finished with a master's thesis that developed materials to teach the physics concept of frames of reference and relative motion through the use of Logo-implemented microworld. She is currently field testing the materials with high school students.

*C. Humphreys has developed a Logo microworld and written materials to teach children the elements of sailing. The work is in collaboration with a Boston Public Schools project centering around the acquisition of a sailing ship for use by the school system.

*E. Long has begun developing materials to teach signal processing to elementary school students using some hardware developed by the Technical Education Research Center. The hardware allows Apple microcomputers to perform such operations as capturing, analyzing (e.g., Fourier transform), modifying and regenerating sound.

Cooperative Work with Arts and Media Technology: Several members of the Educational Computing Group are cooperating with S. Papert's Learning and Epistemology Group in the Arts and Media Technology Laboratory to develop sites in the Boston Public Schools in which to experiment with high densities of computation.

The major thrust is a $1,000,000 effort sponsored by IBM called Project Headlight. The basic idea of these efforts, aside from having an in-school lab site to develop and test ideas, is to simulate the kind of computational resources, both in number and in kind, that will become generally available with the declining cost of machines within the next five years. Only in such environments can one experiment with a major overhaul of the curriculum and style of teaching that technology portends.

Logo 84 and Logo 85: A major activity of the Group in terms of maintaining contact with the growing Logo community is organizing an annual Logo conference at MIT. The bibliography of the conference now shows about 160 books and over 700 articles published on Logo. The conference this year is expected to attract over 1000 people from all over the world.

4. COGNITIVE STUDIES

Other than the work done by Sylvia Weir in the project with dyslexic students at the Carroll School, and a continuing concern for mental models of computation in the Boxer, Bochser and Grasp Systems, there is one project in progress with major emphasis on cognitive analysis. M. Kliman, a visiting graduate student from the University of Edinburgh, is developing a knowledge map of elementary school students with respect to the task of understanding weighing and balancing. The work is uncovering remarkably rich structure and partial understandings and interventions that can be made to help children along the road to a full understanding of that subject. Possible future directions include automating the knowledge map and developing computer-based materials such as a formal system (miniature expert system) in which children can implement working models of balancing based on their current state of understanding.

FUNCTIONAL LANGUAGES AND ARCHITECTURES

Academic Staff

Arvind, Group Leader	R. Nikhil

Research Staff

R. Iannucci	J. Pinkerton

Graduate Students

M. Beckerle	G. Maa
S. Brobst	G. Papadopoulos
A. Chien	K. Pingali
D. Culler	R. Soley
S. Heller	K. Traub
R. Iannucci	B. Vafa
V. Kathail	

Undergraduate Students

G. Bromley	F. Park
D. Clarke	J. Sheffield
E. Hao	J. K. Soon
F. Herrmann	Y. M. Tan
R. Indech	R. Wei
R. Katz	J. Weisz
C. Lee	C. Wong
D. Morais	S. Younis
G. Ng	

Support Staff

S.M. Hardy	N.F. Tarbet

Visitors

B. Blaner
E. Hagersten
D. Lowther
M. Mack
N. Skoglund

1. INTRODUCTION

The Functional Languages and Architectures Group is pursuing two interrelated projects, namely, the Tagged-Token Dataflow Machine and the Multiprocessor Emulation Facility (MEF). The goal of the Tagged-Token Dataflow project is to demonstrate the feasibility of general purpose parallel machines by simulation and emulation. The goal of the MEF project is to construct a "sandbox" to facilitate research and development in parallel architectures and languages. Following this introduction, the report is divided into two major sections describing the ongoing projects.

Our cooperation with IBM on both these projects has increased significantly over the past year. Several joint meetings to discuss the technical details of the two projects were held, and as a consequence of these meetings mid-course maneuvers were required. For example, we have decided to switch to non-IBM technology for the packet communication network for the MEF. On the dataflow project, the discussions resulted in identification of two sub-goals which may be achieved by somewhat different strategies. These sub-goals were (1) the demonstration of the scalability of the Dataflow Machine, and (2) the investigation of a dataflow machine which will require the same amount of hardware as a high performance sequential computer for the same level of performance. Our immediate attention has been focused on the first sub-goal, and consequently changes in the architecture or the compiler which are required only for the demonstration of the second sub-goal, have been deferred.

A lively and productive IBM-MIT workshop was organized last April in Essex, CT, to review multiprocessor research at the two institutions. We expect such interactions and cooperation with IBM will continue at an even greater pace in the coming year.

1.1. Dataflow Project Overview

Since the success of parallel machines will depend on the effective programmability and efficient utilization of resources, the dataflow project has been deeply concerned with high level language support and resource management issues. We have been experimenting with Id, a functional language, for a number of years and have finally produced a documented version of the Id-to-dataflow graph compiler, which can be used as a service program by us and other researchers. We have now entered what may be called the second phase of development for both the language and the compiler. On the language side, some ground work has been done to make polymorphic type-checking and reference count garbage collection possible. A proper semantic view of I-structures, the main data structure in Id, also seems to be emerging. Fortunately, this new view of I-structures, even though fundamentally very important and novel, should have minimal impact on the compiler. The compiler is being rewritten to improve its internal modularity to make possible easy implementation of both anticipated and unanticipated changes in Id. This new phase of language and compiler development is relying, much more heavily than in the past, on the solid theoretical work that has been done at MIT and elsewhere in the area of functional languages.

We think that programmable and scalable machines should be such that a change in machine configuration, e.g., changing the number of processors or memory modules, interconnection topology, does not require the programmer either to rewrite or recompile application programs. Thus, sophisticated runtime resource managers are essential for the Dataflow Machine. We have made significant progress in efficiently executing loop programs; however, further progress in this area is crucially tied to the large scale simulation and emulation experiments. Our simulator, which runs on IBM machines, is in the final stages of debugging and documentation, and should allow us and IBM researchers to experiment with resource management policies in the near future. Large scale emulation experiments will have to wait until the MEF is fully functional.

1.2. MEF Overview

The past year has been one of continual upheaval for the Multiprocessor Emulation Facility (MEF). We have endured one change of CPU, a major change in personnel, several changes of technology, several changes of design strategy, and, most importantly, two changes of name. Despite these changes, or perhaps because of them, we have moved forward and will soon see first fruits. Spurred on by the arrival of funds from DARPA, the Hardware Laboratory was completed and is now in use.

New CPU: At the beginning of this reporting period, we had planned to construct the Emulation Facility out of Symbolics 3670 processors. We have subsequently switched to Texas Instruments (TI) Explorers. The rationale for the change was based partially on price, but predominantly on the open architecture of the Explorer coupled with the fine cooperation we have received and expect to receive from Texas Instruments. Currently, we have eight Symbolics Lisp Machines connected by Ethernet, and two preproduction models of TI Explorers.

Personnel: During the previous reporting period, we had proposed a joint study with IBM whereby they would provide design engineers to work at LCS on the packet switch. The first three engineers have been with us for a year now and have made significant contributions in the areas of high speed serial link protocol development / verification, central switching mechanism design, creation of a library of very high performance programmable logic array cells for custom VLSI, partitioning, packaging, and analysis for the packet switch. In addition, they have researched the existence of standard parts within IBM which will greatly simplify the design of this switch.

Technology: As an adjunct benefit of the Joint Study, IBM was to provide design tools and manufacturing for the gate array logic used in the packet switch. Due to confidentiality constraints, we have declined this offer of technology and have instead negotiated for chip fabrication through LSI Logic Corporation.

Design Strategies: Prior to the switch-over to TI machines, we were designing two different network cards for the 3670: the first was an adaptation of the BBN Butterfly's circuit switch. The second was a packet switch of our own design (both reported previously). The TI machine is smaller than the Symbolics machine in many respects -- most importantly, the circuit cards are smaller and the potential for expansion (card slots, power) is much more limiting. Consequently, we have re-partitioned the design into three circuit cards: (1) a *packet switching network adapter*, (2) a *circuit switching network adapter*, and (3) a *NuBus channel adapter.* The channel adapter, or NuCA, provides direct-memory access between the NuBus and up to three network adapters which may be circuit switches, packet switches, or a combination.

Name Changes: In that the design effort for the MEF is directed at interconnecting an array of Lisp machines and that these machines have been named after trees, e.g., Cherry, Mahogany, Live-Oak, Netleaf-Hackberry, etc., a proposal was put before the Group to name ourselves *Project Tanglewood.* The Boston Symphony, however, refused to permit our use of the name. We had temporarily changed names, only to have to revert to the old (and uninspiring) "MEF".

2. TAGGED-TOKEN DATAFLOW PROJECT

2.1. Compiler Progress

S. Heller and K. Traub have debugged and ported V. Kathail's original Id-to-Graph Compiler to Lisp Machines. Now simply called the Id Compiler (Version 1), it has been used successfully to compile several hundred programs since its release in January 1985. The compiler output has been verified both by execution on the simulator and emulator, and by detailed hand inspection. The largest program that this compiler has processed is *Simple,* a 1200-line hydrodynamics program from Lawrence Livermore Laboratory. Simple consists of 19 individual Id procedures, which in turn compile into 55 code blocks representing the procedures and the loops nested therein. The total size of the compiled code is about 86,000 bytes of code for the Tagged Token Dataflow Architecture (TTDA), representing over 10,000 dataflow operators. The current release of the compiler implements *bounded loop schema* which was developed this year and is discussed in Section 2.4

Coincident with the release of Version 1 of the Id Compiler was the publishing of the *ID Compiler Users's Manual,* which contained the first definitive description of Id syntax, as well as instructions for using the compiler. A revised and expanded version of the user's manual has been published as CSG Memo 248.

The past year we have been increasingly aware of the need to adapt the Id Compiler to changing requirements. We are now interested in particular in changing the basic schema to incorporate new ideas about resource management, in changing the object code generation to reflect the evolution of the TTDA, in optimizing code through common subexpression elimination and code-block merging, in enhancing the language with a deductive type-checking facility to permit efficient management of structures, and in performing a host of other compiler-related experiments. To meet these requirements, we have begun work on Version 2 of the Id Compiler. The major design goals of Version 2 are to:

- Provide a well-documented modular structure that allows the easy addition and removal of compiler phases, such as type-checking or common subexpression elimination. In this way, experimental compilation techniques can be introduced at each step of the compilation procedure.

- Make the major transformation phases of the compiler (parsing, abstract graph generation, machine code graph generation, and assembly) as specification- driven as possible. This allows easy modifications to syntax, schema, and machine architecture without the need for detailed knowledge of data structures and algorithms internal to the compiler.

We are now designing Version 2 and plan to complete it by June 1986.

2.2. Simulator Progress

The simulation facility has evolved into a solid testbed for the TTDA. The simulation software has stabilized and we are proceeding with experiments aimed at enhancing our understanding of the architectural tradeoffs to be made in the construction of a dataflow multiprocessor. Experiments conducted by S. Brobst, D. Culler, and B. Vafa have helped us to identify conditions under which the machine will deadlock, bottlenecks in the execution pipeline of the processing elements, and the need for controlling the amount of parallelism exploited during program execution. Programs that we are currently running on the simulation facility include matrix manipulations, the *Livermore Kernels*, and parts of the *Simple Code* developed at Lawrence Livermore National Laboratory. The resource requirements within the TTDA necessary for executing the partial-differential equation simulations implemented by the Simple Code are a subject of current investigation.

In support of these simulation experiments, R. Wei, with the help of others, has built a sophisticated user interface on top of the software implementation of the TTDA. This interface allows complete configuration of a dataflow multiprocessor, including the number of processing elements, relative speeds and technology of the hardware stages in a PE, and resource management policies. Simulation experiments consist of both a machine configuration and a specification of the source program to be executed. These experiments can be run either in a remote batch mode via a simulation server on its own virtual machine, or in a user's own virtual machine to facilitate interactive resource management decisions, as well as debugging.

In addition, a suite of data collection and analysis tools have been developed by R. Katz as a means for evaluating the performance of various implementations of our dataflow multiprocessor. A database system specifically built for the simulation facility was put in place early last fall. The data collected includes performance profile information, as well as a summary of all relevant benchmark characteristics of an experimental run. A number of statistic-calculating packages are available for analyzing this performance data. C. Wong has developed a facility for graphic output of experimental data, and it is one of the most useful tools we have for providing insight into the dynamic execution characteristics of the TTDA. Most recently, an interface between Hewlett-Packard plotters and the IBM 4341 has been developed to facilitate high-quality graphic representation of performance data.

S. Brobst, D. Culler and G. Maa are currently in the midst of a software engineering effort to tighten up the interfaces between simulation modules, as well as to strengthen the abstractions between the architectural, scheduling, and data collection components of the simulator. This will be the last stage of our software development effort before we release the simulation facility software to the IBM Research Center at Yorktown Heights. In the near future, we will be upgrading the simulation hardware resources to an IBM 4381 in an attempt to increase the performance of simulation experiments.

Most of our experimentation to date has been directed at verifying the simulator and the code produced by the compiler. Some of the benchmarks we employed are mentioned above. A variety of experiments were conducted to verify claims concerning processor performance. For machine configurations of up to 16 PEs (the largest we are currently able to simulate on the IBM 4341), good scalability was demonstrated. We also observed that independent threads of computation interlace nicely within the processor pipeline, as expected. Until the PE becomes fully utilized, additional threads of computation can be processed with no increase in processing time. A number of experiments were conducted to verify claims made by D. Culler on the resource requirements of loops. We observed that (1) the resource requirements of loops do increase essentially linearly with the amount of unfolding, (2) for simple inner loops pipeline constraints serve to restrict the unfolding, (3) for loops that involve a significant amount of parallel computation in each iteration, the unfolding is dramatic and independent of the amount of parallelism the machine can exploit, (4) loops can be constrained to achieve substantial reductions in resource usage without performance degradation. For large programs, it appears necessary to constrain program unfolding to keep the resource requirements within reason. We are currently modifying the compiler and the simulator to support new graph schemata which allow parameterized. This unfolding will allow us to study strategies for controlling dataflow programs in the large.

2.3. Resource Management

Our recent work in resource management focuses on certain differences between the U-interpreter and the TTDA. The model of computation embodied in the U-interpreter is extremely powerful; it places minimal constraints on the execution order, and thereby allows maximal parallelism. It imposes no resource constraints whatsoever; an unbounded number of activities may be performed concurrently, unbounded queueing of tokens on the arcs is permitted, and activity names may grow, i.e., if a *greedy* schedule is followed, programs unfold in accordance with whatever parallelism is present. The U-interpreter also provides a convenient framework for reasoning about dataflow programs and for proving properties of programs. In part, the ease with which the U-interpreter lends itself to formal analysis stems from its rather idealized view of computational resources. The U-interpreter also provides a convenient framework for reasoning about dataflow programs and for proving properties of programs. The TTDA captures the essential execution mechanism of the U-interpreter, allowing programs to unfold in accordance with whatever parallelism is present in the program, but imposes rather strict resource constraints; tokens must reside in the waiting-matching store, activity names are represented by fixed-size tags, etc. D. Culler and B. Vafa have attempted to overcome the differences between the model and the machine in regard to resources through a mixture of techniques: program analysis to deduce the resource requirements of sizable portions of dataflow programs, dynamic resource management to distribute work over the machine without overcommitting individual resources, and program transformation to make programs more suitable for execution on the TTDA.

Last year we reported on a dynamic resource management system, developed as part of the simulation and emulation efforts. That system provides a means of distributing work and data over the machine (using fairly straight-forward load-leveling techniques) and takes care of allocating and deallocating all explicitly managed resources. Deallocation of resources requires embellishing the program graphs to detect when certain resources are no longer in use. Basically, this requires augmenting the graph for each code-block with arcs so that a certain node is guaranteed to be the last activity in the code-block.

The hardware allocates implicitly certain resources (notably, waiting-matching storage) on demand. When a code-block is invoked, a certain collection of processing elements are designated to perform the invocation. These processing elements must provide storage for waiting tokens generated in the course of the invocation. The load on the waiting-matching store is particularly crucial; if the matching store fills up and the overflow is used, performance degrades considerably; if the overflow store fills up, the program deadlocks. One of the motivations for developing the simulation and emulation was to determine how large the waiting-matching store would have to be for large programs to run efficiently on the machine. It became clear that the resource management system should not ignore the load on the waiting-matching stores when distributing work over the machine. However, in order to account for the load placed on the waiting-matching, the token storage requirements of code-blocks must be determined in advance. As the token storage requirement depends on the order on execution, it was necessary to deduce from the program graph the worst-case storage requirement under all possible legal execution orders. The basic Idea is to model the space of legal configurations as a linear program and to solve for the maximum number of tokens on the arcs. For acyclic blocks without conditionals, this can be solved efficiently. For blocks with conditionals, tight bounds are NP-complete, but approximate bounds are easily computed with branch-and-bound techniques.

Loops present another class of problems. One of the virtues of the U-interpreter and the TTDA is that loops unfold automatically, exposing what parallelism is present. Iterations are distinguished by an *iteration number* carried as part of the activity name or tag. However, this automatic unfolding introduces certain complications. Since tags are of fixed size in the TTDA, the iteration number field may overflow. Moreover, the resource requirements of an activation (in particular, the token-storage requirements) increase linearly with the number of concurrent iterations. To overcome these difficulties, we desired a mechanism for controlling the extent of unfolding and automatically recycling iteration numbers. The key result is that a loop has bounded unfolding if and only if the graph forms a single connected component. Working from this, D. Culler developed techniques for augmenting loop code-blocks so that the degree of unfolding is controlled by a single run-time parameter.

Finally, we considered measures for constraining the automatic unfolding of programs in order to reduce the resource requirements. A greedy scheduling strategy tends toward breadth-first unfolding, which offers maximal parallelism, but has large resource requirements. A depth-first strategy reduces the resource requirement, but also reduces the amount of parallelism. Of the variety of techniques developed for controlling the evaluation strategy, the most promising relies on the resource manager to queue invocation requests when the machine is saturated with work.

2.4. Reduction Systems and Combinators

During the last year, we have continued our investigation of reduction systems and combinators. In particular, K. Pingali has been examining the relationship between *supercombinators* and dataflow. A supercombinator can be thought of as a closed λ-abstraction whose definition is of the form $\lambda x.\lambda y....\lambda w.e$ where e is a λ-expression without any nested λ-abstractions. The number of leading λ's in the definition of a supercombinator is called the *arity* of the supercombinator. There are many techniques, such as *λ-lifting* and *mfe-abstraction*, for converting conventional functional language programs into supercombinatory form, i.e., into a set of supercombinator definitions and an expression involving supercombinator applications.

To evaluate supercombinator programs, the definitions of the supercombinators can be used as rewrite rules -- any application of a supercombinator to as many arguments as its arity can be replaced by the body of the supercombinator with the arguments substituted in. This is a straightforward reduction implementation of supercombinators. It was noticed by Johnsson that is possible to do better than this. Each supercombinator definition can be compiled into code for a stack machine (or for that matter any sequential machine), so that many intermediate steps of the reduction can be avoided altogether. He calls this process "short-circuiting" graph reduction.

We have found that the essence of short-circuiting the reduction of a supercombinator application is captured by converting the body of the supercombinator into *continuation-passing* style. The code generated for the stack machine is an implementation of this transformed program. This view of short-circuiting has two advantages. First, it relates the work done by Johnsson to the work done by Sussman and Steele, Wand and others on compiling Scheme-like languages. Second, it is easy to extend this idea to parallel implementations of functional languages. We are currently trying to understand dataflow implementations in this light. If successful, this will permit us to relate dataflow to reduction.

3. MULTIPROCESSOR EMULATION FACILITY PROJECT

3.1. The Hardware Laboratory

During the previous reporting period, R. Iannucci, J. Pinkerton, G. Papadopoulos and others put together the plan for a new MEF Hardware Laboratory (Tanglewood Design Note 4). This year we saw the dream come to life in the form of a 600 sq. ft. room on the second floor which now provides:

- Two AED 767 color VLSI layout stations running CAESAR, HPEDIT, and (soon) MAGIC, interfaced to a Hewlett-Packard 8 pen plotter.
- Two Apollo design workstations for schematic capture, simulation, timing verification, and component placement. We are also adding a 500 MB file server to our Apollo ring network.
- A drafting table (when all the fancy electronic stuff fails).
- Four fully instrumented logic design / debug stations, each with an oscilloscope, a logic analyzer, function generator, counter, digital multimeter, programmable power supplies, an instrumentation computer, and a full set of hand tools. In addition, we have a high performance digitizer and a time domain reflectometer.
- A stockroom.
- A VLSI microprobe station.

We also recently learned that our request for a grant of $200,000 worth of additional Apollo workstation hardware has been approved; we will take delivery on the equipment before the end of the summer.

3.2. Packet Switch

B. Blaner, M. Mack, and D. Lowther joined the MEF team from IBM during the summer of 1984. During the last year, they have made noteworthy progress on the development of the packet switch. Major accomplishments include

- Development of a protocol for the high speed serial links of the switch, which is an improvement over the echo-acknowledge protocol previously reported: Blaner's contributions here are very significant; he will be continuing his work on protocol verification over the next few months.

- Development of the central switching mechanism for the packet switch card: Mack's high-level and detailed designs are sufficiently far along that we expect to release a chip description to the manufacturer by the end of the summer. The result will be an 8-in, 8-out, 4-bit-wide crossbar on a single 84-pin chip. Performance will far exceed the requirements of the packet switch.

- Partitioning of the card's function down to the chip level: The partitioning is such that the high-speed serial logic is entirely modular and is separate from the central switch. This makes it conceivable to design both smaller and larger packet switches which are protocol compatible with the existing design; such redesigns would be nearly trivial. This opens the possibility of using the serial link as a building block for many other applications, e.g., dataflow processor cards, backbone local area networks. We expect to have a working link in prototype form by early Fall 1985, with the cMOS version to follow in the second quarter of 1986.

- Selection of our target technology: We are currently planning on using LSI Logic's LL7600 series and LL5220 series for the high speed serial logic and the central switch, respectively. We will also be using several components available through IBM to make the design task simpler.

- Design of a set of very high performance PLA cells for custom VLSI: We have just received our first test chip from MOSIS. Measurements will be taken within a month.

Performance of the switch is dominated by two major factors: the speed of the serial links (target is 32 Mbits/sec) and the latency of allocating a path through each switch upon the arrival of the head of a message. The switch is optimized for high-traffic message-passing communication where total throughput is more important than per-message latency. We are currently re-investigating design enhancements that will noticeably improve our ability also to support a *remote memory reference* model of communication, i.e., shorter messages, with round-trip delay more important than total message throughput.

During the next year, the packet switch team will continue to grow. It is likely that we will have our first prototype card by summer of '86; we expect to be running packet switch / NuCA test shortly thereafter. Between now and then, a significant amount of effort will be required to implement the necessary NuCA microcode, Explorer microcode, and LISP code to make efficient data transfer accessible from Lisp.

3.3. Circuit Switch and NuCA

We have adapted the BBN Butterfly circuit switching network as an interim communications medium until the packet switch becomes available. The primary contributors to this effort have been G. Papadopoulos, G. Bromley, C. Lee, and F. Park. The BBN switch node is a 4x4 4-bit serial, globally synchronous crossbar. The raw bandwidth of each link is three megabytes per second, and a maximum utilization of 20% of this raw rate is expected. This yields an aggregate useful bandwidth of over one gigabit per second for a 32-processor configuration.

We have added error detection, noise immunity, and electrical isolation to make the BBN switch more suitable for a higher integrity and more physically dispersed multiprocessor. The circuit switch can be attached directly to the TI NuBus, controlled by Lisp or Explorer microcode. In addition, S. Younis has engineered a high precision distribution network designed to deliver the globally synchronous six megahertz clock to 64 TI Explorers, within a 25 nanosecond maximum skew.

To off-load the Lisp processor from low level message formatting and protocol, we are developing a microcoded I/O processor, the NuBus Channel Adapter, or NuCA. G. Papadopoulos, E. Hagersten and N. Skoglund (the latter two on loan from Ellemtel, Sweden) have developed the NuCA. The NuCA provides an intelligent DMA interface between the NuBus and a local bus. The local bus is a simplified byte-wide block oriented bus consisting of an input and output link, each capable of delivering or receiving 10 megabytes per second, and an asynchronous Spy bus for diagnostics, error recovery, and configuration. The NuCA will support both the circuit and packet switches simultaneously. Internally, the NuCA provides two 5 Mips 64-bit micromachines, 4096 bytes of high speed FIFO buffering, 64K bytes of scratch pad memory, in addition to an autonomous, multiported NuBus master interface capable of four-way request interleaving and burst transfers without Lisp processor intervention.

In Fall 1984, we demonstrated a re-engineered circuit switch prototype for the Symbolics 3600. The circuit switch in its standalone NuBus configuration should enter production by Fall 1985, becoming available to MEF users during the first quarter of 1986. A NuCA prototype should be operational by the first quarter of 1986.

3.4. MEF Software

The MEF software development in progress is comprised of three parts, (1) generic software for emulating multiprocessor architectures on the MEF, (2) the software for the TTDA experiment, and (3) a general-purpose text illustration program called ILLUSTRATE.

Generic Software -- Early in the year, we found that the low speed of the dataflow emulation (less than 100 dataflow instructions per second per processor) was at least partially due to design decisions taken in the MEF generic architecture emulation code.

R. Soley (the original author), M. Beckerle, and D. Morais did some to speed tuning up this substrate. Due to these efforts, we are now emulating dataflow instructions at the rate of about 500 instructions per second per processor. Scaling of this speed by increasing the number of processors, however, is severely bandwidth-limited by the current interconnection hardware (10 Megabit ethernet running Chaos and TCP protocols).

In order to support stepped-up use of the MEF substrate software at MIT and elsewhere, R. Soley's thesis covering the theory and practice of its use is now available for distribution. LCS/ TR-339 describes the basic abstractions of the MEF software, as well as extensions and examples of its use. Among our future plans are the upgrading of the MEF substrate software for more speed, support of the Texas Instruments implementation language for portability, and support of the NuCA network interface. This work will be performed over the next six months by R. Soley.

Dataflow Emulation Experiment -- In Spring 1985, we decided to halt work temporarily on the dataflow emulation software. First, we knew that the upcoming switch to the TI Lisp Machines from the Symbolics machines would require some changes to the software. Since the dataflow emulation software is in need of some major revisions, it seemed best to wait until the new software environment was available before undertaking major changes.

In addition, within the next few months there are two sources for potential modifications to the architecture. At meetings with researchers from I.B.M., we formed a committee to simplify the instruction set for the architecture. Also, with the simulator now producing real results, we felt that the simulation experiments might lead to some design changes for the architecture. Therefore, rather than completing the emulator for the current architecture, it seemed more practical to wait for a new specification, and in the mean time, to concentrate on learning as much as possible from simulation experiments.

The ILLUSTRATE Program -- During the past year, work has continued on the ILLUSTRATE graphical illustrate program. D. Morais, the original author, carried out most of it, with the assistance of D. Clarke and R. Soley.

ILLUSTRATE is a highly interactive object-oriented graphical illustration program, designed primarily for adding illustrations to theses, reports, books, etc. Built originally around the Alto Draw concepts, design ideas have been added by many members of the group. Running on the Symbolics 3600 Lisp Machine processor, Illustrate is becoming quite popular at MIT and other sites around the country. Over the next year, Illustrate will also be ported to the TI Explorer processor.

References

1. Cardelli, L. "ML Under Unix," Technical Report, AT&T Bell Laboratories, Murray Hill, NJ, 1983.

2. Culler, D.E. "Resource Management for the Tagged-Token Dataflow Architecture," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1985.

3. Milner, R. "A Theory of Type Polymorphism in Programming," *J Comp & Sys Sci*, 17, (1978), 348-75.

4. Mohamed-Ali, K.A. "Distributed Garbage Collection Algorithms for a Loosely-Coupled Multiprocessor System," CSALAB Working Paper 1983-03-09, Royal Institute of Technology, Stockholm, Sweden, 1983, 51.

Publications

1. Arvind and Culler, D. E. "Final Report: Program Decomposition for Multiple Processor Machines," CSG Memo 244, MIT Laboratory for Computer Science, Cambridge, MA, December 1984.

2. Brobst, S. A. "Tagged-Token Dataflow Architecture Simulation Facility User's Manual," CSG Memo 250, MIT Laboratory for Computer Science, Cambridge, MA, March 1985.

3. Culler, D.E. "Resource Management for the Tagged-Token Dataflow Architecture," MIT/LCS/TR-332, MIT Laboratory for Computer Science, Cambridge, MA, January 1985.

4. Heller, S. and Traub, K. "Id Compiler User's Manual," CSG Memo 248, MIT Laboratory for Computer Science, Cambridge, MA, May 1985.

5. Pingali, K. and Arvind "Efficient Demand-Driven Evaluation (I)," TOPLAS, May 1985.

6. Soley, R. M. "A Third Opinion on Dataflow Machines and Languages," CSG Memo 241, MIT Laboratory for Computer Science, Cambridge, MA, October 1984.

7. Soley, R.M. "Generic Software for Emulating Multiprocessor Architectures," MIT/LCS/TR-339, MIT Laboratory for Computer Science, Cambridge, MA, June 1985.

8. Traub, K. "An Abstract Architecture for Parallel Graph Reduction,"

MIT/LCS/TR-317, MIT Laboratory for Computer Science, Cambridge, MA, June 1984.

Theses Completed

1. Bacon, S. "A Supercombinator Compiler for Scheme," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

2. Bromley, G. "Waiting/Matching for Tagged-Token Dataflow Architectures," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

3. Culler, D. E. "Resource Management for the Tagged-Token Dataflow Architecture," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1985.

4. Ng, G. W. "A C-Language Instrument Control Function Laboratory," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

5. Soley, R. M. "Generic Software for the Emulation of Multiprocessor Architectures," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

6. Vafa, B. "A Resource Management Policy for the Tagged-Token Data Flow Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

7. Wei, R. C-S. "The Design of An User Interface Facility for the Tagged-Token Dataflow Architecture Simulator," S.B. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

8. Weisz, J. "A Hardware Description Translation System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

Theses in Progress

1. Beckerle, M. J. "Logical Structures for Functional Languages," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1985.

2. Brobst, S. A. "Token Storage Requirements in a Dataflow Supercomputer," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1985.

3. Hughes, G. "Wavefront Synchronism," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1986.

4. Pinkerton, J. "A High-Speed Serial Link Optimized for Packet Switching," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1985.

5. Pingali, K. "Design and Implementation of Dataflow Languages with Streams," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1986.

Talks

1. Arvind. A series of four lectures on "Computer Architecture" at DEC, Shrewsbury, MA, September and October 1984.

2. Arvind. "Why Dataflow Architectures?" Honeywell Symposium, Minneapolis, MN, September 1984.

3. Arvind. "Why Dataflow Architectures?" IBM-Poughkeepsie 40th Anniversary Symposium, Kutsher's Country Club, NY, November 1984.

4. Arvind. "Fundamental Issues in the Design of Multiprocessor Systems," ElectroTechnical Laboratory, Tsukuba, Japan, November 1984.

5. Arvind. "The MIT Tagged-Token Dataflow Machine: Current Status," ElectroTechnical Laboratory, Tsukuba, Japan, November 1984.

6. Arvind. "Why Dataflow Architectures?" University of Minnesota, Minneapolis, MN, January 1985.

7. Arvind. "Dataflow Experiments on the Multiprocessor Emulation Facility," MIT Laboratory For Computer Science, Cambridge, MA, February 1985.

8. Arvind. "The Goal of the Tagged-Token Dataflow Project," IBM-MIT Workshop, Essex, CT, April 1985.

9. Arvind. "Why Dataflow Architectures?" MIT Department of Aerodynamics and Astrophysics, Cambridge, MA, April 1985.

10. Arvind. "Why Dataflow Architectures?" University of Massachusetts, Amherst, MA, May 1985.

11. Brobst, S.A. "Waiting-Matching Requirements of the Tagged-Token

Dataflow Dataflow Architecture," Harris Corporation, Advanced Technology Division, Melbourne, FL, January 1985.

12. Brobst, S.A. "Application Domain of the Tagged-Token Dataflow Architecture," Harris Corporation, Advanced Technology Division, Melbourne, FL, January 1985.

13. Brobst, S.A. "The Next Generation of High Performance Computer Systems," MIT Sloan School of Management, Cambridge, MA, February 1985.

14. Brobst, S.A. "Performance Evaluation of the Tagged-Token Dataflow Architecture," Workshop on Performance Evaluation of High-speed Computers, Institute for Computer Sciences and Technology, National Bureau of Standards, Gaithersburg, MD, June 1985.

15. Culler, D. E. "Overcoming Latency in Parallel Computer Systems," Lawrence-Livermore Laboratory, Livermore, CA, July 1984.

16. Culler, D.E. Experience with Tagged-Token Dataflow Architecture Simulator," Essex, CT, MIT /IBM Workshop on Multiprocessors, May 1985.

17. Iannucci, R. A. "The MIT Multiprocessor Emulation Facility," MIT Summer Dataflow Course (6.83s), Cambridge, MA, August 1984.

18. Iannucci, R.A. "The MIT Multiprocessor Emulation Facility," IBM Glendale, Endicott, NY, December 1984.

19. Iannucci, R.A. "High Speed Packet Communication," IBM Burlington, Burlington, VT, January 1985.

20. Iannucci, R. A. "High Speed Point-to-Point Serial Data Communication," IBM Raleigh, Raleigh, NC, February 1985.

21. Iannucci, R.A. "Dr. Strangehost (or *How I stopped worrying and learned to love CMS*)," MIT Laboratory for Computer Science, Cambridge, MA, May 1985.

22. Heller, S. "Automatic Storage Reclamation," Boston University, Boston, MA, August 1984.

23. Nikhil, R.S. "Functional Databases, Functional Languages," Microelectronics and Computer Technology Center, Austin, TX, June 1985.

24. Nikhil, R.S. "Functional Databases, Functional Languages ," University of Texas, Austin, TX, June 1985.

25. Papadopoulos, G. "The MEF: A Multiprocessor Sandbox," with R.M. Soley, MIT Laboratory for Computer Science, Cambridge, MA, February 1985.

26. Papadopoulos, G. The Multiprocessor Emulation Facility," with R.A. Iannucci, MIT Summer Dataflow Course (6.83s), Cambridge, MA, August 1984.

27. Pingali, K. "Sharing of Computations in Functional Language Implementations," Workshop on Functional Language Implementations, Goteborg, Sweden, February 1985.

28. Pingali, K. "Sequential Implementations of Functional Languages," Royal Institute of Technology, Stockholm, Sweden, February 1985.

29. Soley, R. M. "The MEF: A Multiprocessor Sandbox," with Gregory Papadopoulos, MIT Laboratory for Computer Science, August 1984.

30. Traub, K. "An Abstract Parallel Graph Reduction Machine," International Symposium on Computer Architectures, Boston, MA, June 1985.

IMAGINATIVE SYSTEMS

Academic Staff

D. Gifford, Group Leader

Research Staff

S. Berlin

Graduate Students

R. Baldwin	D. Rosenblitt
D. Carnese	R. Schooler
C. Chiang	J. Stamos
J. Lucassen	

Undergraduate Students

H. P. Brondmo	R. Hyre
R. Dreyer	J. LaRocca
B. Gunther	J. Yoon

Support Staff

R. Bisbee

1. INTRODUCTION

The Imaginative Systems Group has expanded its work on the Boston Community Information System over the past year. The Boston Community Information System is a prototype of a large scale information system, and has been used by our research group to explore research issues in databases, packet radio communication, user interfaces, and operating systems. Highlights of our accomplishments in the past year include (1) the enhancement of our information sources, including the addition of the Associated Press and Internet bulletin boards, (2) work on fundamental database theory that will allow us to automatically route queries to appropriate databases, (3) the design of a new inexpensive receiver that can receive our digital transmissions, (4) the enhancement of our personal database system that allows it to access our database servers via two way communication in addition to its basic function of gathering broadcast data, and (5) publication of our research results. The first section of this chapter expands on these accomplishments and outlines our plans for the coming year.

Another activity that our research group has in progress is in the area of programming languages. This effort has examined the thesis that it is possible to simplify existing programming languages by allowing features normally implemented by built-in types to be supplied by library packages. The second section of this chapter discusses this activity in detail.

2. THE BOSTON COMMUNITY INFORMATION SYSTEM

The goal of the research reported here is to use computers to improve communication between people. Our view is that the computer is an excellent communication medium because of its ability to process, index, edit, and display information, and that this capability can be well applied on a large scale to communication within a community. However, building a computer system large enough to serve a community is a difficult problem. Our major design goals were the following: we wanted the system to

- economically serve a major metropolitan user community,
- provi le a high-quality user interface,
- give its users access to a wide variety and a large volume of information,
- allow its users to add value to the information provided by the system by specifying filtering and further processing,
- safeguard the privacy of users,
- be easily extensible to new services.

Our design seeks to address these goals by combining personal computation and communication. A personal computer is located at every user site. Information is transmitted to these personal computers via broadcast communication. The personal computers retain information of interest to their owners and provide a personalized information service. Because each user station has local processing and storage capability, the user can gain effective access to much more data than in a Teletex system without resort to the central, per-user processing required for Viewdata.

The approach of sending information to the user's location and processing it there has a number of advantages. First, the central site can support any number of broadcast service users. Second, locating processing power with the user allows for a high-quality user interface. Third, local processing and storage can be used to assist the user in managing a larger volume of available information. Fourth, the user can choose how to add value to information, integrating received information with local computational tools and data bases. Fifth, the local processing of information keeps private information confined to the user's site. Finally, because the personal computers are fully programmable, the system is easily extensible to new services.

2.1. Broadcast Packet Radio System

In the past year we continued operating our digital broadcast system on WMBR-FM in Boston, and added a second transmitting site on WERS-FM. The second transmitting site provides us with a larger coverage area, and thus future installations of our system will use receivers tuned to this transmitter. As part of our effort to bring up a second transmitting site we worked with an outside contractor to develop a low cost receiver. The receiver, including all of its components, is available retail for under $100.

Both of our transmitting sites use the three-layer protocol we have developed for unidirectional byte channels characterized by burst errors. The three layers of the protocol and their respective functions are shown in Table 7-1. The protocol has a low implementation complexity, and is efficient enough to permit continuous error detection and correction at 4.8 KBits/sec on a personal computer without any special-purpose hardware, using only a fraction of the available CPU power.

The first and lowest layer of our protocol is the byte string layer. A byte string is defined as a finite sequence of arbitrary bytes. There is no guarantee that a byte string is delivered to the receiving sites or that it is delivered without errors, but all byte strings that are delivered are guaranteed to arrive in the order in which they were transmitted.

The byte string layer is implemented directly on top of the byte channel, and the end of each string and the beginning of the next is indicated by one or more bytes serving as separator tokens. Since the contents of the byte string can be arbitrary, any instances of the separator token in the byte string itself are mapped into other values by means of byte-stuffing.

The packet layer of the protocol is implemented on top of the byte string layer. It provides for transmission of packets of arbitrary contents, up to a certain length. (In our implementation, the length in bytes must be a multiple of four not exceeding 4*255). The packet layer serves as an error detection layer: there is no guarantee that individual packets are delivered, but all packets that are delivered are guaranteed to be complete, free of errors, and in order. To accomplish this, the packet layer transmits each packet as a byte string, prefixing it with a length field and a checksum.

If the length field of a received packet does not match the actual length of the received byte string, the packet is rejected. Otherwise, the checksum is computed. If the checksum does not match, the packet is also rejected. Otherwise, the packet is accepted.

Note that the guarantee of error-free transmission is merely probabilistic: if a packet has been corrupted, it will nevertheless be accepted if the length field and the checksum both match. With a 32-bit checksum, an error can go undetected with a likelihood of no less than 2^{-32}.

The data layer of the protocol is implemented on top of the packet layer. It provides for the transmission of data blocks of arbitrary contents, up to an implementation dependent length. The data layer serves as an error correction layer: although it provides the same guarantees as the packet layer (delivered data blocks are complete, free of errors, and in order), it has the potential for greater useful throughput when channel errors are likely to occur.

For transmission, each data block is divided into a number of fixed-size packets. (The last packet may be shorter than the others if the block length is not an even multiple of the packet length). The fragmentation is performed subject to the length constraint on packets and the implementation-defined limit (currently 100) on the maximum number of packets.

Recall that the underlying packet layer does not guarantee that individual packets are delivered. To increase the likelihood that a packet is delivered intact, it may be transmitted more than once. Because the physical channel exhibits burst errors (and no apparent periodicity), we create time diversity by interleaving these redundant transmissions: all the packets of the data block are transmitted once, and then this sequence is repeated as many times as is necessary to achieve the desired expected level of reliability.

The data layer appends a header containing reassembly information to each packet as shown in Figure 7-1. The header format is as follows:

4 bytes	checksum (part of the packet layer)
1 byte	length of packet in 4-byte words (part of the packet layer)

1 byte	reserved to specify the protocol that is used
1 byte	sequence number of the data block (mod 256)
1 byte	reserved for future use
2 bytes	packet number (within the data block). The high-order bit indicates whether this is the highest-numbered packet of the data block.

Using this information, data blocks can be straightforwardly assembled as follows. Since packets are guaranteed not to be delivered out of order, and cannot contain erroneous data, the arrival of a packet with a data block serial number different from that of the previous packet signals the start of a new data block. The bit-vector that records which packets of the data block have been received is initialized to all zeroes, and the expected number of packets (for the new data block) is initialized to "unknown".

For each packet that arrives, this bit vector is consulted. If the packet contents is already present, the packet is ignored and the system waits for the next packet. Otherwise, the packet contents is copied into the data buffer, starting at a location determined by the packet length and the packet number, and its presence is noted in the bit vector. If the packet is tagged as the highest-numbered packet of the data block, the expected number of packets is filled in. Each time a packet is copied into the data buffer, the bit-vector and the expected number of packets are examined to determine whether a complete data block has been received.

Whenever a complete data block is received it is handed over for further processing. In our system, this processing is performed in place, so control is not returned to the data layer until all processing has been completed. This may include decrypting the data block, scanning it to see if it matches the user's filter, and possibly saving the contents on disk. When the data layer regains control, it directs its attention to the incoming packet stream, and is ready to assemble the next data block.

Note that the correctness guarantee on data blocks is once again a probabilistic one: if any packet has been undetectably corrupted, the data block of which it is part may reflect the error, either directly or indirectly through incorrect assembly. Moreover, the one-byte data block serial number does not allow detection of transmission outages that last 255 data blocks (mod 256). Such outages must be detected by a timeout mechanism.

The digital broadcast system that we operate in Boston uses an FM Subsidiary Communications Authority (SCA) channel. SCA channels are subcarriers that can be used by an FM radio station without interfering with normal broadcasting. Typical uses of SCA channels include Muzak, stock quote services, and reading for the blind.

In our application the SCAs of WMBR-FM and WERS-FM are modulated with a frequency shift keyed signal (FSK) that carries RS-232 compatible asynchronous data at 4.8 KBits per second. An FSK system that uses asynchronous signalling is particularly simple to demodulate and to interface to existing serial ports on personal computers: the cost of the receiving equipment (not including the personal computer) is under $100 per receiver.

The SCA channel transports a byte stream to our users. The time-average byte error rate of the channel varies, depending on the receiver configuration (the type of antenna and receiver circuit employed) and receiving conditions (distance between transmitter and receiver, multipath interference, interference from appliances, weather). The time-average byte error rate is given below for several receiver sites.

Currently, the transmission rate of our broadcast subsystem is limited to 4.8 Kbits/sec. This is not a limiting factor, since a typical personal computer can just accommodate continuous 4.8 Kbits/sec transmissions.

We will now describe the procedure used to determine appropriate values for the packet size and the number of repetitions to be employed by the data layer of the protocol. The parameters of the underlying byte channel, such as the transmission rate and the byte format, are presumed to be fixed and are not considered here.

The first step of the process is to determine what objective function (of the parameters in question) we wish to maximize. We have considered the following functions in particular:

- The packet-level throughput rate: this is the ratio between the number of data bytes in a packet and the total packet size, times the probability that a packet arrives intact.

- The block-level throughput rate: this is the ratio between the number of data bytes in a data block and the total number of bytes it takes to transmit the data block, times the probability that the data block arrives intact. We will refer to this quantity as the channel utilization.

- The block delivery probability: this is the likelihood that a transmitted data block arrives intact.

The packet-level throughput rate would make a poor objective function, because it reflects neither the number of repetitions nor the effects of the block size.

The block-level throughput rate, or channel utilization, makes a much better objective function. In fact, if a fixed set of data blocks is transmitted repeatedly in round-robin fashion, and block delivery errors are statistically independent, we can minimize the mean latency from the time a data block is first transmitted until the time it is first correctly delivered by maximizing the channel utilization. Channel utilization will be our primary objective function.

The block delivery probability always approaches unity as the number of repetitions is increased. Therefore, it is not a useful objective function for choosing parameter values. It is nevertheless a useful quantity, and we monitor it to avoid wasteful use of the channel.

To calculate the channel utilization and the block delivery probability for a particular channel, we begin by estimating the packet error rate of the byte channel for various packet sizes. We define a k-burst (of errors) as a maximal-length sequence of bytes terminated by bytes transmitted incorrectly that does not contain a subsequence of k error-free bytes.

Figure 7-4 shows the distribution of error burst lengths observed on our noisy channel for k=5, along with the distribution that would be expected if byte errors were statistically independent, with the same byte error rate. The plateau in the expected distribution is an artifact corresponding to the definition of burst errors, but is nevertheless expected in the actual data. Instead, the observed distribution has a secondary peak at a burst length of 4 or 5, and has a much higher tail than the expected distribution. Therefore, we conclude that byte errors on the channel are not independently distributed, and that we cannot calculate packet error rates directly from the byte error rate of the channel.

Instead, we estimate packet error rates by means of a data collection program that maps the observed sequence of byte errors into (simulated) continuous packet streams of various packet sizes. We have run this program at several receiver sites, analyzing one megabyte of received data at each site. The number of byte errors varied from 4870 (in a windowless, partially shielded, electrically noisy room near the transmitter) to 55 (in the suburbs 7.4 miles west of the transmitter), to 12 (7.5 miles north). A fourth test site, located 9.8 miles southeast of the transmitter, proved to have such poor reception that we cannot deliver data blocks to that site reliably.

The relationship between packet size and packet error rate is determined by the extent to which byte errors are clustered. This dependence is best illustrated by our noisiest set of observations Table 7-2 shows the observed packet error rate as a function as packet size. For comparison, we have included estimated packet error rates based on the (erroneous) assumption that byte errors are independent with a byte error rate of 4870 errors per 1M bytes. It is clear from the table that byte errors are not independent, and that assuming independence would lead one to consistently overestimate the packet error rate. This confirms our earlier conclusion based on the distribution of error burst lengths.

The observed relationship between packet size and packet error rate indicates that even packet-level errors are not independently distributed over the time span investigated, up to 16K bytes. (This can be illustrated by comparing the delivery probability of two 1K packets, $(1-0.415)^2=0.342$, with the delivery probability of one 2K packet $(1-0.570)=0.430$. Since a 2K packet can be viewed as two adjacent 1K packets, we find that the delivery probability of two adjacent 1K packets is greater than the delivery probability of two independently chosen 1K packets. Thus, packet errors are not independent).

Because packet errors on the channel are not independently distributed, we cannot calculate the block delivery probability directly from the packet error rate of the channel. We could proceed as before, and estimate the block delivery probability by mapping the observed sequence of packet errors into a (simulated) continuous stream of data blocks. However, to obtain accurate estimates in this way, we would need data collection runs lasting several orders of magnitude longer than those used to estimate the packet error rates.

Therefore, we are forced to estimate the block delivery probability directly from the packet error rates anyway, as a function of the block size, the packet size, and the number of repetitions. We have made these calculations using the channel model described below.

The channel model we have developed is used to choose suitable parameter values for the data level protocol. The model is based on the assumption that packet errors are independent. To the extent that this is not true, our reliance on average values at the packet level tends to produce an over-estimate of the error rates at the data block level, especially for small packet sizes and high repetition rates.

Let us call the block size N, the packet size n, the number of packets per data block k, and the number of repetitions r. Assume that the probability of packet error, p(n), is given for all packet sizes of interest. Assume further that each packet contains a header of H bytes, leaving room for (n-H) bytes of data. We will use H=10 throughout.

The number of packets per data block (not counting repetitions) is:

$$k = \lceil N/(n-H) \rceil$$

When the block size is not an exact multiple of the packet size, the last packet may be shorter than the rest. Because the packet error rate as a function of packet size is well-behaved, we can account for this by using a non-integer approximation of the number of packets per data block, namely the total number of bytes transmitted per repetition of the data block divided by the packet size:

$$k' = \frac{N + H * \lceil N/(n-H) \rceil}{n}$$

The potential channel utilization (which is achieved when there are no errors) is the ratio between the block size and the total number of bytes transmitted to get each data block across:

$$\frac{N}{r * (N + H * \lceil N/(n-H) \rceil)}$$

If each packet of a block is transmitted r times, the probability that all r copies of a given packet are damaged or lost in transmission is $p(n)^r$ (assuming independence of individual packet errors). Thus, the probability that at least one copy of a packet arrives intact (which is all that is needed) is $1-p(n)^r$. If a block is fragmented into k' packets (not counting repetitions), then the block delivery probability is equal to the probability that all k' packets can be reconstructed, or $(1-p(n)^r)^{k'}$.

The channel utilization is the product of the potential channel utilization and the block delivery probability:

$$\text{channel utilization} = \frac{N * (1-p(n)^r)^{k'}}{r * (N + H*\lceil N/(n-H)\rceil)}$$

Note that H is fixed, N is fixed for each block, N and n together determine k', and the function p(n) is fixed given the receiver site. Thus, we can now maximize channel utilization over all possible values of n and r, subject to the restriction that p(n) is known only for selected values of n.

Table 7-3 gives the combinations of packet size and number of repetitions that maximize the channel utilization for selected block sizes, for the high-error channel mentioned before. Packet sizes of successive powers of two were considered, up to 1024 bytes or the block size (whichever was less). Suboptimal parameter choices, included for comparison, are indicated with (<). Note again that this channel does not reflect realistic receiving conditions, which are up to three orders of magnitude better.

In this table, we can distinguish three different operating regions, namely those with optimal repetition rates of 1, 2 and 3.

For very small data blocks, it does not pay to retransmit packets more than once because the error rate on single transmission is so low that a channel utilization greater than 0.5 can be achieved, which is not possible when each packet is transmitted more than once. For such small data blocks, the largest possible packet size is always optimal.

Next, there is a crossover region where r=2 is optimal. Somewhere within this region, it pays to reduce the packet size from 128 bytes to 64 bytes, as indicated. The model does not allow for accurate comparisons between error rates obtained with and without retransmission, so it is hard to determine the precise block size at which retransmission becomes worthwhile.

Next, for block sizes of 4K and up, the model yields a remarkably stable optimum with r=3 throughout our range of measurement. The model indicates that throughout our range of actual block sizes (4K to 10K), a packet size of 128 bytes is optimal. For very large blocks, the packet size should be reduced to 64 bytes.

Table 7-4 shows how the block delivery probability and the channel utilization vary with the packet size and the number of repetitions for a given block size (4K bytes). Note that for r=1, there is no error correction and therefore channel utilization is maximized with the largest possible packet size, because it yields the most compact encoding and thus minimizes the probability of error.

For r=2, the *block delivery probability* is maximized with a packet size of 32. However, the overhead on these packets due to headers is substantial. Indeed, the *channel utilization* is maximized with a packet size of 64, despite a somewhat lower block delivery probability. Note also that channel utilization is bounded above by 0.5.

For r=3, the channel utilization is bounded above by 1/3. This value is most nearly reached with a packet size of 128. This combination of repetition rate and packet size happens to be the best value in the table. In particular, nothing can be gained by increasing r, since this would limit channel utilization to 0.25 (for r=4), 0.20 (for r=5) and so on.

Finally, compare the data for r=2 at r=3 for a packet size of 64 bytes. The channel utilization rates differ only minimally, but the block delivery probability is much greater for r=3. In fact, it is about 50% greater, which nearly compensates for the fact that each packet is transmitted three times instead of twice.

If we had complete and accurate information regarding packet error rates for all receiver sites, we might be able to choose the packet size and retransmission rate for each data block so as to maximize some function of the channel utilization observed at all receivers. A policy decision would have to be made regarding the relative level of service to be delivered to nearby and distant users.

Resource limitations have prevented us from collecting accurate packet error rate estimates for all receiver sites. Moreover, the channel utilization figures computed with the aid of the model are only approximate. We presently set our operating parameters on the basis of available packet error rate estimates for several receiver sites we believe to be representative.

Note that for large block sizes, the channel utilization and the delivery probability decline exponentially with increasing block size, regardless of the packet size and retransmission rate. Thus, the broadcast protocol without acknowledgments is not effective for very large block sizes.

2.2. Information Protection

Since our broadcast system uses a public medium, we cannot prevent unauthorized users from listening to the broadcasts. Yet, the dissemination of information must often be limited. For example, there may be copyrights and other restrictions attached to the information to be broadcast. To enable such control over the dissemination of information, we encrypt all data blocks that are intended for a restricted audience.

Each block is encrypted using a combination of a master key and a randomly generated data block key. The data block key is different for each data block, and is transmitted along with it. The master key is secret: it is made available only to the legitimate users of the service. In practice, we employ a table of master keys, identified by number. Each encrypted data block carries a number identifying the master key that was used to encrypt it. Unencrypted data blocks are identified by a key number of zero. This scheme has the following properties:

- The information that is broadcast can be thought of as being separated into distinct logical streams, each with its own master key. Consequently, users can subscribe to certain services without having access to all services.

- The master key used for the encryption of a certain information stream may be changed periodically, for example once a month. Paying subscribers can be provided with the keys for the duration of their subscription. Since the key number changes along with the key, the receiver will automatically switch to the new key whenever a switch takes place. The key numbers in the table may be reused.

Because of the hardware limitations of our receiver stations, we are unable to utilize better-known encryption techniques such as DES or RSA. Instead, we have implemented an algorithm which is very efficient, and appears to afford a level of security commensurate with the value of the information we seek to protect. Our cryptographic algorithm uses a linear feedback shift register with non-linear output to generate a pseudo-random stream that is combined with the transmitted data. The shift register component of the algorithm is shown in Figure 7-5. At an effective data rate of 2.2 KBits per second, the decryption utilizes about 6.8% of the available CPU time on an IBM PC. (All performance figures pertain to implementations written entirely in C)

2.3. The Predicate Data Model

When considering how to provide access to community information one soon realizes that it is a problem that cannot be solved by the application of standard commercial database techniques. Relational and hierarchical data models are far too restrictive to allow users to locate information of interest, because of the relatively unstructured information (such as newspaper articles) that the system must handle.

We have developed a new data model, called the Predicate Data Model, for our application. The Predicate Data Model builds on ideas from full-text retrieval systems to fill our needs. The Predicate Data Model is capable of handling a wide variety of information, including text-oriented data (news stories and electronic mail), information that has somewhat more structure (community event descriptions and city guides), and other kinds of data, including computer programs.

Every entry in a predicate data base, whether it be a *New York Times* article or a restaurant review, consists of a number of fields. The specific fields found in an entry depends on its type. For example, a newspaper article has a source identifier (e.g., "New York Times"), along with *category, subject, priority, section, title, author, date,* and *text* fields. The *text* field contains the body of the article. Likewise, an event listing includes *location, time, title,* and *abstract* fields. Some fields can only contain certain words: for example, the *priority* field of a news article is chosen from the words *flash, bulletin, urgent, regular,* and *deferred.* Other fields, such as the *author* or *text* field, contain arbitrary text, and a user query can include arbitrary words and phrases.

A user phrases queries as boolean combinations of predicates on fields. The predicates restrict fields to hold certain desired values: for example, predicates can limit attention to data base records that contain specified date ranges, words, or phrases. This approach is in contrast to controlled vocabulary systems where information is only indexed on a predefined set of index terms and the user is limited to these terms when formulating a query.

When the user submits a query to the system, a list of matching data base entries is displayed along with a summary of each entry. Figure 7-1 is a picture of the user interface that shows the result of entering the query "technology & (category financial)". Once the menu of available entries is displayed, the user may enter the number of the desired entry on the end of the query and press the return key. Figure 7-2 shows the display of the second entry from the menu of Figure 7-1.

We decided against basing our system solely on menus because we felt that free text searching provides more expressive power and is easy to understand and use. Menu based systems do have the advantage that the user can easily browse the database to see what is available without having something specific in mind. We have tried to retain this advantage of menu systems.

Our personal database system uses both free text and menu-based retrieval in an effective way. A user specifies what information should be kept in the local database by composing a set of free text queries. Because the system knows nothing *a priori* about the user's interests (and the user knows little about the system's capabilities and the scope of available information) unrestricted text is by far the most efficient medium for expressing such information filters.

However, when it comes to examining the local database that was compiled with the aid of these filters, the system "knows" what the user's interest profile is. This makes menu-based retrieval the most efficient medium. The information filters defined by the user serve as a menu of what is available in the local database; furthermore, the user can use any of the filters, or any query, for example "(category news)", to obtain a list of matching articles, and then browse through that list. We find that most of our users use their information filters to browse the database.

The greatest asset of our data model is that it is simple to understand. Thus our users have a good conceptual model of precisely what the system can and will do when they compose a query. This in turn allows them to use the system effectively. However, our users still need to know certain things about the data base in addition to the access mechanisms. Consider the problem of locating movie reviews. Is the query "movie & review" appropriate, or might "(subject review) & movie" be better? To help users learn about the data base and about composing queries, we provide printed and on-line documentation, and we also provide a library of pre-planned queries that users can incorporate into their personal data base system.

2.4. Reasoning About Predicate Databases

We have successfully demonstrated a prototype reasoning system that can determine when a database contains information needed to process a given query. This reasoning system uses a set of axioms for the predicates that are defined as part of our data model to determine when a statement of the form "Q(x) -> DB(x)" is true. When Q(x) implies DB(x), then the result set of Q will be a subset of DB. Thus, Q can be processed using data found at DB.

The application of our content reasoning work is to enable us to use a collection of independent databases to form an integrated system that we call a *multi-database.* For example, our personal database system maintains a private database of information that is of interest to its owner. However, when it accepts a query that it can not process, we would like the personal database system to forward the query to another system. Work on a full implementation of query routing is underway.

2.5. Plans for the Next Year

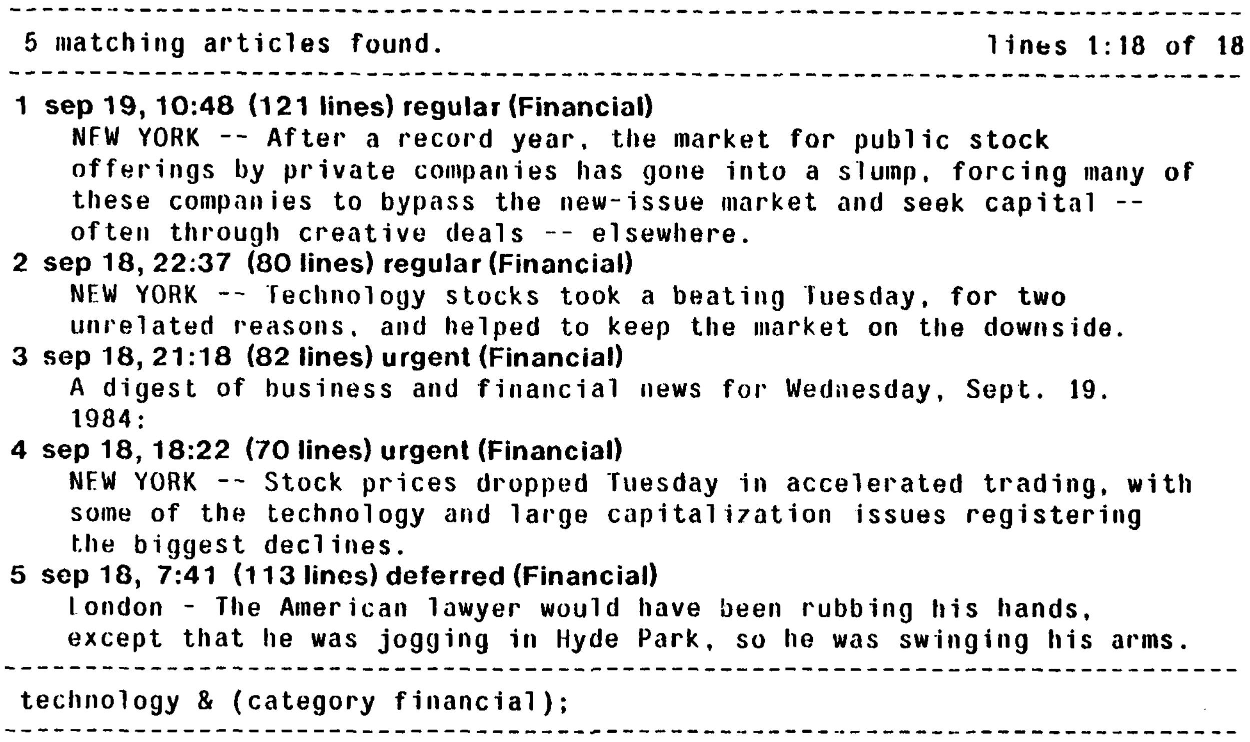
Our plans for the coming year for our Community Information System project are to gain more practical experience with the system that we have built, and to begin to exploit our fundamental work on database theory through implementation. Our plans for gaining experience with user communities include (1) a two hundred site test of our technology this fall at homes in the Boston area (this test is being underwritten by another sponsor), (2) export of software that will allow other DARPA sites to access our databases, and (3) the integration of our database system into the LCS Common System. The fundamental work that we plan to integrate into our system is the addition of the database content reasoning. Content reasoning will allow us to direct user queries to appropriate databases, and thus produce an integrated system out of a collection of independent databases.

3. IMAGINE PROGRAMMING LANGUAGE

Work continues by D. Gifford and J. Lucassen on an investigation of how to simplify programming languages by eliminating many of the distinctions between built-in and user-defined types. The intent is to allow many facilities that are normally provided as part of a language environment to be supplied as library packages. In order to accomplish this goal we have focused on how to make user defined types "first class citizens" in a programming languages, with the same rights and privileges as built-in types. The two specific areas that we have examined include (1) new ways of type checking programs and (2) optimization of programs.

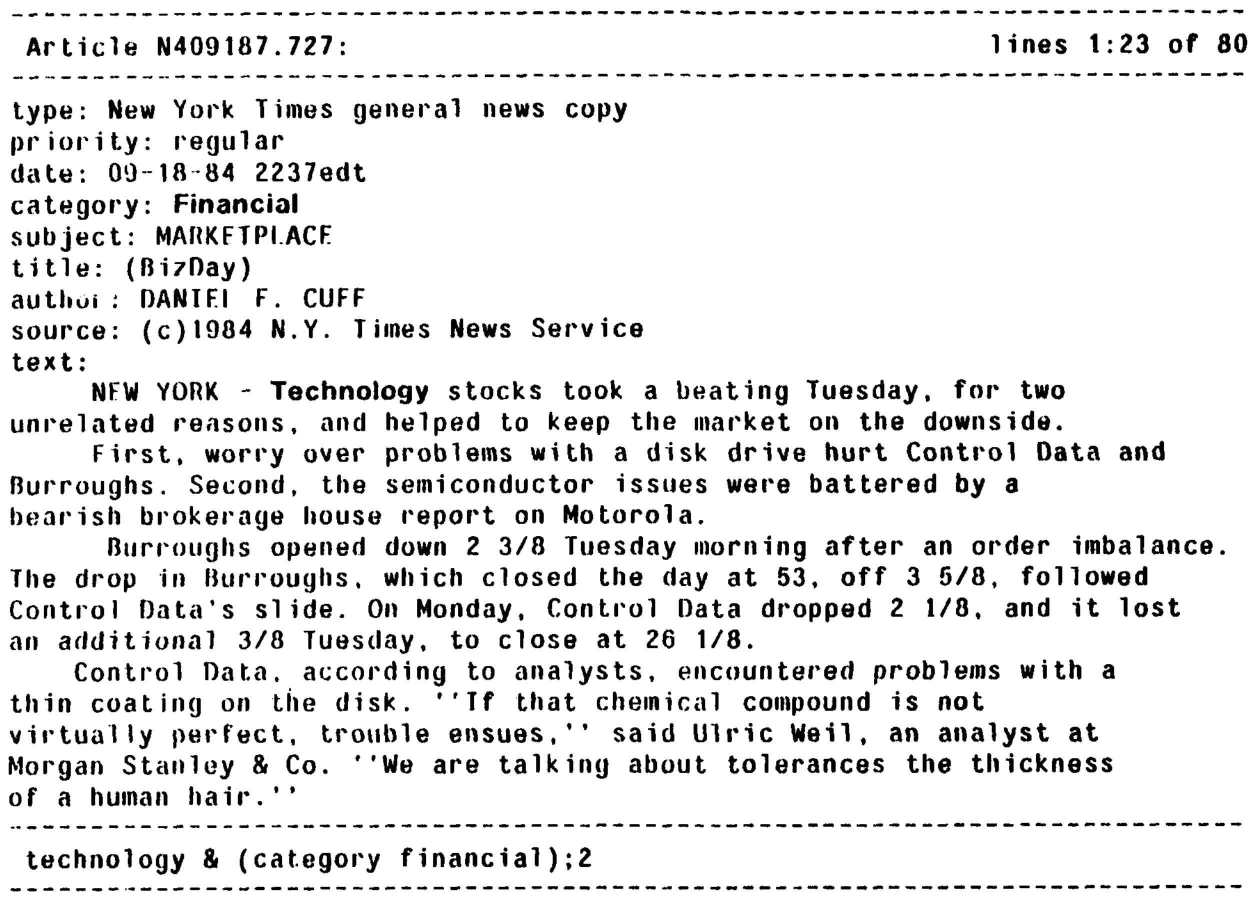
The type checking work that we have completed has examined how a type checker for the second-order lambda calculus could be extended by users. The idea is to allow the type of certain expressions to be computed by user defined functions. This simple addition would allow a wide range of flexibility in the type of programs that could be accepted by the type checker.

The optimization work that we have completed has examined focused on the partial evaluation of programs that have functional subcomponents. The programming language that we have developed as part of this work, Imagine, has a functional sublanguage. Thus the portions of a program that are functional can be easily statically determined. Partial evaluation is a powerful technique that encompasses optimizations such as constant folding and in-line procedure substitution.



```
------------------------------------------------------------------------------
 5 matching articles found.                                  lines 1:18 of 18
------------------------------------------------------------------------------
1 sep 19, 10:48 (121 lines) regular (Financial)
   NEW YORK -- After a record year, the market for public stock
   offerings by private companies has gone into a slump, forcing many of
   these companies to bypass the new-issue market and seek capital --
   often through creative deals -- elsewhere.
2 sep 18, 22:37 (80 lines) regular (Financial)
   NEW YORK -- Technology stocks took a beating Tuesday, for two
   unrelated reasons, and helped to keep the market on the downside.
3 sep 18, 21:18 (82 lines) urgent (Financial)
   A digest of business and financial news for Wednesday, Sept. 19.
   1984:
4 sep 18, 18:22 (70 lines) urgent (Financial)
   NEW YORK -- Stock prices dropped Tuesday in accelerated trading, with
   some of the technology and large capitalization issues registering
   the biggest declines.
5 sep 18, 7:41 (113 lines) deferred (Financial)
   London - The American lawyer would have been rubbing his hands,
   except that he was jogging in Hyde Park, so he was swinging his arms.
------------------------------------------------------------------------------
 technology & (category financial);
------------------------------------------------------------------------------
```

Figure 7-1: User Interface: Menu Screen



```
--------------------------------------------------------------------------------
 Article N409187.727:                                         lines 1:23 of 80
--------------------------------------------------------------------------------
type: New York Times general news copy
priority: regular
date: 09-18-84 2237edt
category: Financial
subject: MARKETPLACE
title: (BizDay)
author: DANIEL F. CUFF
source: (c)1984 N.Y. Times News Service
text:
     NEW YORK - Technology stocks took a beating Tuesday, for two
unrelated reasons, and helped to keep the market on the downside.
     First, worry over problems with a disk drive hurt Control Data and
Burroughs. Second, the semiconductor issues were battered by a
bearish brokerage house report on Motorola.
      Burroughs opened down 2 3/8 Tuesday morning after an order imbalance.
The drop in Burroughs, which closed the day at 53, off 3 5/8, followed
Control Data's slide. On Monday, Control Data dropped 2 1/8, and it lost
an additional 3/8 Tuesday, to close at 26 1/8.
    Control Data, according to analysts, encountered problems with a
thin coating on the disk. ''If that chemical compound is not
virtually perfect, trouble ensues,'' said Ulric Weil, an analyst at
Morgan Stanley & Co. ''We are talking about tolerances the thickness
of a human hair.''
--------------------------------------------------------------------------------
 technology & (category financial);2
--------------------------------------------------------------------------------
```

Figure 7-2: User Interface: Article Display

```
Bytes                Contents

           ---------------------------------
0:1        |            checksum           |
           ---------------------------------
2:3        |            checksum           |
           ---------------------------------
4          | Packet length |
           -----------------

                           -----------------
5                          |   Protocol    |
           ---------------------------------
6:7        | Block seq no. |   Reserved    |
           ---------------------------------
8:9        | F |        Packet no.         |
           ---------------------------------
```

The format of packet headers, including header information attached by both the packet layer (bytes 0 through 4) and the data layer (bytes 5 through 9.)

Figure 7-3: Packet Headers

```
Burst        No. of
Length:      Bursts:               Observed bursts, * = 10

        Expected  Observed
 1        4400       736           *****************************************//*
 2          20        54           *****
 3          20       123           ************
 4          20       138           **************
 5          20       137           **************
 6          20       110           ***********
 7                    90           *********
 8                    41           ****
 9                    21           **

10-14                 29           ***
15-19                  7           *
20-26                  4
65                     1
```

Histogram of error burst lengths expected and observed on one megabyte of data received over a noisy channel.

Figure 7-4: Error Burst Lengths

```
.-------.-------.-------.-------.-------.-------.-------.-------.
        |   0       1       2       3       4       5       6       7   |
        ----------------------------------------------------------------
            |       |                                               |
            |   -----------                                     ----------
            |  |Shift Right|                                   |Shift Left|
            |   -----------                                     ----------
            |       |                                               |
            |       |       ---------------------------------------
            |       |       |
         -----------------------
        | Bit-wise Exclusive-Or |
         -----------------------
                    |
                 New Byte
```

Figure 7-5

NAME OF LAYER:	UNIT OF TRANSMISSION:	PURPOSE OF LAYER:
Data Layer	Data Blocks	Error Correction
Packet Layer	Packets	Error Detection
Byte String Layer	Byte Strings	Framing
(Byte Channel)	(Bytes)	(Transport)

The three layers of the broadcast protocol built on top of the byte channel.

Table 7-1: Protocol Layers

Packet Size (in bytes)	Estimated Packet Error Rate	Observed Packet Error Rate
4	0.018	0.009
8	0.037	0.014
16	0.072	0.025
32	0.138	0.044
64	0.258	0.075
128	0.449	0.125
256	0.696	0.193
512	0.908	0.286
1024	0.991	0.415
2048	1.000	0.570
4096	1.000	0.711
8192	1.000	0.835
16384	1.000	0.953

Estimated packet error rates based on 4870 independent byte errors per megabyte of data, and packet error rates actually observed on one megabyte of data with 4870 byte errors.

Table 7-2: Packet Error Rates

Optimal Block size	Optimal Packet size	Block number of repetitions	Corresponding delivery probability	channel utilization
256	256	1	.7940	.7365
512	512	1	.7045	.6780
1K	1K	1	.5789	.5678
2K	128	2	.7615	.3500
3K	128	2	.6645	.3054 (<)
3K	64	2	.7246	.3056
4K	64	2	.6508	.2745 (<)
4K	128	3	.9349	.2871
6K	128	3	.9039	.2774
8K	128	3	.8740	.2684
10K	128	3	.8452	.2597
10K	64	3	.9227	.2594 (<)
12K	64	3	.9079	.2553
14K	64	3	.8935	.2512
16K	64	3	.8792	.2472

Optimal packet size and number of retransmissions, and resulting block delivery probability and channel utilization, as a function of block size for a particular noisy channel.

Table 7-3

Number of repetitions (r):

	r=1		r=2		r=3		r=4	
Pkt Size	Deliv. Prob.	Chann. Utiliz	Deliv. Prob.	Chann. Utiliz	Deliv. Prob.	Chann. Utiliz	Deliv. Prob.	Chann. Utiliz
16	0.0000	0.0000	0.6441	0.1207	0.9889	0.1236	0.9997	0.0937
32	0.0002	0.0001	0.6936	0.2381	0.9839	0.2252	0.9993	0.1715
64	0.0027	0.0023	0.6508	0.2745	0.9683	0.2723	0.9976	0.2104
128	0.0098	0.0090	0.5805	0.2674	0.9349	0.2871	0.9917	0.2284
256	0.0283	0.0272	0.5325	0.2557	0.8873	0.2840	0.9773	0.2346
512	0.0636	0.0622	0.4974	0.2434	0.8238	0.2687	0.9465	0.2315
1024	0.1141	0.1127	0.4651	0.2298	0.7406	0.2439	0.8852	0.2186

Block delivery probability and channel utilization as a function of packet size and number of repetitions, for blocks of 4K bytes transmitted over a particular noisy channel.

Table 7-4

Publications

1. Gifford, D. and Donahue, J., "Coordinating Independent Atomic Actions,", Proceedings of IEEE Spring COMPCON 85, February 27, 1985, San Francisco, CA.

2. Gifford, D., Lucassen, J., and Berlin, S., "Application of Digital Broadcast Communication to Large Scale Information Systems", *IEEE Transactions on Selected Areas in Communications*, (May 1985).

3. Gifford, D. and Spector, A. (ed.) "A Case Study of The Space Shuttle Primary Computer System," *Communications of the ACM*, (September 1984).

Thesis Completed

1. Carnese, D. "Multiple Inheritance in Contemporary Programming Languages," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1984.

2. Gunther, B. "Extended User Interface Support for a Personal Database System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

3. Schooler, R. "Partial Evaluation as a Means of Language Extensibility," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1984.

4. Yoon, J., "Broadcast Based Electronic Mail Services", S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

Theses in Progress

1. Chiang, C. "Primitives for 3D Graphics," S.M thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1985.

2. Lucassen, J. Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1986.

3. Slivan, S., "An Interactive Graphics Interface for Clinical Applications Software," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1985.

4. Stamos, J. "Remote Evaluation," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1985.

Talks

1. Gifford, D. "Research in Operating Systems and Computer Networks," ACM SIGOPS and IBM Zurich, January 1985.

2. Gifford, D. "The Boston Community Information System," Cambridge University, Cambridge, England, September 1984.

3. Gifford, D. "The Boston Community Information System," Digital Equipment Corporation, Hudson, MA, August 1984.

INFORMATION MECHANICS

Academic Staff

E. Fredkin, Group Leader

Research Staff

T. Toffoli

G. Vichniac

Visiting Scientist

V. Sewelson

Graduate Student

N. Margolis

Undergraduate Students

P. Tamayo

C. Ferreira

Support Staff

T. Cloney

D. Zaig

1. OVERVIEW

Our field of research: Information Mechanics is concerned with dynamical systems that can be given two distinct interpretations: models of idealized physical processes, and physical models of computational processes. On the one hand, the study of these system leads to a better understanding of the physical basis of computation, both in practical terms (e.g., efficient use of energy, virtually dissipationless computing) and in theoretical terms (borrowing of physical concepts and mathematical methods). On the other hand, this study provides new insights into computational models of natural sciences. Whereas such insights have shown to be relevant at quite a conceptual level, this second approach finds a practical motivation in the need for efficient computation. This motivation suggests a very original use of computers in the sciences and stems from the following tension. While digital computers are discrete objects, much of science is expressed with continuum mathematics (e.g., the differential equations of physics). In order to overcome this mismatch and let computers do what they do best; i.e., logical manipulation of bits as opposed to necessarily imprecise floating-point arithmetic, one can restrict oneself to models that are already fully discrete. Such models lend themselves to exact simulation by finitary means. Numerical analysis, an inevitable and often opaque screen between standard (continuum) models and their discrete simulations is eliminated altogether.

We concentrated this year again on these important systems of information-mechanics: cellular automata (see below, Sections 2 and 3), reversible computation (Section 4), and reversible finite-difference equations (Section 5). The theoretical aspect of or work was complemented and stimulated by the construction of hardware dedicated to the efficient simulation of these dynamical systems.

2. CELLULAR AUTOMATA

Description: Cellular automata are distributed dynamical systems governed by laws that are local and uniform, in analogy with the laws of physics. However, unlike the traditional models of physics, viz., differential equations, cellular automata operate on discrete space and time, and the dynamical variables take on values from a finite set. As a result, one obtains **exactly computable** models of dynamical processes.

The cellular-automaton machine (CAM:) The exact computability of cellular automata is exploited by our cellular-automaton machine -- a high-performance simulator of discrete, distributed dynamical systems. CAM has now reached a mature stage of development; it has been used extensively by us and by a number of visiting scientists, and has been demonstrated on numerous occasions in the U.S. and abroad.

The success of CAM has raised an issue of scientific ethics. In the natural sciences, an experiment is considered *scientific* only if it is repeatable by independent investigators. But while the natural world is freely accessible to anyone, experiments that are possible only with a high-performance machine can be repeated and extended only by investigators who have access to comparable computing resources. And in fact, we have received a great number of requests from individuals and institutions who want a copy of CAM for their own laboratories.

To satisfy these requests, we have commissioned Systems Concepts, San Francisco, CA, to develop and produce a version of CAM that fits in a single slot of the IBM-PC, and we are providing them with technical assistance and with the entire software (programs and applications data). The design of this software and the reaction of a book to accompany the commercial version of CAM has been one of our main efforts this year. We have striven to make the software extensible by the end-user, and to this purpose we have adopted and upgraded a public-domain version of the Forth programming language. Both hardware and software should be available to the public in the third quarter of 1985.

Inhomogeneous cellular automata: Standard cellular-automaton rules are defined to be **uniform**, i.e., the same transition function holds at every site of the lattice. We have recently investigated a generalized class of networks, the *Inhomogeneous Cellular Automata*, where **several** transition rules are distributed on the lattice. We found that the interplay of two simple Boolean functions reveals ordered and adaptive behavior of importance in the study of self-organizing systems.

Parallel computation: In cellular automata, all the sites compute their next state in a synchronous way. Cellular automata thus offer a paradigm for parallel architecture. They provide a stimulus to designing algorithms which effectively exploit the potential of massively parallel hardware. We have studied the effects of synchronous updating in a regular network of locally interconnected Boolean variables (i.e., Ising spins). We recently found that even in this very simple instance of distributed system, one must strictly follow somewhat counter-intuitive rules to make a full use of the parallel computational resources. We are now investigating how this parallelism can split a simple network into two separated complex interweaved ones.

The CAM-7 project: We have investigated in considerable detail the feasibility of constructing a special-purpose machine for the efficient simulation of three-dimensional cellular automata of size 512 x 512 x 512, with 2 bits per site and an update rate of 60Hz for the whole array. We are now completing a proposal for a major grant to built such a machine.

3. EXTENSION OF COMPUTATIONAL CONCEPTS TO NATURAL SCIENCES

Conservation laws: W have continued this year our program of reducing concepts from physics to informational primitives such as counting, labeling, and comparing. Very important examples of such concepts are quantities conserved under information-preserving dynamics. In *continuous* variational systems, these conserved quantities are intimately related to symmetries (Noether's theorem). *Discrete* systems, on the other hand, are not known to obey a similar relation. The derivation of a systematic way to construct conserved quantities is a central and difficult problem in the theory of discrete structures; it has resisted our efforts so far. But recently, concentrating on a cellular-automaton rule singled out by us as "most remarkable," the hydrodynamicist Y. Pomeau succeeded in deriving for that rule a conserved quantity analogous to energy. We are now striving to extend this result, originally derived by methods borrowed from fluid dynamics, by using techniques more suited to discrete structures, such as those devised recently by E. Goles Chacc (who will be a Visiting Scientist in our Group next spring).

Information treatment of an irreversible process: We found recently that any iterated deterministic manipulation on a random configuration of symbols is analogous to the *quenching* of a disordered system to low temperatures. This permits an analysis of the processes of ordering via domain growth and interface motion in terms of propagation of information.

Applications to efficient computation: Motivated by the importance of exactly computable models (see above), we have investigated finitary models based on the combinatorics of a large number of discrete variables. We have found, using Partitioning Cellular Automata, that well-known parabolic ("heat") and hyperbolic ("wave") partial differential equations are the limits, over distances much greater than the mean free path, of suitable *lattice gases* of binary variables.

4. REVERSIBLE INFORMATION PROCESSING: FOUNDATIONS AND QUANTUM THEORY

Foundations: A major part of our current research builds upon the discovery by Bennett, Fredkin, and Toffoli that reversible computation is compatible with the laws of physics. We developed the concept of "virtually dissipationless computation," and precisely characterized it by a quantitative bound that takes into account noisy environment.

Quantum Computation: We have completed a unified treatment of Quantum Computation, based on the Billiard-Ball-Model cellular automaton. This model yields a description of deterministic computations which could, in principle, operate within the constraints of a local quantum-mechanical Hamiltonian. An open problem which we are studying is the question of whether a truly parallel quantum implementation of such a cellular automaton can operate at a uniform computational rate. If this is the case, an important barrier to computation using elements of atomic dimensions will have been passed.

5. REVERSIBLE FINITE-DIFFERENCE EQUATIONS

Oscillations resulting from information conservation: We found that the conservation of information entail oscillations in reversible finite-difference equations. We completed a treatment of these oscillations in terms borrowed from numerical analysis. Taking now the point of view of dynamical system theory, we are starting a more quantitative study that involves the measurement of Liapunov exponents for the oscillations.

6. SCIENTIFIC EXCHANGES

Various kinds of exchanges: The novelty of our field of research, the absence of a journal fully dedicated to it, and the interest from the scientific community make it necessary to devote much effort in scientific exchanges. In innumerable occasions, we have demonstrated CAM to scientists from various disciplines. We have accepted many invitations to give seminars, conferences talks, and courses. We have also built and installed a copy of CAM for permanent exhibit at the new Computer Museum in Boston.

Benefits of exchanges: These scientific exchanges have proven to be very beneficial to our research, although they demand much time and effort. They allow us to expose (and subcontract) problems related to our field; and conversely to receive suggestions how best to use our methods and hardware facilities.

Publications

1. Vichniac, G.Y. "Instability in Discrete Algorithms and Exact Reversibility," *SIAM Journal Alg. Disc. Meth.* 5 (1984), 596.

2. Toffoli, T. "A Comment on 'Dissipation and Computation,'" *Phys. Review Letter* 53 (1984), 1204.

3. Giraud, B.G. and Vichniac, G.Y. "Effective Forces and Rigorous Variational Principles," to appear in *Phys. Rev. A.*

4. Vichniac, G.Y. "Cellular Automata and their Applications," to appear in *Disordered Systems and Biological Organization,* F. Fogelman, G. Weisbuch, and C. von der Malsburg (Ed.), Lectures Notes in Computer Science, Springer-Verlag.

5. Hartmann, H. and Vichniac, G.Y. "Inhomogenous Cellular Automata," to appear in *Disordered Systems and Biological Organization,* F. Fogelman, G.~ ~Weisbuch, and C. von der Malsburg (ed.), Lectures Notes in Computer Science, Springer-Verlag.

Talks

1. Fredkin, E. "Physics and Computation,"

   Geological Institute, Moscow, September 1984.

   Space Center, Moscow, September 1984.

   Physics Colloquium, Carnegie-Mellon University, October 1984.

   Physics Colloquium, Harvard University, October 1984.

2. Fredkin, E. "Personal Computers and the Future,"

   Soviet Academy of Science, Address to the Presidium, Moscow, September 1984.

   Academy of Science, Computation Center, Moscow, September 1984.

   Dickson Prize Acceptation Ceremony, Carnegie-Mellon University, October, 1984.

3. Toffoli, T. "Parallel Computation: Connection with Physical Modeling,"

Institute of Mathematics, University of Rome, Italy, October 1984.

Institute Applic. Computers, National Research Council, Rome, Italy, October, 1984.

National Institute for Nuclear Physics, Frascati, Italy, October, 1984.

Centro Studi IBM, Rome, Italy, October 1984.

Electrical, Computer, and System Engineering Research Seminar, Boston University, November 1984.

Computer Science Department, Northeastern University, Boston, December 1984.

Computer Science Colloquium, Rensselaer Polytechnic Institute, Troy, NY, April 1985.

4. Toffoli, T. and Margolus, N. "CAM Demonstration," Workshop on Theories of Complexity: Common Frontiers of Physics, Biology, and Computation; Dedham, MA, August 1984.

5. Vichniac, G. Course at the NATO Advanced Research Workshop on Disordered Systems and Biological Organization, Les Houches, France, February/March 1985.

6. Vichniac, G. Speaker at Meeting on Problems in Optimization and Complexity and the Statistical Mechanics of Disorder, CECAM, Orsay, France, March 1985.

7. Vichniac, G. Speaker at UNESCO international Workshop on the Use of Microcomputers in Science Education for Schools, Balaton, Hungary, May 1985.

Awards

1. Fredkin, E. 1984 recipient of the Dickson Prize of Science.

MESSAGE PASSING SEMANTICS

Academic Staff

C.E. Hewitt, Group Leader

Research Staff

G. Agha
H. Lieberman
T. Reinhardt

Graduate Students

J. Amsterdam
P. deJong

Undergraduate Students

J. Allard
J. Aspnes
O. Etzioni
C. Manning
S. Penberthy

Support Staff

C. Smith

1. OVERVIEW

The goal of the research is to develop a parallel architecture for "Open Systems" using the formal theory of actors.

2. OBJECTIVES

We are addressing the problem of facilitating cooperation between open systems which are separately developed and maintained. For separately built systems to communicate, there must be some minimum of common knowledge and structure. In addition, each system must have some self-referential ability in order to relate its structure to its behavior. Using this common knowledge, open systems must bootstrap further capabilities for communication and cooperation.

In order to provide tools which can be used to build open systems with relative ease, we are developing the Apiary architecture. The Apiary architecture has been implemented using a network of conventional processors; it supports dynamic reconfigurability, extensibility, and resource management using techniques such as load balancing and real-time garbage collection.

3. APPROACH

Several characteristics of open systems must be considered. Among these are:

Asynchrony: One cannot predict when new information may enter an open system and, furthermore, the system may be queried before or after the answers to such queries are available. Changes occur at different points at unpredictable times.

Continuous Availability: It should be possible to modify a system incrementally and dynamically rather than shutting it down.

Arms-Length Relationships: The knowledge bases and internal arrangements of one system (or system component) are not available to others.

Inconsistent Knowledge Bases: Because of the differing concerns of the independent participants, different knowledge bases will contain conflicting beliefs.

Parallelism: Different agents in a system function in parallel. New agents may be created as needed.

Actors are a very general modeling technique that *address* issues of parallelism, resource control, continuous availability and the controlled sharing of information in open systems. The mathematical theory of actor systems addresses issues of abstraction and parallel composition. Abstraction provides us with the ability to view systems at a higher level, ignoring all the details about their internal operations. Larger systems can be built from smaller modules using parallel composition so that the constituent parts are still asynchronous and function independently of each other.

4. CURRENT STATUS

The research so far has led to exciting developments in the understanding of the nature of parallelism and in the semantics of concurrency. Our work has evolved along two lines: the development of the theoretical foundations, and an implementation designed to create an environment in order to test the hypotheses.

Recent work in the area of actor semantics has provided a model of actor systems using finitary elements with concrete intuitive structure. A variant of the reduction calculus has been defined which in fact models the technically difficult semantics of arrival order nondeterminism for the incoming communications. An algebraic model for actor systems has been developed. This model takes into account the open interactive nature of such systems. An observation equivalence relation allows us to encapsulate systems and specify their behavior abstractly without assuming a closed-world hypothesis. The study of actor semantics has led to a better understanding of concurrent systems in their full generality.

The development of interactive editing and debugging tools is one aspect of the implementation. In particular, a technique called the biography mechanism has been implemented. When the system is run in a debugging mode, each actor can record all the events which have occurred in its lifetime. From this record, the path of the computation may be reconstructed, examined, and even reversed for debugging purposes. Since the events provide information that is too detailed, a higher level representation is provided in terms of transactions. Transactions group events according to the abstractions of the source program, so they provide a basis for extending abstraction into the debugging tools.

Resource allocation is a critical problem for concurrency. In the context of open systems there are no global consistency requirements. Instead, different actors may compete for limited resources. Sponsors provide the resources dynamically, continually judging the utility of each computation. Sponsors have been implemented to support resource management.

5. FUTURE PLANS

Our major effort over the next two years will be directed towards the study and implementation of description systems based on the actor model. At a linguistic level, description systems provide the means to use generalized pattern-matching for data extraction and authentication. At a higher level, description systems relativized within viewpoints also provide the means to deal with contradictions in the local databases of a system.

Our long-term challenge is to determine the minimal common knowledge and structure necessary to establish communication and cooperation between independently developed systems and to provide tools for the construction and coordination of self-reflective systems.

6. RESOURCES AND PARTICIPANTS

Implementation of demonstration systems takes place primarily on a network architecture based on Symbolics 3600 LISP Machines. The Laboratory's DEC 20/60 is used for auxiliary support.

PROGRAMMING METHODOLOGY

Academic Staff

B. H. Liskov, Group Leader	W. E. Weihl

Research Staff

P. R. Johnson	R. W. Scheifler
S. Perl	

Graduate Students

S.-Y. Chiu	G. T. Leavens
M. S. Day	B. M. Oki
E. Kolodner	J. P. Restivo
R. Ladin	E. F. Walker

Undergraduate Students

L. W. Allen	C. D. Chambers
E. E. Anderson	M. D. Ober

Support Staff

A. L. Rubin

Visitors

R. Leidhammar	T. Takai

1. OVERVIEW

This year we have continued to work on the Argus programming language and system (see [30], [32]). Argus is being developed to support the programming and execution of distributed programs, which run on nodes (i.e., computers) connected by a network. It provides two major mechanisms. *Guardians* allow a distributed program to be decomposed into components. A guardian resides at a single node and contains within it data objects and processes; it is resilient to crashes of its node. It provides a set of operations called *handlers* that can be called by other guardians to access and modify its objects. Some of its objects are *stable* and are written to stable storage devices periodically. After a crash, the Argus system restarts the guardian with its stable objects as they were as of the last write to stable storage; the guardian then runs a special recovery process to restore the rest of its state.

Atomic actions, or *actions* for short, allow distributed computations to cope with concurrency and failures. Concurrent actions are *serialized*, which means that their effect is the same as if they were run in some serial order. Also, they are *total*: either an action completes entirely, or, if this is impossible, it has no effect. In the former case we say the action *commits*; otherwise, it *aborts*.

Actions in Argus can be nested; an action can have one or more *subactions*. Subactions provide a clean way of having concurrency within an action, and can also be used as a checkpoint mechanism. An action without a parent is called a *topaction*. A handler call is performed as a subaction of the caller, which ensures that a call is performed either zero or one time. If the called guardian is up and accessible, the call is performed once. Otherwise, the call subaction aborts, ensuring that the call has no effect.

During the current year, we continued working on the Argus implementation, and succeeded in running distributed programs for the first time. We also continued developing the debugger, and began experimenting with some applications written in Argus. In addition, we studied issues in program structure for distributed systems, completed the design of a debugging system that takes advantage of information about atomic actions, developed a concurrency control method for long read-only actions, and developed an algorithm for determining approximate agreement. These accomplishments are discussed in more detail below.

2. IMPLEMENTATION

During the preceding year, we had succeed in getting individual guardians to run in isolation. This meant that we had to implement enough of the action system that actions running at a single guardian could lock objects and commit or abort. However, we did not need to worry about committing actions that visited many guardians, nor did we need to run handler calls. This year we continued our work on the implementation, and now have enough of the system running that users can begin to experiment with Argus.

Our implementation of handler calls works as follows. The arguments of handler calls are communicated by value, which ensures that objects belonging to a guardian cannot be accessed directly by any other guardian. The Argus system is responsible for constructing and sending the messages that must be exchanged in order for the called guardian to be aware of the call, and the calling guardian to find out about the results.

Values of a type are exchanged by using a canonical *external representation* for communication. Each implementation of the type provides an *encode* operation that maps from the internal representation to the external representation and a *decode* operation to perform the reverse mapping. This approach allows each implementation to be developed independently of other implementations of its type; the implementations need agree only on the external representation. Our method allows for sharing structure to be preserved. For example, if an argument to a call is a graph with a shared subnode, then we will preserve this structure in the message so that the graph that arrives at the called guardian also contains the shared subnode. The approach is described in [23].

As mentioned above, each handler call in Argus is run as a subaction of the calling action. Our communication method relies on this. A call message is divided into packets, if necessary. Then each packet is sent as a datagram. Although it is quite likely that each datagram will arrive at its destination, assuming the destination is running, delivery is not guaranteed. However, we do not buffer the datagrams after they are sent. Instead, if the calling guardian discovers that a call did not arrive, it simply redoes the entire call. There are two reasons why this is possible:

1) The calling program must wait until the call returns. Therefore, the arguments are still accessible and furthermore have not been changed. They can simply be re-encoded.

2) Each call runs as a subaction. If the first attempt to do a call does not succeed, we abort its subaction, which ensures that the system state cannot reflect any effects of the first attempt. Aborting is necessary because the call may have been performed either totally or partially, even though it appears otherwise to the caller. Since the first attempt aborted, it is safe to make a second attempt. The second attempt runs as a subaction that is distinct from the one used in the first attempt.

Not buffering the datagrams is a good idea provided they are delivered with high probability. If not, the work required to redo the call is too large. We do not know yet whether the probability of delivery is high enough. If we discover that it is not, we will add flow control at some future time. Notice that we are doing an end-to-end approach [39] here, with checking at the top level, i.e., the call. As is always the case with this approach, lower level checking may be needed to improve the performance.

We are also developing an Argus debugger. We want debugging of distributed programs to be similar to debugging of sequential programs, with breakpoints settable at calls and returns, etc. Of course, it is necessary that information about all breakpoints be displayed at the same terminal, even if the breakpoints take place at guardians at different nodes.

To this end, we are developing a distributed debugging system. The debugger actually runs in each guardian being debugged. To make display of information convenient, we are developing a server, called the X window system, that controls access to a bitmap display and provides primitives for window management. The server can be used by the debugger at the various guardians by sending requests to it, so it supports our goal of displaying information at a single device.

The debugger has two additional goals of interest. First, it allows us to switch a guardian's code between production mode and debugging mode. In a system like Argus, individual components are long lived. A production component must run efficiently; we do not want to pay for the ability to debug the component in either space or time. On the other hand, if a problem develops in the component, we want to switch to debugging mode. Our implementation will allow the debugger to be brought in on the fly, and swapped out again later.

The second goal is to have all interaction with debugger be strongly typed. For example, this allows the debugger to display a string in a meaningful way, e.g., "abc", instead of as a bunch of bits or integers. Also, since Argus supports user-defined abstract types, we want the same ability for these types. A thesis completed this year by J. Restivo [38] contains a design for this part of the debugger.

A special problem in a distributed system is that a single type may have different implementations at different nodes. This is one reason why we chose to have the debugger actually reside in each guardian. The debugger inside a guardian can deal with the internal representation chosen for a type in that guardian in a straightforward manner.

3. LINGUISTIC ISSUES IN DISTRIBUTED PROGRAMS

A programming language for distributed computing must provide a set of communication primitives and a structure for processes. In work done this year (see [33]), we examined one possible choice, synchronous communication primitives (such as rendezvous or remote procedure call) in combination with modules that encompass a fixed number of processes (such as Ada tasks or UNIX processes), and we evaluated the program structures needed to manage certain common concurrency control problems. Our concern here is expressive power: the degree to which common problems may be solved in a straightforward and efficient manner. Our analysis indicates that the combination of synchronous communication with static process structure imposes complex and indirect solutions, and therefore is poorly suited for applications such as distributed programs in which concurrency is important. To provide adequate expressive power, a language for distributed programming should abandon either synchronous communication primitives or the static process structure.

Our analysis is based on the *client/server model*, in which a distributed program is organized as a collection of modules, each of which resides at a single node in the network. Modules do not share data directly; instead they communicate through messages. Modules act as clients and as servers. A *client* module makes use of services provided by other modules, while a *server* module provides services to others by encapsulating a resource, providing synchronization, protection, and crash recovery. The client/server model is hierarchical: a particular module may be both a client and a server.

Although other models for concurrent computation have been proposed, the hierarchical client/server model has come to be the standard model for structuring distributed programs. Lauer and Needham [29] argued that the client/server model is equivalent (with respect to expressive power) to a model of computation in which modules communicate through shared data. Nevertheless, the client/server model is more appropriate for distributed systems because speed and bandwidth are typically more critical in the connection between a module and its local data than between distinct modules. Although certain specialized applications may fit naturally into alternative structures such as pipelines or distributed coroutines, the hierarchical client/server model encompasses a large class of distributed programs.

There are two main alternatives for communication primitives: synchronous and asynchronous. *Synchronous* mechanisms provide a single primitive for sending a request and receiving the associated response. The client's process is blocked until the server's response is received. Examples of synchronous mechanisms include procedure call, remote procedure call [36], and rendezvous [11]. Languages that use synchronous mechanisms for communication include Mesa [35], DP [6], Ada [11], SR [2],[1] MP [41] and Argus [30].

Asynchronous communication mechanisms typically take the form of distinct *send* and *receive* primitives for originating requests and acquiring responses. A client process executing a send is blocked either until the request is constructed, or until the message is delivered (as in CSP [24]). The client acquires the response by executing the receive primitive. After executing a send and before executing the receive, the client may undertake other activity, perhaps executing other sends and receives. Languages that use send/receive include CSP and PLITS [17].

There are also two choices for process structure within a module. Modules having a *static* structure encompass a fixed number of threads of control (usually one) that are available to respond to clients' requests. The programmer is responsible for

[1]In addition to remote call, SR provides the ability to send and request without waiting for the response.

multiplexing these threads among a varying number of activities. Examples of modules having a static process structure include Ada tasks, DP monitors, and CSP processes, where there is just one process per module, and SR, where there may be multiple processes per module. (Although an Ada task can create subsidiary tasks dynamically, these subsidiary tasks cannot be addressed by the client as a group.) The multiplexing mechanisms available to the programmer include guarded commands and condition variables.

An alternative to the static structure is the *dynamic* structure, in which a variable number of processes may execute within a module. (Note that a module's process structure is independent of the encompassing system's process structure; the number of processes executing within a module may vary dynamically even if the overall number of processes in the system is fixed.) The system is responsible for scheduling, but the programmer must synchronize the use of shared data. MP, Argus, and Mesa are examples of languages in which the basic modular unit encompasses a dynamic process structure.

All four combinations of communication and process structure are possible. Figure 10-1 shows the combinations provided by several languages. We are not aware of any languages that provide asynchronous communication with dynamic processes. Although such languages may exist, this combination appears to provide an embarrassment of riches not needed for expressive power.

	Static	*Dynamic*
Synchronous	Ada tasks, DP, SR	Argus, Mesa, Starmod, MP
Asynchronous	CSP, PLITS, Starmod	

Figure 10-1: Communication and Process Structure in Some Languages

To determine whether a particular structure is adequate, it is necessary to consider the concurrency requirements of the modules that make up a distributed program. Our discussion centers on the concurrency needs of modules that act as both clients and servers; any linguistic mechanism that provides adequate concurrency for such a module will also provide adequate concurrency for a module that acts only as a client or only as a server.

The principal concurrency requirement is the following: if one activity within a module becomes blocked, other activities should be able to make progress. A system in which modules are unable to set aside blocked activities may suffer from unnecessary deadlocks and low throughput. For example, suppose a server cannot carry out one client's request because another client has locked a needed resource. If the server then becomes blocked, rendering it unable to accept a request to release the resource, a deadlock will occur that was otherwise avoidable. Even when there is no prospect of deadlock, a module that remains idle when there is work to be done is a performance bottleneck.

We can distinguish two common situations in which an activity within a module might be blocked:

1) *Local Delay*: A local resource needed by the current activity is found to be unavailable. For example, a file server may discover that the request on which it is working must read a file that is currently open for writing by another client. In this situation, the file server should temporarily set aside the blocked activity, turning its attention to requests from other clients.

2) *Remote Delay*: The module makes a call to another module, where a delay is encountered. The delay may simply be the communication delay, which can be large in some networks. Alternatively, the delay may occur because the called module is busy with another request or must perform considerable computing in response to the request. While the calling module is waiting for a response, it should be able to work on other activities.

In [33] we analyze several program structures that might be used to meet the concurrency requirement stated above. Based on this analysis, we found that languages that combine synchronous communication with a static process structure can provide adequate expressive power for avoiding local delays, as illustrated by monitors and by languages with a fully general guarded command mechanism. These languages do not, however, provide adequate expressive power for avoiding remote delays. For monitors, this problem has come to be known as the *nested monitor call* problem [22], [34]. Our analysis shows that these languages suffer from an analogous problem.

We believe that it is necessary to abandon either synchronous communication or static process structure, but a well-designed language need not abandon both. If the language provides asynchronous communication primitives, then a server can work on one client's request in parallel with waiting on behalf of some other client. However, the need to perform explicit multiplexing is a disadvantage of this choice of primitives.

Alternatively, if a language provides dynamic process creation within a single module, as in Argus and Mesa, then the advantages of synchronous communication can be retained. When a call message arrives at a module, a new process is created automatically (or allocated from a pool of processes) to carry out the request. When the process has completed the call, a reply message is sent back to the caller, and the process is destroyed or returned to the process pool. Only two messages are needed to carry out the call. The new process ensures that the module is not blocked even if the request encounters a delay. The process must synchronize with other processes in the module, but all the processes are defined within a single module, which facilitates reasoning about correctness. Finally, a dynamic process structure can mask delays that result from the use of local input/output devices, and may permit multiprocessor nodes to be used to advantage. We think synchronous communication with dynamic processes is a better choice than asynchronous communication with static processes; a detailed justification for this opinion is given in [31].

4. A DEBUGGING METHOD TAILORED TO ATOMIC ACTIONS

S.-Y. Chiu completed a thesis on debugging with a system that supports atomic transactions [10]. The main conclusion of this thesis is that structuring activities as nested atomic actions makes debugging concurrent and distributed programs easier. The thesis develops a method for debugging computations in Argus. Using the method, a person debugs a concurrent and distributed computation much like he or she would debug executions of traditional sequential programs. The method and implementation approach presented in this thesis are applicable to other action systems, even though details may differ.

The method uses action trees, together with a serialization order, to summarize a computation to the user. A *computation* is a group of topactions; an *action's tree* is the hierarchy of subactions that are contained within the action; a *serialization order* is the ordering of sibling subactions and topactions in some equivalent serial execution.

In the method, a node in an action tree is viewed as a map from a *pre-* state to a *post-*state. (A state is a map from object identifiers to object values.) The value of an object in the pre-state of an action A is the net effect at the object of all modifications made by actions serialized before A and all modifications made by ancestors before A ran. The value of an object in the post-state of A is the pre-A value of the object updated by the modifications of A, if any. A person who is debugging an action will be interested primarily in objects that are accessible from the action's environment, i.e., variables that are global to the action, as well as the arguments and input to the action and the results and output from the action.

A user follows the progress of an action by doing a *serial walk* of the action's tree. In a serial walk, committed subactions of a node are visited in serialization order. Where appropriate, the user can choose to ignore the details of any subaction in the tree.

Debugging a faulty action involves three phases. In *Phase Zero*, some subset of an action's history is collected as the action runs. The history that is collected is used for supporting the next two phases of the method, and is discussed further below. Phase Zero history is saved for all actions that run in a program of interest. Older saved history, however, is discarded as space is needed. Phase Zero does not require any user intervention.

Phase One is the first of the two interactive phases in the method. In Phase One, the user uses the computation's action trees, the serialization order of siblings, and other saved history to narrow a bug to as "small" an action in a tree as possible: this is the youngest action that maps a "correct" pre-state to an "incorrect" post-state. (Action A is younger than action B if A is a descendant of B.) The user, not the debugging system, decides whether an action's pre- or post-state is correct. The debugging system helps by displaying the value of an object in an action's pre- or post-state on request from the user.

Sometimes the bug becomes obvious once the user narrows it to an action; at other times, it is not. If the bug is still not obvious after the fault has been narrowed to an action, the user moves on to *Phase Two.* In this phase, the faulty action's code is re-executed, using the data collected during the original computation to recreate the action's history. A single thread of control is used when an action is retraced; concurrent siblings are retraced in their serialization order. The user uses the usual break-and-examine tools of sequential debugging (e.g., breakpointing and single-stepping) on the single thread of control in the re-execution to isolate the bug. Because we assume the action will perform the same when retried, the method is only suited to deterministic programs.

4.1. Implementing the Method

In Argus, actions are guaranteed to be atomic only if they share *atomic objects.* Atomic objects are objects that provide synchronization and recovery for actions that access them. Atomic objects that are built into Argus use two-phase locking [16] for concurrency control and back-up versions for recovery. Locks are automatically acquired and versions automatically created when actions invoke operations on atomic objects. The run-time system dispenses of an action's locks and versions appropriately when the action terminates. Argus also supports user defined atomic objects.

We introduce timestamps and multiple versions into Argus so the debugging system can provide values of atomic objects in an action's pre- and post- state, and can retrace action histories. The recovery versions that are created for built-in atomic objects to ensure action recovery on aborts are mainly what the debugging system needs. The timestamps are generated with Lamport clocks [26] and are assigned to actions when the actions terminate. Since Argus releases locks only when an action terminates, these timestamps order the lock points of actions and give a valid serialization order [3], [16]. These timestamps, together with versions, can be used to give the pre- and post-values of atomic objects for topactions as well as subactions, regardless of whether the actions committed or aborted, and can be used to retrace an action's history in serial-walk order. The thesis ([10]) explains in detail how information is saved and how Lamport clocks are used to order actions.

In the retrace, the appropriate portion of the program is re-executed. Even though the same objects are referenced in the re-execution, "old" recovery versions are used so that a retrace of an action reads the same values as the original computation. A retrace accesses only saved history in an object and, therefore, does not disrupt other actions. Recovery versions and timestamps have been used in concurrency control; our work is the first detailed design that uses them for debugging nested actions.

5. DISTRIBUTED VERSION MANAGEMENT FOR READ-ONLY ACTIONS

One particularly useful kind of atomic action is a read-only action: an action that reads, but does not modify, part of the state of the system. A read-only action can be used to take a checkpoint of the system state for recovering from subsequent catastrophic hardware or software failures, to audit a system (e.g., checking whether the state is consistent), or simply to extract information about the current state of the system. Indeed, many applications are characterized by a predominance of read-only actions.

As noted by Lamport [25], traditional locking implementations (e.g., see [16]) of atomic actions have difficulties with read-only actions. For example, a read-only action that reads most or all of the system state may tie up a large part of the system for a long time, possibly delaying many update actions. In addition, a read-only action that lasts a long time may be likely to be aborted because of a deadlock or a hardware failure, making it unlikely that the action will ever finish.

We have developed three new concurrency control protocols that permit read-only actions to run without interfering with update actions or with each other. The protocols make use of the semantic information that read-only actions make no changes to the state of the system, and require read-only actions to be identified to the system before they begin execution. The protocols are based on the idea of maintaining multiple versions of the state of each object in the system, and having read-only actions read old versions while update actions manipulate the "current" version. By preventing interference between a read-only action and other actions, the protocols increase the likelihood that a read-only action will be able to complete successfully. In addition, the performance of update actions is likely to be better when there is no interference from read-only actions [7].

There are several problems that must be solved by a protocol that uses multiple versions to prevent interference among read-only actions and other actions. For example, one must be careful in selecting the old versions to be read by a given read-only action to ensure that the state seen by the action is consistent. In this work we have explored several different levels of consistency, and developed protocols that achieve each. In addition, the need to save old versions to be read by read-only actions introduces a storage management problem, namely how to determine that an old version is no longer needed so that it can be discarded. In this work, we have focused on protocols designed to operate in a distributed system -- one in which the state of the system is distributed among a collection of nodes which communicate via messages -- and considered how to minimize the space needed for old versions of objects.

In Section 5.1 below we discuss several possible correctness criteria. Then, in Section 5.2, we describe one of the protocols from [43]. Details of the other protocols and a discussion of related work can be found in [43].

5.1. Correctness Criteria

We assume that users desire update actions to be serializable and recoverable. In addition, we assume the existence of an invariant that defines the set of consistent states of the system. Given this invariant, we assume that each update action, when run alone and to completion, preserves consistency, and that the system starts in a consistent state. Given these assumptions, we can imagine several possible requirements for read-only actions.

First, we could require simply that the values read by each read-only action be consistent. We call this requirement *consistency.* Consistency demands only that the state seen by a read-only action satisfy the invariant, and not that it bear any particular relation to the values actually written by update actions. It might seem that this requirement is too weak to be useful; however, at least one proposed scheme [20] does not satisfy any stronger requirements. All of the protocols in [43] satisfy stronger, more useful requirements.

Second, we could require that the values read by each read-only action be the result of a serial execution of some subset of the update actions. We call this requirement *external consistency.* External consistency is at least as strong a requirement as consistency, because the result of a serial execution of update actions is always consistent. In fact, it is a stronger requirement, because in a given computation not all consistent states satisfy the requirement of being the result of a serial execution of some subset of the update actions in the computation.

Finally, we could require *serializability* -- i.e., all actions in a computation, both updates and read-only actions, must be serializable as a group. As the following example illustrates, serializability is a stronger requirement than external consistency. Consider the following execution, involving two update actions, A and B, two read-only actions, R and S, and two objects, X and Y, and assume that X and Y both start with initial value 0:

> Action A writes 1 into X.
> Action B writes 1 into Y.
> Action R reads 1 from X.
> Action R reads 0 from Y.
> Action S reads 0 from X.
> Action S reads 1 from Y.

The values read by R are the result of a serial execution of the single action A, while the values read by S are the result of a serial execution of the single action B. Thus, the execution is externally consistent. However, there is no single serial execution of all four actions in which each read operation returns the most recent value written, so the execution is not serializable.

From the point of view of trying to understand the interactions among actions in an execution, or trying to reconcile the values read by two different read-only actions, serializability is clearly preferable to external consistency. Hence, one might wonder why external consistency is interesting. As discussed in [43], it can be cheaper to ensure external consistency than to ensure serializability. For those applications that can tolerate a weaker requirement, such as external consistency, the potential performance gain could be significant.

In addition to the requirements above, we might also require the values read by a read-only action to be reasonably *current*: they should reflect the modifications of at least all update actions that finished before the read-only action started. This requirement is useful because it rules out a trivial solution, in which we save a copy of the initial state of the system, and read-only actions read from that copy. This trivial solution, while it ensures serializability and prevents read-only actions from interfering with update actions, is not useful for most applications.

5.2. A Protocol

In this section we present one of the three protocols from [43]. The state of the system is contained in *objects*; each object provides operations by which actions can read and update the object's state. (An object might be a node in the distributed system, or part of the state at a node; we will not discuss here the factors involved in choosing an appropriate granularity.) The protocol requires that objects maintain multiple versions of their state so that read-only actions can read old versions while new updates continue to run using the current version. An important issue is how to manage versions: when must a version be kept, and when can an old version be thrown away? The problems of version management have a strong influence on the design of the algorithms.

To simplify the presentation, we assume a simple model for objects, with each object providing read and write operations. We also assume that two-phase locking [16] is used for update actions, with exclusive locks used for write operations, and shared locks for read operations. As discussed in [1], [40], [42], [44], greater concurrency among update actions can be permitted if more information about the specification of each object is used. The protocols presented here are easily adapted to use this kind of type-specific information.

In addition, we assume that update actions use a two-phase commit protocol [21], [28] and some form of crash-tolerant storage (e.g., stable storage [28]) to achieve resilience to node failures. Our protocol works in part by piggybacking information on messages used in the two-phase commit protocol.

The protocol described here is similar to the protocols of [4], [8], [9], [15]. It uses timestamps and multiple versions to ensure serializability of all actions while permitting a read-only action to run without interfering with other actions. Our contribution consists of an "initiation phase" for read-only actions that allows us to tell when a version might be needed by some read-only action. For some applications, however, the communication cost of the initiation phase could be too high. The remaining two protocols in [43] represent attempts to avoid this cost. Both work by a propagation technique, analogous to that in [20]. The first ensures only external consistency, and is relatively cheap. The second ensures serializability, but is more expensive.

The protocol described here combines timestamps and locking. Each action is assigned a unique timestamp. Update actions are synchronized using ordinary two-phase locking [16]. The timestamp for an update action is chosen as it commits; as discussed below, care is taken to ensure that the timestamp order on committed update actions is consistent with the partial order induced by the locking protocol. (Updates that abort can be ignored; versions written by such updates are simply discarded.) Read-only actions are handled differently: the timestamp for a read-only action is chosen when it begins. When a read-only action with timestamp T attempts to read an object, the version of the object written by the action with the largest timestamp less than T is selected as the value to be returned by the read operation. The initiation protocol for read-only actions, described below, ensures that objects retain the versions needed by a read-only action while the action is active.

As noted above, operations invoked by update actions are processed using ordinary two-phase locking: when an update action invokes a read operation on an object, it first waits until it can lock the object in read mode, and then reads the most recently written version. When an update action invokes a write operation, it locks the object in write mode, and then creates a new version. If the action later aborts, the newly created version will be discarded. For now, let us assume that if an update action commits, all versions that it created will be retained forever. Shortly we will explain how to retain only those versions that are actually needed by read-only actions, and how to discard a version that is no longer needed.

To generate timestamps for update actions, we maintain a timestamp for each object. The timestamp for object X represents the maximum of the timestamps of update actions that have accessed X and committed, and the timestamps of read-only actions that have accessed X (regardless of whether they have already committed). Timestamps for update actions are generated as follows: during the first phase of the two-phase commit protocol for an update action, the action collects the timestamps of all objects that it accessed. (This is easily done by piggybacking the timestamps on the normal messages of the two-phase commit protocol.) Upon receiving timestamps from all accessed objects, the action chooses a unique timestamp greater than all the timestamps received from the objects. Then, in the second phase of the commit protocol, the chosen timestamp is broadcast to all objects (piggybacked on the message containing the decision to commit). Each object, upon receiving this message, updates its timestamp to the maximum of its current value and the action's timestamp,[2] marks any versions written by the action with the action's timestamp, and then releases any locks held by the action. It is easy to show that any conflicts between two update actions are reflected in the relative order of their timestamps, and hence that updates are serializable in timestamp order.

Timestamps are chosen for read-only actions using the following initiation protocol: when a read-only action begins, it sends messages to all objects that it might access asking for their current timestamps. When an object receives such a request, it records the action's identifier along with the object's current timestamp. The object then responds to the request with its current timestamp. After receiving responses from all objects, the action chooses a unique timestamp greater than all the responses. The timestamp recorded for the action at each object is thus a lower bound on the timestamp chosen by the action. As we discuss below, this information is used in deciding when to retain or discard a version.

When a read-only action with timestamp T invokes a read operation on an object, it chooses the version of the object with the largest timestamp less than T. After finding this version, the read operation updates the object's timestamp to the maximum of its current value and the read-only action's timestamp, and then returns the version as the value read. Updating the object's timestamp forces update actions that commit later to choose timestamps larger than T, ensuring that the version that should be selected for the read-only action does not change, and thus that all actions remain serializable in timestamp order.

There is one potential race condition for which we must provide. When a read-only action with timestamp T invokes a read operation on an object, an update action might be in the middle of its commit protocol. Unless we can be sure that the timestamp that will be chosen by the update will be greater than T, we cannot tell whether to read the version written by the committing update or some earlier version. In this case, the read-only action must wait until the update action finishes its commit protocol. We can reduce the amount of waiting by recording, for each committing update action at the object, the timestamp returned to it in the first phase of its commit protocol. This timestamp gives a lower bound on the timestamp that will eventually be chosen by the update action. The read-only action can proceed if its timestamp is less than the lower bound for each committing update action.

It is easy to show that the above protocol ensures serializability. The protocol for generating timestamps for update actions ensures that any conflicts between updates will be reflected in the relative order of their timestamps. Read-only actions then read

[2]The object's current timestamp when it receives the commit message from an update action could be greater than the timestamp chosen by the action if another action was executing its commit protocol at the object concurrently.

versions of objects consistent with all actions executing in timestamp order. Notice, in addition, that the protocol ensures that read-only actions are given reasonably current views: the version of an object read by a read-only action will be no older than the version that was current at the time that the action initiated at the object.

To this point we have described the protocol as if all versions were retained forever. We will now show how versions can be discarded when they are not needed by a read-only action. Recall that each object keeps track of the read-only actions that have initiated at the object, along with a lower bound on the timestamp chosen by each action. Objects can use the following rule to decide which versions to keep and which to discard: a version with timestamp T must be retained (1) if there is no version with timestamp greater than T (in which case it is the current version and is needed for update actions), or (2) if there is a version with timestamp T1 > T, and there is an active read-only action whose timestamp might be between T and T1. By having a read-only action inform objects when it completes, objects can discard versions that are no longer needed. This process of informing objects that a read-only action has completed need not be performed synchronously with the commit of the action. It imposes some overhead on the system, but can be done by piggybacking information on existing messages, or sending messages when the system load is low.

The protocol described here is effective at minimizing the amount of storage needed for versions. For example, unlike the "version pool" scheme in [8], [9], it is not necessary to discard a version that is needed by an active read-only action because buffer space is being used by a version that no action wants to read. However, ensuring that each object knows which versions are needed at any point in time has an associated cost, namely the initiation phase for read-only actions. A read-only action cannot begin executing until it has chosen a timestamp, a process that requires communicating with all objects that it might access. If the set of objects that might be accessed by a read-only action is relatively small, then the cost of initiation might not be too great. However, if the set is large, or if it is difficult to predict (in which case the estimated set must be large to be safe), then the initiation phase could be a performance bottleneck. The other protocols described in [43] avoid the problems of an initiation phase.

6. REACHING APPROXIMATE AGREEMENT IN THE PRESENCE OF FAULTS

In collaboration with Danny Dolev, Nancy Lynch, Shlomit Pinter, and Eugene Stark, William Weihl developed new algorithms and lower bounds for the problem of reaching approximate agreement in a distributed system with faulty processes. An earlier version of this work appeared in [13]; the new algorithms and lower bounds appeared in [14].

In designing fault-tolerant distributed systems, one often encounters questions of agreement among processes. In the Byzantine Generals problem [27], [37], the objective is for nonfaulty processes to agree on a value, in spite of the presence of a small number of "Byzantine" types of faults -- completely arbitrary, even possibly malicious, behavior. Several variations on the problem can be considered -- the model can be synchronous or asynchronous, and either exact or approximate agreement can be demanded. In this work, we consider a variant on the traditional Byzantine Generals problem, in which processes start with arbitrary real values, and where approximate, rather than exact, agreement is the desired goal. Approximate agreement can be used, for example, for clock synchronization and for stabilization of input from sensors.

We assume a model in which processes can send messages containing arbitrary real values, and can store arbitrary real values as well. We assume that each process starts with an arbitrary real value. For any preassigned $\epsilon \geq 0$ (as small as desired), an *approximate agreement algorithm* must satisfy the following two conditions:

- Agreement: All nonfaulty processes eventually halt with output values that are within ϵ of each other.

- Validity: The value output by each nonfaulty process must be in the range of initial values of the nonfaulty processes.

Thus, in particular, if all nonfaulty processes should happen to start with the same initial value, the final values are all required to be the same as the common initial value. This is consistent with the usual requirements for Byzantine agreement algorithms. However, should the nonfaulty processes start with different values, we do not require that the nonfaulty processes agree on a unique final value.

We consider both synchronous and asynchronous versions of the problem. Systems in which there is a finite bounded delay on the operations of the processes and on their intercommunication are said to be synchronous. In such systems, unannounced process deaths, as well as long delays, are considered to be faults. For synchronous systems, we give a simple and rather efficient algorithm for achieving approximate agreement. This algorithm works by successive approximation, with a provable convergence rate that depends on the ratio between the number of faults and the total number of processes. The algorithm is guaranteed to converge in the case where the total number of processes is more than three times the number of possible faults. Termination is achieved using a technique that ensures that all nonfaulty processes halt, but allows different processes to terminate at different times.

For asynchronous systems, in which a very slow process cannot be distinguished from a dead process, no exact agreement can be achieved [19], even if no malicious failures occur [12]. An interesting contrast to the results in [12], [19] is our second algorithm, which enables processes in an asynchronous system to get as close to agreement as one chooses. Our algorithm for the asynchronous case also works by successive approximation. In this case, however, the total number of processes required by the algorithm is more than five times the number of possible faults. As in the synchronous case, we achieve termination using a technique that ensures that all nonfaulty processes halt, but permits different processes to terminate at different times. It is possible to achieve simultaneous termination in the synchronous case; the technique used for simultaneous termination, however, does not extend to the asynchronous case.

Our algorithms to obtain approximate agreement are of a very simple form. Namely, at each round until termination is reached, each process sends its latest value to all processes (including itself). On receipt of a vector V of values, the process computes a certain function f(V) as its next value. The function f is a kind of averaging function. Here we use functions that are appropriate for handling t faults. We show that these functions have particularly nice approximation behavior. In particular, we show that, for algorithms of a particular form, no approximation function can provide uniformly faster convergence than the functions used in this paper. An earlier paper [13] presented similar algorithms, but used approximation functions that provided slower convergence than achieved by the functions used in this paper.

For the synchronous case, it is not difficult to show that 3t + 1 processes are necessary to solve the approximate agreement problem. The proof is an adaptation of the lower bound proof in [27], and appears in [18]. For the asynchronous case, our algorithm uses 5t+1 processes, which is not optimal. In fact, it appears possible to reduce the number of processes to as few as 3t + 1. This reduction is obtained using a more complex algorithm, based on some of the interesting ideas of [5]. This algorithm has a slower rate of convergence than ours.

References

1. Allchin, J. E. and McKendry, M. S. "Synchronization and Recovery of Actions," in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, 1983, 31-44.

2. Andrews, G. R. "Synchronizing Resources," *ACM Transactions on Programming Languages and Systems 3*, 4 (October 1981), 405-43.

3. Bernstein, P. A. and Goodman, N. "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys 13*, 2 (June 1981), 185-221.

4. Bernstein, P. A. and Goodman, N. "Multiversion Concurrency Control -- Theory and Algorithms," *ACM Transactions on Database Systems 8,* 4 (December 1983), 465-483.

5. Bracha, G. "An Asynchronous L(n - 1)/3J -Resilient Consensus Protocol," *Proceedings of the 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 1984, 154-162.

6. Brinch Hansen, P. "Distributed Processes: A Concurrent Programming Concept," *Communications of the ACM 21*, 11 (November 1978), 934-941.

7. Carey, M. J., and Muhanna, W. A. "The Performance of Multi-version Concurrency Control Algorithms," Computer Sciences Technical Report 550, University of Wisconsin at Madison, 1984.

8. Chan, A., et al. "The Implementation of an Integrated Concurrency Control and Recovery Scheme," ACM SIGMOD Conference on Management of Data, 1982, 184-191.

9. Chan, A. and Gray, R. "Implementing Distributed Read-Only Transactions," *IEEE Transactions on Software Engineering SE-11*, 2 (February 1985), 205-212.

10. Chiu, S. Y. "Debugging Distributed Computations in a Nested Atomic Action System," MIT/LCS/TR-327, MIT Laboratory for Computer Science, Cambridge, MA, 1985.

11. Department of Defense. "Reference Manual for the ADA Programming Language," Report ANSI/MIL-STD-1815A-1983, Washington, DC, 1983.

12. Dolev, D., Dwork, C., and Stockmeyer, L. "On the Minimal Synchronism Needed for Distributed Consensus," *Proceedings of 24th Annual Symposium on Foundations of Computer Science*, 1983, 393-402.

13. Dolev, D., Lynch, N. A., Pinter, S., Stark, E. W. and Weihl, W. E. "Reaching Approximate Agreement in the Presence of Faults," *Proceedings of 3rd Annual IEEE Symposium on Reliability in Distributed Software and Database Systems*, 1983, 145-154.

14. Dolev, D., Lynch, N. A., Pinter, S., Stark, E. W. and Weihl, W. E. "Reaching Approximate Agreement in the Presence of Faults," *Journal of the ACM 33*, 3 (July 1986), 499-516.

15. DuBourdieu, D.J. "Implementation of Distributed Transactions," in *Proceedings of the Sixth Berkeley Workshop on Distributed Data Management and Computer Network*, 1982, 81-94.

16. Eswaran, K. P., Gray, J. N., Lorie, R. A. and Traiger, I. L. "The Notions of Consistency and Predicate Locks in a Database System," *Communications of the ACM 19*, 11 (November 1976), 624-633.

17. Feldman, J. A. "High Level Programming for Distributed Computing," *Communications of the ACM 22*, 6 (June 1979), 353-368.

18. Fischer, M., Lynch, N. A. and Merritt, M. "Shifting Scenarios: Easy Impossibility Proofs for Distributed Consensus Problems," submitted for publication.

19. Fischer, M., Lynch, N. A. and Paterson, M. S. "Impossibility of Distributed Consensus With One Faulty Process," *Proceedings of 2nd ACM Symposium on Principles of Database Systems*, 1983.

20. Fischer, M. J., Griffeth, N. D. and Lynch, N. A. "Global States of a Distributed System," *IEEE Transactions on Software Engineering SE-8* 3 (May 1982), 198-202.

21. Gray, J. "Notes on Database Operating Systems," *Operating Systems -- An Advanced Course, Lecture Notes in Computer Science 60.* Berlin: Springer-Verlag, 1978.

22. Haddon, B. K. "Nested Monitor Calls," *ACM Operating Systems Review 11*, 4 (October 1977), 18-23.

23. Herlihy, M. and Liskov, B. "A Value Transmission Method for Abstract Data Types," *ACM Transactions on Programming Languages and Systems 4*, 4 (October 1982), 527-551.

24. Hoare, C. A. R. "Communicating Sequential Processes," *Communications of the ACM 21*, 8 (August 1978), 666-677.

25. Lamport, L. "Towards a Theory of Correctness for Multi-User Data Base Systems," Report CA-7610-0712, Massachusetts Computer Associates, 1976.

26. Lamport, L. "Time, Clocks, and the Ordering of Events in a Distributed Systems," *Communications of the ACM 21*, 7 (July 1978), 558-565.

27. Lamport, L., Shostak, R. and Pease, M. "The Byzantine Generals Problem," *ACM Transactions on Programming Languages and Systems 4*, 2, 1982, 382-401.

28. Lampson, B. "Atomic Transactions," in *Distributed Systems: Architecture and Implementation, Lecture Notes in Computer Science 105*, (Ed.), Goos and Hartmanis. Berlin: Springer-Verlag, 1981, 246-265.

29. Lauer, P. E. and Needham, R. M. "On the Duality of Operating Systems Structures," *Proceedings of the Second International Symposium on Operating Systems Structures*, 1978; also *ACM Operating Systems Review 13*, 2 (April 1979), 3-19.

30. Liskov, B. and Scheifler, R. W. "Guardians and Actions: Linguistic Support for Robust, Distributed Programs," *ACM Transactions on Programming Languages and Systems 5*, 3 (July 1983), 381-404. Also published as Computation Structures Group Memo 210-1, MIT Laboratory for Computer Science, Cambridge, MA, August 1982.

31. Liskov, B. and Herlihy, M. P. "Issues in Process and Communication Structure for Distributed Programs," *Proceedings of the Third Symposium on Reliability in Distributed Software and Database Systems*, 1983. Also Programming Methodology Group Memo 38, MIT Laboratory for Computer Science, Cambridge, MA, July 1983.

32. Liskov, B. "Overview of the Argus Language and System," Programming Methodology Group Memo 40, MIT Laboratory for Computer Science, Cambridge, MA, 1984.

33. Liskov, B., Herlihy, M. and Gilbert, L. "Limitations of Synchronous Communication with Static Process Structure in Languages for Distributed Computing," *Proceedings of the 13th ACM SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, January 1986. Also Programming Methodology Group Memo 41-1, MIT Laboratory for Computer Science, Cambridge, MA, October 1985.

34. Lister, A. "The Problem of Nested Monitor Calls," *ACM Operating Systems Review 11*, 2 (July 1977), 5-7.

35. Mitchell, J. G., Maybury, W. and Sweet, R. "Mesa Language Manual, Version 5.0," Technical Report CSL-79-3, Xerox Palo Alto Research Center, Palo Alto, CA, April 1979.

36. Nelson, B. "Remote Procedure Call," Technical Report CMU-CS-81-119, Carnegie Mellon University, Pittsburgh, PA, 1981.

37. Pease, M., Shostak, R. and Lamport, L. "Reaching Agreement in the Presence of Faults," *Journal of the ACM 27*, 2, 1980, 228-234.

38. Restivo, J. P. "Addition of Type Information to the Argus Debugger," S.M. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

39. Saltzer, J. H., Reed, D. P. and Clark, D. D. "End-to-End Arguments in System Design," *ACM Transactions on Computer Systems 2*, 4 (November 1984), 277-288.

40. Schwarz, P. and Spector, A. "Synchronizing Shared Abstract Types," *ACM Transactions on Computer Systems 2*, 3 (August 1984).

41. Segall, Z. and Rudolph, L. "PIE -- A Programming and Instrumentation Environment for Parallel Processing, Technical Report CMU-CS-85-128, Carnegie Mellon University, Pittsburgh, PA, 1985.

42. Weihl, W. E. "Specification and Implementation of Atomic Data Types," MIT/LCS/TR-314, MIT Laboratory for Computer Science, Cambridge, MA, 1984.

43. Weihl, W. E. "Distributed Version Management for Read-Only Actions," *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, August 1985, 122-135.

44. Weihl, W. and Liskov, B. "Implementation of Resilient, Atomic Data Types," *ACM Transactions on Programming Languages and Systems 7*, 2 (April 1985), 244-269.

Publications

1. Allen, L. W. "Design of a Kernel for Argus," Programming Methodology Group Memo 43, MIT Laboratory for Computer Science, Cambridge, MA, June 1985.

2. Dolev, D., Lynch, N. A., Pinter, S., Stark, E. W. and Weihl, W. E. "Reaching Approximate Agreement in the Presence of Faults,"

MIT/LCS/TM-276, MIT Laboratory for Computer Science, Cambridge, MA, February 1985; also submitted for publication.

3. Liskov, B., Herlihy, M. P. and Gilbert, L. "Limitations of Remote Procedure Call and Static Process Structure for Distributed Computing," Programming Methodology Group Memo 41, MIT Laboratory for Computer Science, Cambridge, MA, September 1984.

4. Weihl, W. E. "Data-dependent Concurrency Control and Recovery," *ACM SIGOPS Operating Systems Review*, 19, 1 (January 1985); originally published in Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, August 1983.

5. Weihl, W. E. "Distributed Version Management for Read-Only Actions," Programming Methodology Group Memo 44, MIT Laboratory for Computer Science, Cambridge, MA, June 1985; also *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing*, to appear.

6. Weihl, W. E. "Atomic Data Types," *IEEE Bulletin on Database Engineering*, June 1985.

7. Weihl, W. E. and Liskov, B. "Implementation of Resilient, Atomic Data Types," *ACM Transactions on Programming Languages and Systems*, 7, 2 (April 1985), 244-269.

Theses Completed

1. Allen, Larry W. "Design of a Kernel for Argus," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1985.

2. Anderson, E. E. "Design of a Macintosh CLU Implementation," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1985.

3. Chiu, S. Y. "Debugging Distributed Computations in a Nested Atomic Action System," Ph.D dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, December 1984.

Theses in Progress

1. Wagner, R. M. "Integrating Animation into a Programming Environment," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1985.

Talks

1. Liskov, B. "Argus: The Programming Language and System,"

   State University of New York, Stony Brook, NY, October 1984

   Wang Institute of Graduate Studies, Tyngsboro, MA, October 1984

   University of Pennsylvania, Philadelphia, PA, November 1984

   University of Delaware, Newark, DE, April 1985

2. Liskov, B. "The Argus Language and System: Concepts and Issues, Argus Features, Example, Subsystems, Implementation, User-defined Atomic Data Types, Discussion," Advanced Course on Distributed Systems -- Methods and Tools for Specification, Munich, Germany, April 1985.

REAL TIME SYSTEMS

Academic Staff

M.L. Dertouzos	S.A. Ward, Group Leader
R.H. Halstead	R.E. Zippel
C.J. Terman	

Research Staff

D. Robinow	J. Pezaris
D. Goddeau	

Graduate Students

J. Arnold	M. Ma
E. Bradley	J. Miller
E. Burger	P. Neves
S. Gray	D. Nussbaum
C. Hoffman	P. Nuth
M. Katz	P. Osler
S. Ku	S. Seda
J. Marantz	L. Seiler
C. Kuhlman	J. Sieber
S. Lau	M. St. Pierre
W. Lau	G. Waters

Undergraduate Students

V. Ali	J. Loaiza
D. Becker	A. Magnani
M. Blair	A. Manzoor
E. Burger	K. O'Neill
K. Briggs	S. Raman
S. Cohen	E. Siedman
R. Jenez	A. Sieving
A. Kohli	S. Solomon
B. Lim	J. Vance

W. Liske

M. Wade

Support Staff

A. Forestell
J. Hoppe

S. Thomas

Visitors

T. Goblick
M. Hommel

M. Shen
G. Zientara

1. INTRODUCTION

During the 1984-85 year, VLSI CAD tools and multiprocessor architectures have continued as major research thrusts within RTS. The Nu and Trix projects have been largely completed, and work toward a major new architectural project has begun. Subsequent sections describe each of these activities.

2. SCHEMA

Our work on Schema is now beginning to bear fruit. Large portions of the system have now reached usable states and we are beginning to 1) accommodate the work of other CAD tool designers within Schema and 2) allowing patient designers to use the system for design. In this section, we will describe the basic structure of Schema as it now stands and point out some of the highlights of this year's work.

Schema is an environment for developing knowledge based, computer-aided design tools for electronic systems. The three major goals of its design were:

1) Provide an integrated environment for all aspects of the synthesis and analysis of electronic designs from MSI PC boards through circuit and mask design of VLSI devices.

2) Simplify the development of these synthesis and analysis tools by encouraging and supporting the construction of CAD tools from libraries of standard routines, by using uniform data structures and by providing libraries of advanced control structures appropriate for CAD development.

3) Allow the designer to delay making decisions until necessary; for example, the technology (TTL, ECL, gate array or custom MOS) used in a logic design need not be specified until timing simulations or physical design is begun.

The key to achieving these goals is the development of an *integrated design environment* where design tools can easily communicate and cooperate. This has been achieved by the innovative software architecture used in the development of Schema.

Schema achieves coherence not by specifying the interchange formats that need to be used between different CAD programs, but rather by specifying the data structures the CAD programs should use. Schema specifies a set of abstract data types for dealing with electronic designs, and a set of policies to be used when dealing with the new data types. This approach provides a common layer on which different CAD tools may be built, and it allows the CAD tools to invoke each other and cooperate easily by interchanging pieces of electronic designs.

These data types are implemented in a message passing, object oriented programming style called *flavors*. These structures represent circuit topologies and schematics, mask artwork, floorplans, and simulation waveforms (both digital and analog). Circuit topologies represent the connectivity of a circuit, schematics are representations of the graphic images of a circuit that are drawn on paper. Since these structures are instances of flavors, they also incorporate pieces of code, called ***message handlers***, that allow them to directly provide procedural functionality. That is, a transistor contains the information and *code* required to display itself on the screen, write itself out to a file or participate in a simulation. This raises the semantic level at which the CAD tools deal with objects, simplifying their development. It also allows implementational and operational decision to be delayed and even changed without modifying the code that makes use of them.

2.1. Modules

The basic component of a design in Schema is a *module*. Each module consists of a *topology* and several *descriptions*, e.g., schematics, icons, layouts, and simulation results. Examples of modules in a design include: an inverter, a half adder, an arithmetic logic unit, a data path, a cache, instruction fetch unit, and a memory system. Each of these modules includes not only the schematic (and its corresponding topology), but also the results of various tests that have been performed on the circuit (simulation results), documentation and design notes, and physical specifications (VLSI layouts or PC board designs). The modules represent a complete view of a design component.

The designer rarely interacts directly with the topology portion of a module, but instead deals with the descriptions (schematics). The analysis tools (simulators, timing verifiers and other consistency checkers) work with the topology, and usually use descriptions only for communicating with the designer. The only major exceptions are the physical design tools, VLSI layout system and wire wrap and PCS board systems which, by necessity, must work with the physical descriptions.

The system ensures the topology remains consistent with the description edited by the designer and warns him/her when two descriptions of a design become inconsistent by the use of timestamps and limited edit trails. This division allows the electronic designer to use the most appropriate mechanism for describing the design without worrying about getting formats correct for the CAD tools, and the CAD tool designer deals only with design descriptions that are both appropriate and "pre-parsed."

The topology and its descriptions are implemented as procedures, though they are usually edited via one of the *description editors:* schematic, layout, or waveform. This procedural structure, similar to the approach used in DPL allows a great deal of flexibility parameterizing the different components and provides an excellent point at which to install intelligent synthesis modules. For instance, in an earlier version of Schema, this was used to implement an ALU module that chose different carry look-ahead schemes depending on the width of the data word.

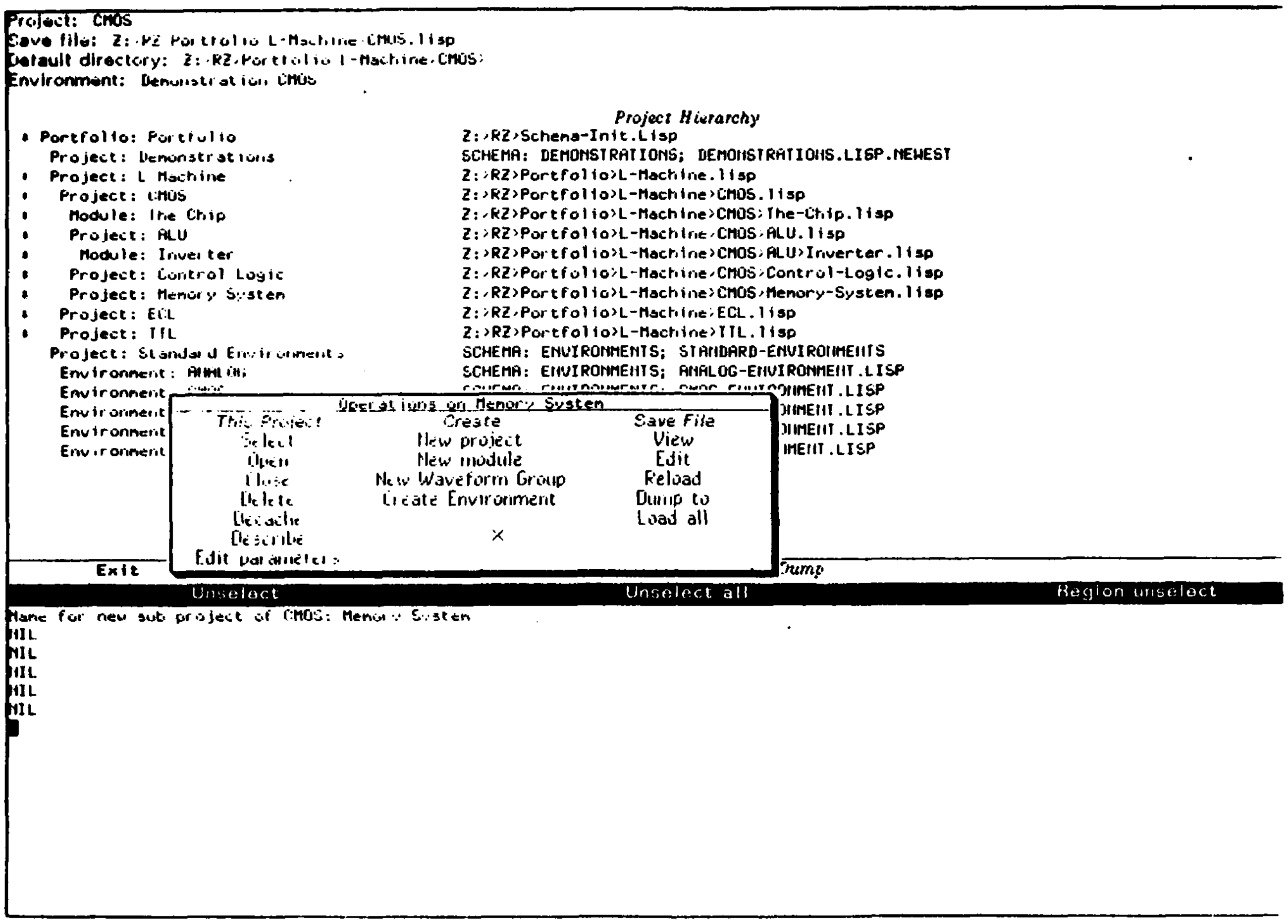
These procedures can also make use of other modules by a slightly stylized version of procedure invocation. The system uses the context of the call to decide which descriptions is appropriate. If a topology invokes another, a module then a sub-topology is generated. When a schematic invokes another module, one of the module's icons is inserted in the schematic. This basic mechanism issued to implement hierarchical designs.

These hierarchical descriptions also incorporate a *multiple viewpoint*, or *slides*, mechanism to allow simulation and analysis modules to annotate the topologies. The multiple viewpoints are used to control the visibility of certain information to the CAD tools. For instance, transient analysis programs like Spice want to be aware of parasitic capacitances and resistances while a simple logic simulator might not. Rather than generating the two different topologies for the different simulators, the same topology is used for both, but the parasitics are only visible when the *transient* viewpoint is made visible by Spice. This way, annotations to the topology made by the two simulators can be easily examined by their counterparts.

2.2. Project/Module Hierarchy

The modules of a design are collected into a *project*, which in turn can be a component of a larger project. For instance, there might be an **L machine** project which is used to hold all the design components of the L machine. Several different versions of the L machine might be designed, so there might be **TTL, CMOS** and **ECS** sub-projects of the L machine. Then within the CMOS project, there might be different projects to contain the design of the data path, control logic, and memory management system. These main modules of each of these projects would be combined by the main module contained in CMOS to produce the final chip.

Each designer maintains his or her own hierarchy of projects. The root of this hierarchy is called *portfolio*. By having sub-projects point to the same *save file*, designers can share projects. Figure 11-1 gives a screen image of Schema just as the designer is making some modification to the memory system project. At the root of the hierarchy is the Portfolio, which appears on the first line actual hierarchy listing. Below it is the **Demonstrations** projects and then the L Machine save file of each **Project** or module appears in the right hand column. The *pop up menu* in the middle of the screen indicates various operations than can be performed on the **Memory System** project.



```
Project: CMOS
Save file: Z:>RZ>Portfolio>L-Machine>CMOS.lisp
Default directory: Z:>RZ>Portfolio>L-Machine>CMOS>
Environment: Demonstration CMOS

                                                      Project Hierarchy
 • Portfolio: Portfolio                      Z:>RZ>Schema-Init.Lisp
   Project: Demonstrations                   SCHEMA: DEMONSTRATIONS; DEMONSTRATIONS.LISP.NEWEST
 • Project: L Machine                        Z:>RZ>Portfolio>L-Machine.lisp
 •  Project: CMOS                            Z:>RZ>Portfolio>L-Machine>CMOS.lisp
 •   Module: The Chip                        Z:>RZ>Portfolio>L-Machine>CMOS>The-Chip.lisp
 •   Project: ALU                            Z:>RZ>Portfolio>L-Machine>CMOS>ALU.lisp
 •    Module: Inverter                       Z:>RZ>Portfolio>L-Machine>CMOS>ALU>Inverter.lisp
 •   Project: Control Logic                  Z:>RZ>Portfolio>L-Machine>CMOS>Control-Logic.lisp
 •   Project: Memory System                  Z:>RZ>Portfolio>L-Machine>CMOS>Memory-System.lisp
 •  Project: ECL                             Z:>RZ>Portfolio>L-Machine>ECL.lisp
 •  Project: TTL                             Z:>RZ>Portfolio>L-Machine>TTL.lisp
   Project: Standard Environments            SCHEMA: ENVIRONMENTS; STANDARD-ENVIRONMENTS
    Environment: ANALOG                      SCHEMA: ENVIRONMENTS; ANALOG-ENVIRONMENT.LISP
    Environment                                                              ONMENT.LISP
    Environment                                                              ONMENT.LISP
    Environment                                                              ONMENT.LISP
    Environment                                                              MENT.LISP

                          Operations on Memory System
             This Project            Create                  Save File
               Select             New project                  View
               Open               New module                   Edit
               Close          New Waveform Group              Reload
               Delete         Create Environment              Dump to
               Decache                                        Load all
               Describe                 X
            Edit parameters
        Exit                                                            Dump
              Unselect                          Unselect all                      Region unselect
Name for new sub project of CMOS: Memory System
NIL
NIL
NIL
NIL
NIL
```

Figure 11-1

2.3. Environments

By specifying an *environment*, designer makes precise what types of modules and tools should be available for the design. Each environment consists of a collection of primitive modules that may be used, command dispatch tables for the description editors, design rules, simulation models, and so on. The environments themselves are organized as a directed acyclic graph. At any time, the designer can refine the environment being used. For instance, one could begin a design in the *Basic Logic* environment and later when it had been decided to use CMOS, switch to the *Generic CMOS* environment. Finally, when a foundry had been chosen, the **Designer** would select an environment for the specific process to be used. While the environment was *Basic Logic*, the designer would be able to draw logic schematics and simulation, but would be unable to get any timing information (other than in gate delay units) or do any circuit design. After switching to *Generic CMOS,* transistor level circuits and stick diagrams could be developed. When the process specific environment has been chosen, detailed masks could be designed and accurate timing information would be available.

2.4. Software Tools

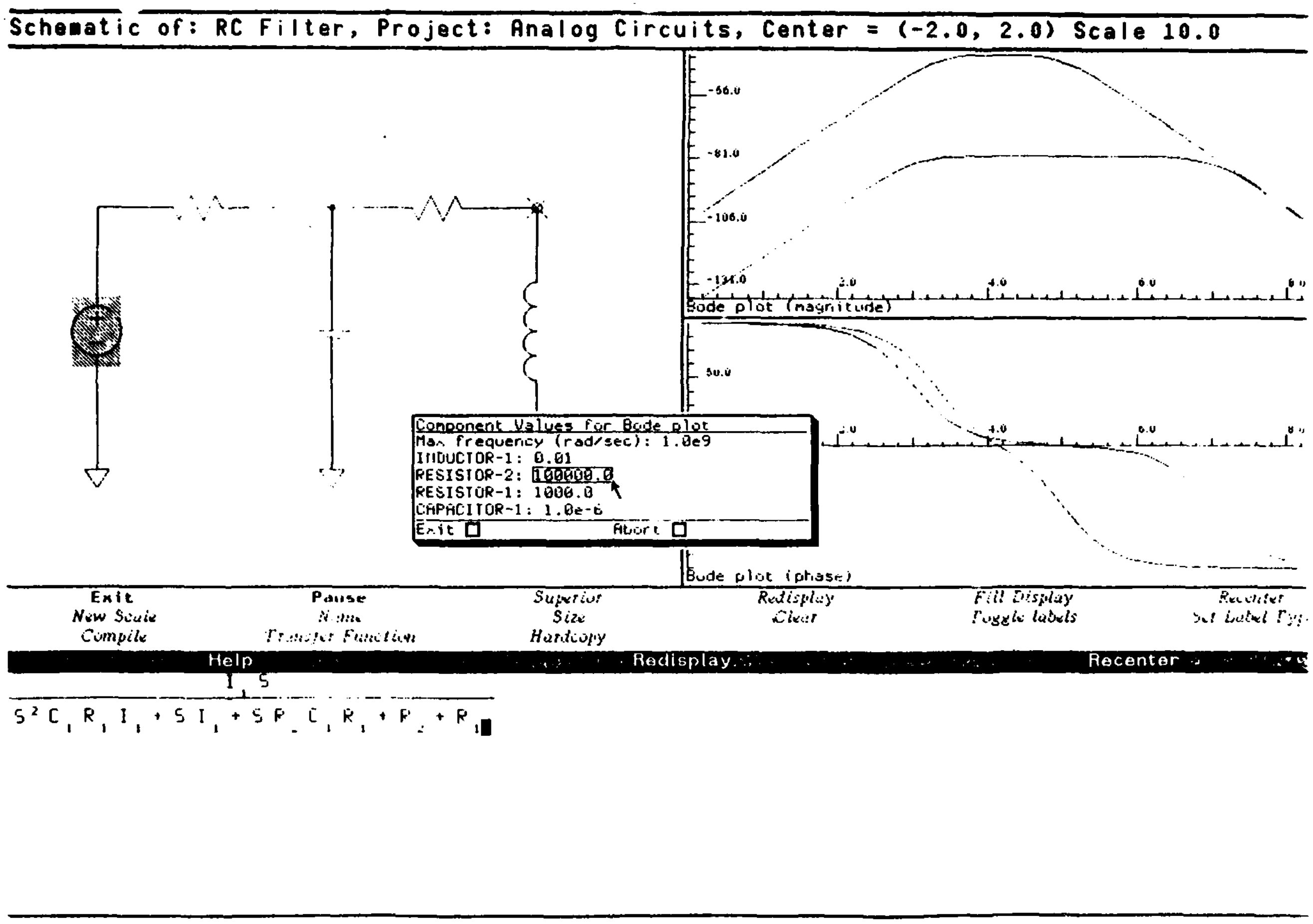
Through most of the time the data structures needed by a CAD designer are already in place, Schema also includes a large library of compatible flavors (abstract data types) for constructing new structures. Within this library are mechanisms for dealing with many different types of hierarchy, prototypes, "creation on demand", timestamping, and so on. When creating a new data structure, the designer merely picks the flavors that provide the functionality desired and includes his own customizations. This fine grained modularity has helped maintain a high level of uniformity within the system. The modularity techniques used are based on the Capsule ideas.

In addition, there is also a growing library of useful CAD oriented procedures that may be drawn upon. Among them are sparse matrix routines, linear and non-linear equation solvers, a moderate size symbolic algebra package, topological traversal routines, two dimensional spatial management packages, and so on. The existence of these packages has enabled CAD builders to build on each others' work more than in previous systems. The totality of these tools, mechanisms and policies remove much of the drudgery from CAD tool development and encourages tool developers to proceed in a cooperative, cumulative fashion. For the electronic designer, it provides a uniform environment, with uniform access to a wide variety of different synthesis and analysis tools.

2.5. Example

Figure 11-2 gives an example of the use of Schema to analyze a simple linear system. The schematic of an RLC system was sketched in the left hand window and then the user selected the item *Transfer Function* from the menu in the middle of the screen.

The user was then asked to indicate a driving voltage source and the load node, which are now indicated by a shaded region and the cross respectively. At the bottom of the screen is the exact voltage transfer function of the circuit. This calculation was performed by a simple application of Kirchoff's current law at each node of the circuit to construct a set of equations for the voltages, which is then solved. The polynomial manipulation package simplifies this problem significantly.



Schematic of: RC Filter, Project: Analog Circuits, Center = (-2.0, 2.0) Scale 10.0

Component Values for Bode plot
Max frequency (rad/sec): 1.0e9
INDUCTOR-1: 0.01
RESISTOR-2: 100000.0
RESISTOR-1: 1000.0
CAPACITOR-1: 1.0e-6
Exit ☐ Abort ☐

$$\frac{I_1 S}{S^2 C_1 R_1 I_1 + S I_1 + S R_2 C_1 R_1 + R_2 + R_1}$$

Figure 11-2

The user was then asked to see a Bode plot of the circuit. For this, it was necessary to assign values to the devices which has been done in the pop up menu. Two different sets of Bode plots (for different sets of parameters) are shown in the pair of windows at the right of the screen.

2.6. Milestones

Schema is currently being co-developed with G. Clark and M. MacDonald of Harris, Inc. They have spearheaded the development of the physical design portion of Schema and have made significant progress on the VLSI mask layout and PC board systems. At MIT, we have been concentrating on the overall structure of the system and the analysis tools. A fairly complete graphical waveform entry and editing system has been developed and is being interfaced with remote Spice servers running on Vaxes. This is also being primed to permit the RSIM and other simulators that run on the Lisp Machine to cooperate with Schema. Interface modules for dealing with a new standard interchange language EDIF are now being developed. We are anticipating our first serious users of the system early this fall.

3. DATABASE ACCELERATOR

The Database Accelerator is a new project being jointly pursued by Profs. Reif and Sodini of the Microelectronics Center and Prof. Zippel of LCS as part of the Smart Memories Project. The project is aimed at the development of a special chip that will speed up the database searches, pattern matching and solve various problems in artificial intelligence. The device will be fabricated in the Microelectronics Center under the supervision of Profs. Reif and Sodini.

There are three key ideas in the database accelerator architecture that distinguish it from conventional content addressable memories. First, the data words in the DBA can contain a *don't care* in any or all bit positions. Second, a generalized selector controls which words in the array can participate in an operation. Third, a small state machine is provided for each word for processing complicated sequences of matches. The combination of these functions yields an architecture that is quite extensible and can be used in a wide variety of applications.

A Database Accelerator consists of many *lines*. (MIT's Smart Memories Project is developing a 2048 line, database accelerator chip. Database accelerator systems with between 10^5 and 10^7 lines are feasible with current technology. Each of these lines consists of a 32 bit data word, a small finite state machine, a selector. and some additional select and decode logic. The DBA's operation is divided into four types of cycles. A *write* cycle is used to write data into the 32 bit data word of one or more lines in the DBA. During *match* cycles, a 32 bit *match word* is provided and compared against the contents of each data word in the system, setting or clearing a *match bit*. During *operate* cycles the match bit is combined with other bits to handle complex matching operations. The *readout* cycle provides the address of a particular matched word. Multiple readout cycles are needed when there is more than a single match.

Not all of the words of the CAM need to be involved in the operations. A *selector* word indicates which lines are to participate. The selector word may consist of zeroes, ones and don't cares. This allows a write operation to modify a single line, all of the lines, or certain subsets of the lines in the DBA.

The database accelerator is built from many DBA chips. The detailed design and fabrication of these devices is discussed in Wade's thesis. In this section we discuss some of the issues to establish the feasibility of the concept. Next we describe the memory cell and array structure. These circuits are responsible for the match cycle operation. The following section describes the finite state machine that is used during the operate cycles. And then we discuss the operation of the readout cycles.

3.1. Matching Capabilities

Each line of the DBA contains 32 content addressable memory cells. Each of these cells is capable of storing three values--two for logic one and zero and a third state denoted by X. We say that each cell contains a *trit* of information. In addition, these cells incorporate a comparator that is used in the matching process. Each cell is capable of comparing its contents with that of another trit on a 32 trit bus that is shared among all the cells. Each position in the bus is also capable of holding the three values: 0, 1 and X.

A word matches the quantity on the bus if, in each trit position, either the data word and the bus word have the same value or one of them is an X.

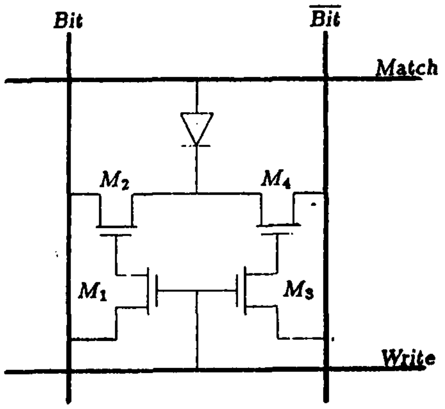


Figure 11-3: Lea-Wade CAM cell

The cell we use is shown in Figure 11-4. It is a derivative of the cell developed by Lea and modified by John Wade. It consists of four transistors that perform the bulk of the operations of the cell, and an MOS diode that is used to eliminate sneak path problems. Two data buses, *Bit* and *Bit*, run through the cell. These buses carry data to be written into the cells and is also used to present the data to be matched against. A word line is connected to the access devices M_1 and M_3,and is used to select the words to be written. Finally, a match line passes through the cell, AND-ing together the results of each cell's comparison.

As a data storage device, the transistor pairs M_1-M_2 and M_3-M_4 each form a conventional one transistor dynamic memory cell. Devices M_1 and M_3 are the access devices and M_2 and M_4 are the storage capacitators. On write cycles, a particular word is chosen, its wordline raised and the desired data is driven onto the *Bit* and *Bit* lines. The access devices then charge or discharge the gate capacitances of transistors M_2 and M_4 appropriately. Logic values one and zero are written driving the bit lines to opposite supply rails. An X value can be written into the cell by driving both bit lines low. The configuration with both gates charges is illegal since it shorts together two lines. Logically it would correspond to a never match in the cell. Indeed, it can be accommodated by adding an addition diode to the cell.

During match operations, transistors M_2 and M_4 form an equivalence gate that discharges the match line to th bit lines on mismatches. The diode D_1 is used to prevent sneak paths between cells of the same DBA line. Careful match sense amplifier design could eliminate the need for this device.

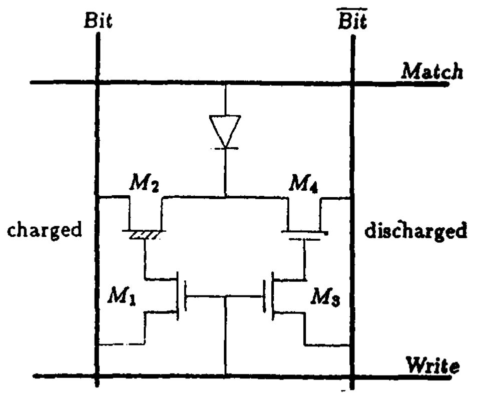


Figure 11-4: Operation of CAM cell during a match operation

In more detail, consider the case when a 1 is stored in the CAM cell as shown in Figure 11-5. If the bit lines also indicate a 1 the match line will not be discharged. The path through M_4 to the low bit line is of very high resistance since M_4 is turned off, and the low impedance path to a bitline, through M_2 is to a high bit line. Thus this cell will not discharge the match line. If a 0 were presented to the cell instead, then the match line would be discharged through device M_2.

In summary, the cell can contain a 0 and 1 by having only one of its storage transistors' gates charged at a time. It can contain a "don't care" (X) by discharging both gates.

3.2. Finite State Machine

The finite state machines are organized as simple one bit data paths that are connected in a linear, nearest neighbor fashion. Each date path contains a function generator capable of performing any single bit boolean operation and a set of registers. The function generator can take its input from e ther the (read only) match latch or a set of single bit registers associated with each DBA line. In the initial designs we are assuming four registers are associated with each line. In addition to these four, the function generator can access the registers of the previous and succeeding lines of the array. The result of the boolean operation can be written into any of the twelve registers that can be used as sources.

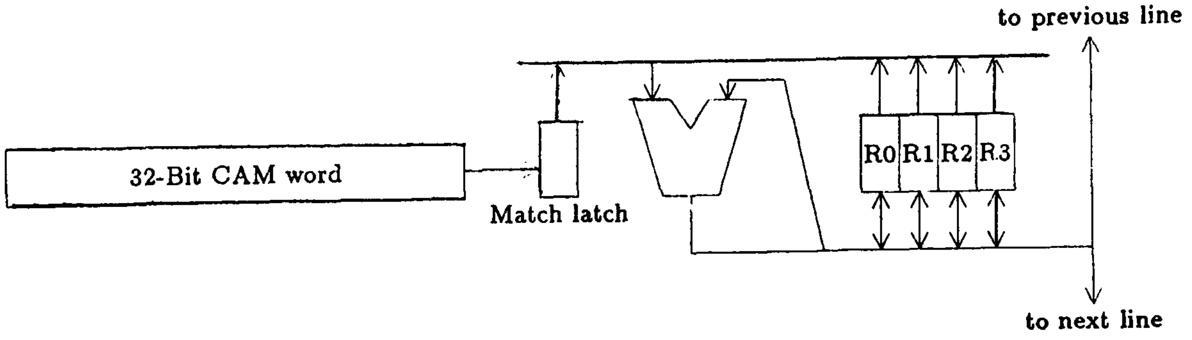


Figure 11-5: The DBA datapath

When focusing on a single line in the array, the registers associated with that line are denoted by 0, 1, 2 and 3. These are called *current registers*. The registers associated with the previous line are denoted by (**:previous***i*) and those associated with the next line by (**:next***i*). The function generator can be driven by the contents of any of the current, previous or next registers. In addition this source is also used as the destination of the function generator's result. The *B* source can be driven by either the match latch or the current registers. This is illustrated in Figure 11-6.

By allowing the *A* side of the ALU to access the registers of the previous and following word, arbitrary length match operations can be performed by using multiple cycles to match against the different parts of the data word. The function generator can then be used to *AND* together the results of these different matches.

It is not necessary or desirable for every line in the DBA to participate in every operation specified for the datapath. Which lines participate is controlled by the *selector field* of the operation. For a DBA system with 2_n lines, the selector field is an n-trit quantity that consists of zeroes, ones and don't cares. This word indicates which lines in the DBA are to participate in the operation. If the selector consists solely of don't cares (probably the most common situation) then every line of the DBA is enabled. To perform a 64-bit match, one match operation is performed using a selector of XXXXXXXXXXX0 followed by an operation with selector set to XXXXXXXXXXX1.

The results of these two operations are then ANDed together to determine the result of a 64-bit match. Other patterns can be used for more devious calculations.

As mentioned, the output of the function generator is written into the register specified by the *A* source. In addition, two global bits are set by *AND*ing and *OR*ing together the selected function generators' results. The *AllMatches* line is driven by the *AND* of the outputs of all the selected function generators. The *NoMatches* line is driven by the *OR* of the outputs of the selected function generators. Among other uses, these signals can be used to implement arithmetic comparisons.

3.3. Read Out

The read out cycle is quite simple. As with all other operations, the selector field controls which DBA lines participate in the operation. In most cases the selector field will consist of all don't cares. The read out instruction indicates which of the four register bits are to be examined. If only one DBA line contains a 1 in that register then its address is presented, encoded in binary, to the pins. The register is then cleared. If more than one register contains a 1 then the DBA line with the lowest address is selected, its register cleared and address presented to the pins.

3.4. Milestones

To evaluate the viability of the database accelerator a simulator has been built that emulates the operation of the DBA chips. This simulator allows us to test algorithms that make use of DBA chips and explore the impact they will have on different applications. The simulation is done in enough detail that we can examine different ways of managing the DBA memory, and the impact of chip organizations on the performance of the overall system. We expect to be using SCHEMA to design the initial full DBA chips later this calendar year.

4. VLSI SIMULATION

The burgeoning interest in integrated circuit design has led to an increased demand for timely, accurate circuit simulation. Due to capacity limitations of existing simulation programs, this demand is increasingly difficult to satisfy, especially as circuit sizes now routinely exceed the 100,000 transistor mark. Algorithmic improvements--*e.g.*, relaxation-based circuit analysis and switch-level transistor models--have helped to close the gap, but even these simulators fall short of providing the capacity that will be required in the near future.

Digital circuit operation exhibits a high degree of locality, with each element of the circuit operating only upon information local to that element. Terman and Arnold have developed a framework for circuit simulation which takes advantage of the locality inherent in circuit operation to achieve a high degree of parallelism. The strategy is to map the circuit to be simulated onto the topology of the target multiprocessor such that the parallelism of the simulation reflects the parallelism of the circuit. For simulation on an *n* processor system, the circuit to be simulated is first broken into *n* subcircuits, or *partitions*. Each partition is composed of one or more *atomic units, e.g.*, gates or subnets, where an atomic unit is the collection of local network information necessary for the simulation algorithm to determine the value of a circuit node. Each processor is then assigned the task of simulating one partition of the circuit.

To minimize communication and to guarantee a consistent view of the state of the network across all processors, we enforce the restriction that the value of every node is determined by exactly one partition. A node may be either an input or an output of a partition, but never both. If more than one partition were allowed to drive a particular node, each partition would require information about the state of the other drivers to determine the correct value of the node. By eliminating the possibility of multiple drivers we eliminate the need for this non-local information and the extra communication required to arbitrate such an agreement.

This is not as serious a restriction as it first appears. In an MOS circuit, it implies all nodes connected through sources or drains of transistors, such as pullup and pulldown chains and pass transistor logic, must reside in the same partition. Since such structures are the components of higher level logic gates, it makes sense to keep them close together. The only difficulty arises from long buses with many drivers. This case results in a "bit slice" style of partitioning, where all of the drivers for one bit of the bus reside in the same partition, but different bits may reside in separate partitions. Since there tends to be relatively little communication from one bit to another, this restriction actually obeys the natural decomposition of digital circuits.

Thus, the partitioning strategy seems to be a good one for several reasons:

- Circuit locality is unaffected by scale. The potential parallelism increases linearly with the size of the circuit to be simulated.

- Since the simulation computation displays locality of reference, the partitioning will not result in a large interpartition communication overhead. In fact, most of the knowledge about the network can be distributed along with the simulation.

- Most circuits have natural boundaries which can be used during the partitioning process since successful designs must constrain communication between submodules to meet routing and bandwidth requirements imposed by the technology.

Ideally each partition could be simulated independently, keeping each of our *n* processors busy doing useful calculations. We may fall short of this ideal for several reasons:

1) Some partitions may be easier to simulate than others; the resulting lack of balance in computation load may lead to some processors lying idle.

2) A node whose value is determined by one partition may be an input to one or more "successor" partitions. This introduces a precedence constraint among the partitions: the simulations of the successor partitions must always lag behind the simulation of the original partition in order to ensure that the correct value of the input is available.

The first problem can be ameliorated by improvements in the original circuit partition; this is an area of active research and will not be pursued further here. The second problem is addressed below.

Enforcing the precedence constraint represented by a node shared between two (or more) partitions requires additional communication and can introduce delay in a poorly balanced simulation. Feedback between two partitions creates a circular precedence constraint, which can result in a forced synchronization of the simulation processes: each partition in the feedback path cannot proceed with the simulation of time step $t + 1$ until all partitions have completed step t. However, if the value of all partition inputs were known for all time, there would be no precedence constraints to enforce and each partition could be simulated independently of the others. This suggests decoupling the partitions by introducing a *history buffer* for each partition which encodes the value of each input for all time. Simulation of a partition is then allowed to proceed based upon the assumption that the current contents of its history buffer is correct.

When a partition changes the value of an output node, it sends a message to the other partitions for which that node is an input. The receiving partitions use this information to update their history buffers, and, if necessary, correct their simulations. In order to correct a simulation to account for a change in the input history, we employ a checkpointing and roll back strategy derived from the state restoration approach to fault tolerance in distributed systems. As the simulation progresses, a partition periodically stops what it is doing and takes a *checkpoint* of the current state of the simulation. The checkpoint contains a record of all of the pieces of state in the partition: the value of every node, all pending events, and any state information kept by the simulation algorithm (*e.g.*, the current simulated time). From this checkpoint, the simulation of the partition can be completely restored to the saved state at any future time, effectively rolling the simulation back to the time the checkpoint was taken. The set of saved checkpoints forms a complete history of the simulation path from the last system-wide resynchronization up to the current time.

When a partition receives an input change, one of two possible actions will occur. If the simulated time of the input change is greater than the current time, a new event representing the change is scheduled and simulation proceeds normally. However, if the simulated time of the input change is less than the current time, the simulation is "rolled back" to a point preceding the input change. This roll back operation is accomplished by looking back through the checkpoint history to find the most recent checkpoint taken prior to the scheduled time of the input change. The simulation state is then restored from that checkpoint, a new event is scheduled for the input change, and simulation is resumed from the new simulated time.

With the scheme outlined so far, it is still possible for two partitions with a circular dependence to synchronize. If both partitions change shared nodes at the same simulated time, each partition will force the other to roll back. To solve this problem, we first make the following assertion about the nature of the simulation algorithm: *the elapsed simulated time between an input change and any resulting new events is non-zero.* This assertion can be guaranteed by proper partitioning of the network. This restriction allows the simulation of a single time step to be sub-divided into two distinct phases:

1) the processing of all internally generated events queued for the current simulated time, including the propagation of output changes to other partitions;

2) the processing of all externally generated input changes queued for the current simulated time.

This in turn permits us to take a checkpoint between the two phases of the simulation, after any output changes have been made and before any input changes have been processed. Forward progress is assured if we can guarantee there will always be a checkpoint at the right time.

The checkpointing strategy must meet the following constraints: the checkpoint must contain all of the state necessary to completely restore the simulation; there must always be at least one consistent state to fall back to; and it must be possible to make forward progress in the event of unexpected synchronization. In addition to these constraints, there are some less important but still desirable properties a checkpoint strategy should have. For example, to prevent rolling back further than necessary, the simulation should be checkpointed frequently. In the limit, a checkpoint at every time step would eliminate redundant work. We would also like the checkpointing process to be as inexpensive in both space and time as possible. There is a tradeoff between the cost we are willing to pay when forced to roll back and the cost we are willing to pay for checkpointing overhead.

We expect the communication between partitions in a statically well-partitioned circuit to be clustered in time, *e.g.*, around clock edges. This implies the probability of receiving a node change is greatest immediately following a change, and decreases as the time since the last change increases. The probability of roll back should follow a similar pattern. Therefore, to reduce the amount of redundant simulation caused by rolling back, we would like to have a high density of checkpoints in the vicinity of communication clusters. If the dynamic balance of the partitioning is less than ideal, some of the partitions will simulate faster than others. In this case, the amount of redundant work forced upon the faster partitions by roll back is less critical, as they will still catch up to and overtake the slower partitions. Hence, if the time since the last roll back is large, we can afford to reduce the density of checkpoints.

These observations have led to a strategy of varying the frequency of checkpointing with time. Following each resynchronization and each roll back, a checkpoint is taken at every time step for the first several steps, thus ensuring forward progress as well as providing a high density of checkpoints. As the simulation progresses, the number of time steps between checkpoints is increased up to some maximum period. The longer the simulation runs without rolling back, the lower the checkpoint density, and hence the overhead, becomes. We have arbitrarily chosen to use an exponential decay function for the frequency until we have a better model of the probability distributions of interpartition communication.

In the last year, a prototype implementation of PRSIM was completed an preliminary performance measurements were made. (PRSIM is a parallel circuit simulator which employs the history and roll back mechanisms presented above; as the name implies, PRSIM is based upon the RSIM algorithm. The initial implementation was designed for the Concert multiprocessor, developed here at MIT.)

Performance measurements are most easily expressed in terms of the *effective speedup*; for N processors this is defined to be $t(N)/t(1)$ where $t(N)$ is the time taken to run a given experiment on N processors. The extra simulation incurred as a result of roll back can be expressed in terms of the *simulation efficiency*, which is defined to be $_s$ = events (1)/events (N) where events (N) is the number of events processed in an N partition experiment. The following table shows measurements taken while simulating a 64-bit adder circuit containing 2688 transistors and 1540 nodes:

N	η_s	Speedup
1	1.000	1.00
2	0.991	1.86
3	0.967	2.43
4	0.937	3.11
6	0.951	3.77

Arnold's thesis contains an expanded presentation of measurements taken from a variety of experiments.

5. PARALLEL PROCESSING

Work on parallel processing has continued on several fronts. We have continued the construction of the Concert multiprocessor hardware, along with investigation of the performance characteristics of Concert-like architectures. The base of utility software for Concert has continued to expand. The Multilisp parallel programming language has been used for some small-scale application studies and has also been the subject of continued development. During the past year, outside organizations have collaborated in several aspects of this work.

5.1. The Concert Multiprocessor Testbed

During the past year, three RingBus Interface Boards (RIBs) were manufactured and successfully connected together creating the first Concert system with more than one Multibus. This represented a major milestone on the way toward building a full eight-Multibus, 32-processor Concert system. As of this writing, the remaining five RIBs have been built and await check-out. All remaining parts needed to populate a full Concert machine have been ordered, and commissioning of the full-scale Concert multiprocessor should be possible by the end of the summer.

Analytical models of Multibus and RingBus contention have been developed and compared with simulation results. The models of Multibus performance have also been tested experimentally. Among the conclusions of this work is that Multibus contention is not likely to affect performance as long as programs are located in the local memory of processors that execute them, but that RingBus contention can be significant in many cases.

The software environment for development of programs to run on Concert has continued to jell, with significant contributions by our industrial partners at Harris Corp. The ROM bootstrap code and the library of subroutines that support programming Concert in C have been reorganized to anticipate the full Concert with many Multibuses, and to accommodate system-wide "servers" connecting to shared resources. A file system server is now complete and working reliably. A network server is under development, and a "terminal server" providing windows for the output of different processes is planned.

5.2. Multilisp

The principal advance in the Multilisp system during the past year has been the design and implementation of a debugger and exception handling system. A task that encounters a problem can yield an "exception value", or error value, as its result, instead of a normal value. If another task touches the exception value, it too may be aborted and yield an exception value as its result. This mechanism provides a solution to the unique problems of exception propagation between tasks using the "futures" mechanism of Multilisp. The implemented system also includes an interactive debugger written in Multilisp.

Application programs written in Multilisp include several sorting procedures, an interpreter for a logic programming language, (Solomon), a polynomial manipulation package, and a gate-level digital logic simulator. Additionally, a partial implementation of the QLAMBDA parallel Lisp programming language was constructed by building a translator from QLAMBDA to Multilisp. Strategies were developed for automatically inserting futures at suitable places in Pure Lisp programs. Several tools have been built for gathering execution statistics of Multilisp programs.

Although many rough edges remain, a basic Multilisp system now exists that is complete enough to experiment with reasonable application programs, and also to serve as a base for further research into the systems aspects of parallel languages. Accordingly, we have begun to consider not only the question of debugging and exception handling, but also broadening the range of parallel algorithms that can be expressed comfortably in Multilisp. There appear to be two sources of parallelism in programs:

- Mandatory work - Often the precedence constraints among operations in a program can be relaxed so that several operations can be performed in parallel, and the results combined later.

- Speculative work - Additional work can be initiated, in an "eager" fashion, on the assumption that it may prove relevant. Speculative tasks may of course turn out to be irrelevant, in which case it is desirable to terminate them.

The "futures" mechanism of Multilisp is useful in relaxing the precedence constraints among the mandatory operations of a program, but Multilisp lacks some vital ingredients for good support of speculative parallelism. In the presence of speculative work, it is desirable for the system to understand which tasks are speculative and which are mandatory, so that speculative tasks will not be executed to the exclusion of mandatory ones. Some method is also needed for locating and cleanly terminating speculative tasks that have become irrelevant.

Another problem with the current Multilisp implementation on Concert is that the introduction of parallelism into programs involves too much overhead for small-to-medium-grain use. A program heavily laden with futures may take four times as much processor time to execute as the same program without futures. A "lazy" implementation of futures that postpones or eliminates most task creation operations may substantially reduce the overhead of using futures.

The coming year should see the development of facilities in Multilisp for speculative parallelism, as well as the refinement of the debugging and exception handling mechanism. Other system issues, such as input and output, also need to be addressed at the language design level. At the implementation level, we expect to investigate ideas such as lazy futures, as well as improved multiprocessor garbage collection algorithms.

5.3. Industrial Collaboration

The second full year of our collaboration with Harris Corp. has just been completed. Highlights of their work are

- The building of the GCM, a crossbar-based replacement for the RingBus. The GCM should exhibit higher performance and is now entering its debugging phase. Ultimately, Harris expects to make a GCM available for the M.I.T. version of Concert.

- The design of some performance measurement hardware for Concert. When built, this hardware will also be made available to M.I.T.

- Development and improvement of several pieces of Concert utility software, notably the file server and the ROM bootstrap monitor.

- Submission of a joint Harris/M.I.T. proposal to the DARPA strategic Computing Program to build a second-generation multiprocessor called SCAMP. Funding of this proposal now seems very likely.

The Multilisp effort has also engendered two other industrial collaborations:

- A group at Bolt, Beranek, and Newman has brought Multilisp up on BBN's Butterfly multiprocessor. Although there are many areas where some tuning would clearly lead to major performance improvements, Multilisp has run successfully on Butterfly machines with as many as 128 processors.

- A group at MCC Corp. has used our Vax simulator for Multilisp to experiment with introducing parallelism into the rule-based system EMYCIN.

6. THE L COMPUTER ARCHITECTURE

The goal of this new research by Terman and Ward is the development of an effective architectural basis for high-performance implementations of modern high-level software. In contrast to conventional computer organizations, L is designed around assumptions of (i) object-oriented storage and (ii) multiple parallel threads of control at its lowest architectural levels. These assumptions allow direct hardware support for a computational model which offers unique advantages both in the organization of multiprocessor machines and in the implementation of advanced control and data structures becoming popular in contemporary software.

The following paragraphs outline the current -- tentative -- model of computation which we assume for L. It should be emphasized that the description is intended to illuminate the architecture of L's low-level computational engine, rather than to describe a programming model. Much of the mechanism referred to below is invisible to the programmer, who typically deals with higher-level program and data constructs which resolve, largely via compiler technology, into the primitives sketched here.

6.1. Chunks

The unit of storage in L is the *chunk*, a fixed-size block capable of holding a modest number (*viz.* nine) of values; each such value may be scalar data or a chunk reference. A single tag or *reference* bit associated with each value distinguishes chunk references from scalar data. Although other runtime tags may be among the chunk data (and we envision instruction-set support for doing so efficiently), the details of such typing are currently left to compiler-enforced convention. The reference bit is the only run-time type information whose integrity is guaranteed by the lowest-level execution model

One of the nine elements of each chunk is reserved to denote the *type* of the object represented. Scalar type values are used to flag certain primitive chunk types (such as those interpreted directly by the processor); higher-level types, such as user-defined type extensions, may exploit references to arbitrary objects (*e.g.*) to type templates for data structures). Each chunk may thus be viewed as a typed eight-tuple, whose elements may themselves by chunks. All higher-level data and code objects are represented using chunks; i.e, lists and strings as a linear chain of inter-referencing chunks, or arrays as trees of branching factor 8.

Each chunk is named by a unique ID which is used when interacting with the memory system. Unlike conventional Von Neuman architectures, a chunk ID in the L architecture is not directly related to the current location of the chunk in memory. Instead, the ID is used as a key to retrieve the desired data from a content-addressable memory. This gives us the freedom to move the actual storage for a chunk to different locations in the memory system; we can use this flexibility to cache a chunk close to the processor(s) which reference it most frequently.

6.2. Computation State

The basic L processor, like conventional processors, may be viewed as a finite state machine. Unlike conventional computers, however, the state of an L processor has been engineered to fit neatly into a single chunk. As an immediate consequence, a computation state may be encoded as a first-class data object, occupying a chunk. To this end, one of the primitive chunk types in L is a *computation state*, or simply STATE, which is the semantic equivalent of a full continuation.

The programming model supported by L plays heavily on the elevation of the computation state to the status of a formal data type. Indeed, we view the L processor state as external to the processor itself: the processor simply takes an input state, S, and transforms it to a successor state S'.

Each STATE chunk has a scalar type identifying it as a computation state; the processor will refuse chunks not so marked. The first value of each STATE chunk is a scalar *microstate* word, containing *prerequisite* and *code offset* bit fields, to be explained presently, and several miscellaneous additional status bits. The next value of a STATE chunk, the *code pointer*, contains a reference to a CODE chunk, containing a stream of pure instructions (possibly intermixed with pure data). The major function performed by the L processor is the interpretation of consecutive instructions from each CODE chunk.

The remaining six values of a STATE chunk are, so far as the low-level computation model is concerned, unallocated. They play roughly the role of general registers in a very simple conventional machine: compiler-enforced conventions will allocate them to such functions as return linkage, environment pointers, static links, and temporaries.

6.3. Instruction Stream Coding

Each CODE chunk contains a modest number of simple instructions, which may contain scalars and chunk references as (pure) immediate constants. Among the latter are references to other CODE chunks, *e.g.* as operands of branches and simple calls. Typical CODE chunks contain at least one such branch, identifying a successor chunk in the instruction stream. As each CODE chunk typically contains several L instructions, the *code offset* bits within the *microstate* value of a STATE chunk are used in conjunction with the *code pointer* to identify the next instruction to be executed.

Most L instructions are conventional 1-, 2-, and 3-address formats. Each operand must be either a value within the current STATE chunk, or a value in some chunk *referenced* by the current STATE. The L operand encoding thus provides for direct access to data whose distance, in levels of indirection, is at most one from the current state; access to more distant data requires multiple instructions. Additional instructions provide for the allocation of new chunks, the activation of new computation states, and for interaction with the external world.

Each L instruction may require, for its complete execution, access to one or several of the chunks referenced by the current computation state. In order to (i) provide for primitive program synchronization, and (ii) avoid unnecessary bottlenecks in critical execution resources, each STATE contains *prerequisite* bits in its *microstate* value which specify which referenced chunks must be locally accessible for the computation to proceed. The prerequisite bits are managed explicitly by the program, and can specify one of several kinds of access--*e.g.* shared (*read-only*), or exclusive (*read-write*)--for each reference within the STATE chunk. The prerequisite bits afford us the guarantee that no computation step will be attempted unless its prerequisites are satisfied and hence the computation can proceed to the next step without delay. Moreover, since several computation states which specify exclusive access to the same object cannot proceed simultaneously, prerequisites provide rudiments from which higher-level synchronization mechanism can be crafted.

6.4. Continuations and Multiple Threads

The abstract computational model engendered by the structure described thus far consists of a dynamic, evolving network of inter-referencing chunks, some of which are identified as type STATE. At each step of the computation, some set of runnable STATE chunks with non-conflicting prerequisites is selected to be advanced to successor states. The STATE chunks so selected can then be advanced to their successor states in any order, or simultaneously, since the respective computation steps are non-interacting. The advancement of any state may lead to multiple successor states--effectively a *fork* operation--to be spliced into the network in the vicinity of the parent. The capacity of this scenario for unbounded numbers of threads of control, together with the explicit coding of criteria for each to progress, reflects our aspiration to highly parallel L implementations.

Nondeterminism enters the model in that the criteria for selection of computation states to be advanced during a given step are deliberately underspecified. In a practical multiprocessor L system, that selection would be constrained by limited resources (*e.g.* processors) as well as by technology-dependent communication costs.

6.5. The L Engine

An L machine contains some number of L *engines* interconnected, along with external memory and devices, in a communications network of some kind. Each engine contains a processor--the stateless FSM described earlier--coupled to some highly accessible local memory used to cache a number of chunks. Very roughly, we envision each engine to be consistent, in terms of complexity, with implementation as a single VLSI chip.

The local cache of an engine typically includes both STATE and data chunks. Dedicated hardware recognizes runnable STATE chunks, and queues them for advancement by the processor. The runnability criteria include verification that the specified prerequisite chunks are accessible in the local cache, and (perhaps) buffering in pipeline registers decoded address information needed to access them in a single clockcycle. The goal is to maximize the rate of useful execution by the processor, by (i) keeping its input queue full and (ii) reducing the cost of moving from the current STATE to the next in the pipe virtually to zero.

An independent subsystem manages communication between the cache and the external system. System-wide communication follows a split-transaction protocol, consuming constant communication resources despite unbounded access times. A STATE whose prerequisites are unsatisfied either waits in the cache while the needed chunks are fetched, or is shipped to another engine (presumably one closer to the missing resource). While we have plausible algorithms and speculative predictions regarding heuristics for such decisions, we expect to perform substantial simulations before these aspects of our implementation will gel.

6.6. L Status and Plans

L is currently an incomplete collection of novel but untried ideas. While we aspire to their eventual implementation using state-of-the-art technology (e.g., in a multiprocessor system using custom VLSI processors) we expect to spend the next one to two years exploring and refining architectural parameters using emulation and other "disposable" implementation technologies.

We plan on an initial implementation using NuBus-based T.I. Explorers, re-microcoded to emulate the L architecture. We hope to achieve performance approaching that of conventional single-processor Lisp Machines, allowing us to evaluation architectural alternatives using software benchmarks of realistic size and functionality. Indeed, we look forward to an early L implementation whose functionality and performance are adequate to serve as a tool for the L research itself: compilers and other system software will be coded for an L processor. Among other advantages, this approach (1) provides an opportunity to evaluate the semantic advantages of the programming model proffered by L, and (2) accumulates a substantial body of interesting benchmark software.

Subsequent prototype implementations are expected to address the multiprocessor aspects of an L system, and will exploit the Laboratory's Multiprocessor Emulation Facility. We expect the MEF implementation to draw heavily on earlier single-processor prototypes, owning to their common basis on the TI processor. It is possible that during this period, specific architectural features (such as metacaches or unique memory subsystems) will be prototyped in hardware.

7. PERSONAL WORKSTATIONS

In a similarly-named section of last year's progress report, we related the delivery by TI of 30 production Nu Machines per their agreement with us. While these machines have been deployed within the Laboratory -- primarily running Trix within the RTS Group -- TI has recently discontinued their manufacture and dissolved the Irvine Group responsible for that product line. The funeral date passed with few tears shed, although 30 tombstones were left and may become a disposal item.

7.1. Trix

During the past year, Goddeau and Sieber brought the Nu implementation of our Trix operating system to acceptable levels of performance and functionality. The current Trix runs major Unix software packages, yet offers a level of network-wide coherence unattainable using Unix: the network itself, including the distinction between local and remote services, can be rendered entirely transparent by Trix.

Despite its technical advantages, modest efforts for direct transfer of Trix to industry have been unsuccessful since the emergence of Unix as an interface standard.

Consequently, the Trix project is winding down; current Trix-related activity is primarily publication of results rather than additional system development. We expect that the ideas and technology pioneered in Trix will be influential in the development of the LCS Common System and similar projects aimed at network-wide coherence.

7.2. The NuBus

The architectural basis of the Nu Machine -- the NuBus -- remains alive and well within TI and elsewhere. The recently announced TI Explorer is NuBus-based, and we expect additional products from TI and other manufacturers to add to the repertoire. TI has, with our encouragement and participation, continued to promote the NuBus as an industry standard. In addition to licensing the bus at a very nominal one-time fee (shared with MIT), TI has devoted considerable energy -- along with some of our own -- toward the IEEE ratification of the NuBus as a standard. Barring unforeseen setbacks (e.g., the departure of key individuals from TI) we expect the NuBus to have an IEEE designation by next year's progress report.

The NuBus continues as an attractive basis for experimental multiprocessors at MIT and elsewhere -- MEF and LCS and the SPUR project at Berkeley being examples.

THEORY OF COMPUTATION

Academic Staff

P. Elias	A. Meyer
S. Goldwasser	S. Micali
F.T. Leighton	G. Miller
C. Leiserson	R. Rivest, Group Leader
N. Lynch	M. Sipser

Associates

F. Feldman	L. Levin
P. Gacs	W. Przytula
S. Homer	S. Schwarz
E. Lander	M. Wand
D. Lehmann	

Graduate Students

B. Aiello	D. Barrington	B. Berger
R. Boppana	V. Breazu	T. Bui
J. Buss	B. Chor	T. Cormen
S. Cosmadakis	P. Feldman	D. Franzblau
B. Gasarch	R. Greenberg	J. Hastaad
R. Hirschfeld	D. Jilk	B. Kaliski
P. Klein	M. Komachak	D. Kornhauser
F.M. Maley	S. Malitz	M. Newman
M. O'Connor	C. Phillips	S. Rao
M. Reinhold	A. Sherman	P. Shor
S. Sur	S. Trilling	S.-M. Wu
	L. Yedwab	

Undergraduate Students

A. Ishii	B. Maggs

Support Staff

A. Benford	I. Radzihovsky
J. McNamara	R. Spenser

L. Quinby	K. Story

Visitors

A. Amir	P. Kanellakis
B. Awerbuch	M. Merritt
L. Berman	J. Reif
C. Dwork	

1. OVERVIEW

This section contains an alphabetized listing of faculty and research endeavors, categorized by personal account. Some of the material for this preliminary group report has been incorporated from the faculty personal statements.

2. INDIVIDUAL PROGRESS REPORTS

2.1. Ravi Boppana

This semester I did research in Boolean formulas and Expander graphs. In the area of Boolean formulas, L. Valiant has a general method for converting probabilistic formulas into deterministic ones without increasing the size by much. I showed that in fact, Valiant's method is optimal in a certain sense, namely the blowup in size is best possible.

In the area of Expander graphs, I have been trying to show that a random regular graph has good eigenvalue properties. If this is true, it would lead to a good method for generating graphs, and then efficiently checking that they are indeed expanders.

This semester I completed my second course of my minor. I am intending to submit a Ph.D. thesis proposal, "Topics in Boolean Formulas," sometime this summer. This Fall I will take my Area Exam.

2.2. Thang Bui

T. Bui completed his joint work with Profs. Tom Leighton and Mike Sipser on the problem of devising a new graph bisection algorithm and analyzing its performance, both analytically and experimentally. The result of this work is a new bisection algorithm along with proofs that this algorithm works well for large, natural classes of graphs. In particular, it is proved that if the algorithm returns a bisection then it is guaranteed to be an optimal bisection. Furthermore, it is shown that for every fixed (d>=3) and for almost all d-regular graphs on 2n vertices with bisection width b = O(n**(1- (1/((d+1)/2)))) the algorithm finds the optimal bisection. The algorithm has been implemented and tested, and found to be competitive or better than well known bisection algorithms, such as Kernighan-Lin or simulated annealing.

2.3. Peter Elias

On-Line Adaptive Source Coding Algorithms. I am writing up two simple on-line adaptive variable-length source coding schemes. In each scheme the encoder encodes the next message selected from a message set M into a codeword selected from a prefix-free set C of sequences, concatenating the result onto previous output. The decoder receives the concatenation and decodes each message as soon as the last symbol of its codeword is received. Either scheme can be used by encoders and decoders who know the sets C and M and have agreed to an indexing of C in an order of increasing length and of M in some standard order, but know nothing about probabilities of messages. The schemes encode with an average codeword length not too much greater than that of Huffman encoding, an optimal on-line nonadaptive scheme which requires exact advance knowledge of message probabilities.

2.4. Ronald Greenberg

C. Leiserson and R. Greenberg have further improved their algorithm for on-line routing of messages in the "fat-tree" interconnection network. This probabilistic algorithm is novel in that it does not randomize in the choice of message paths or in the operation of the switches, .t rather in the choice of whether or not to send a particular message in a particular delivery cycle. The algorithm ensures that a set of messages, M, can be routed with high probability within O(lambda(M)log|M|) delivery cycles, where lambda(M) is the maximum over all communication links of the ratio of the number of messages in M which must pass through the link to the capacity of the link. It is hoped that this work may also have some applicability to routing networks other than "fat-trees."

2.5. F. Thomas Leighton

Professor Leighton is continuing research on several problems involving novel network architectures, parallel computation, VLSI design and the development of algorithms for NP-complete problems which provably work well on the average. Advances have been made in several areas during the past six months. Highlights are described in the following paragraphs.

In the algorithms area, Professors Leighton and Sipser, T. Bui and S. Chaudhuri (University of Washington at Seattle) have developed graph bisection algorithms which (provably) almost always find the minimum bisection of graphs with small bisections. These algorithms perform dramatically better than known techniques for large classes of relevant graphs. The work will form an important part of T. Bui's Ph.D thesis, which should be completed by this summer.

In the area of fault-tolerant construction of VLSI networks, Professors Leighton and Rosenberg (Duke University) and Dr. Chung (Bell Communications Research Labs) have developed efficient algorithms and bounds for representing useful networks as a small number of "stacks" of wires. As the stacks are easily implemented in VLSI, the results make possible the efficient configuration of fault-free networks in environments that contain defective components.

In related work, Professor Leighton and P. Shor (who is also expected to finish his Ph.D thesis this summer) are very close to solving the grid matching problem. Roughly stated, the problem is to determine the expected minimum maximum edge length over all perfect matchings of N random points to N fixed points that are arranged in a | N * | N grid with unit spacing between consecutive rows and columns. Professors Leighton and Leiserson proved an upper bound of O(log N) and a lower bound of Omega | log N for this problem in their work on wafer-scale integration of systolic arrays in 1982. Determination of the precise bound has remained a difficult and important open problem ever since. It now appears that the exact bound for the grid matching problem is theta(log**(3/4)N), improving both the upper and lower bounds. As a direct result of this work, it will be possible to improve the best bounds known for the average case behavior of algorithms for wafer-scale integration as well as for a variety of other packing and assignment problems.

Professor Leighton and J. Hastad (a first year graduate student) are developing efficient circuits for parallel division. Currently, the only known circuit that can compute the N most significant bits of a quotient in O(log N) parallel steps require Theta(N**5) processors. Preliminary work by Leighton and Hastad indicates that the number of processors can be decreased to O(N**(1+epsilon)) where epsilon is an arbitrarily small positive constant. Although not yet practical, the improvement in hardware requirements is significant.

Professor Leighton and B. Berger (another first year graduate student) are developing improved algorithms for 2-layer channel routing. Initial progress in this area suggests that it may be possible to achieve the performance of the Baker-Bhatt-Leighton Manhattan routing algorithm and the Rivest-Baratz-Miller knock-knee routing algorithm with a single, simpler algorithm. More importantly, it appears that the new algorithm can be extended to the unit-vertical-overlap model (in which wires can overlap only for unit distance and only in the vertical direction) where a factor of two in channel width can be saved. The factor of two is significant because the new algorithm always routes 2-point net channels with width d+O(d) instead of the best previously known bound of 2d-1. Here d denotes the density of the channel which, of course, is a lower bound on channel width. The results also hold for multipoint net problems, except that an additional factor of two in channel width is required.

2.6. Charles E. Leiserson

My research work continues to center on "fat-trees." My students and I have refined the organization of fat-trees, discovered new efficient message routing techniques, and broken ground in the design and analysis of parallel algorithms for fat-trees. I am also continuing work on VLSI in general. Specifically, we are making progress in understanding timing in synchronous circuits clocked by level-sensitive latches, we have new algorithms for planar routing, routability testing, and compaction, we have discovered computational geometry algorithms for computer-aided design that work well in systems with limited primary memory, and we are studying VLSI designs for switching bit-serial messages and have implemented two such designs.

C. Leiserson and R. Greenberg have discovered a good algorithm for on-line routing of messages in the "fat-tree" interconnection network. This probabilistic algorithm is novel in that it does not randomize in the choice of message paths or in the operation of the switches, but rather in the choice of whether or not to send a particular message in a particular delivery cycle. The algorithm ensures that a set M of messages can be efficiently routed on an n-processor fat-tree. With high probability, the number of delivery cycles required is O(lambda(M)+\lg n\lg\lg n), where lambda(M) is the maximum over all communication links of the ratio of the number of messages in M which must pass through the link to the capacity of the link.

B. Maggs and I have discovered an algorithm for computing a minimum spanning forest of a graph which runs on a fat-tree architecture. The algorithm uses Eulerian tours of components of the graph in order to bound the amount of communication needed during execution of the algorithm.

I have been investigating the algorithms used to determine the clock period of MOS circuitry, and I have discovered that many in the literature are incorrect, many are exponential-time, and many combine these features. The problem is that MOS circuits are typically clocked by level-sensitive latches, rather than edge-triggered latches as is typical in TTL. It turns out that by using a little algorithmic graph theory, determining the clock period of MOS circuits can be done in polynomial-time.

M. Maley and I have discovered new algorithms for routing and testing routability of planar VLSI layouts. Miller has extended this work into an impressive theory of planar routing and routability. By using ideas from topology such as covering spaces, he has been able to prove a difficult theorem which reduces routability testing to checking relatively few constraints between objects in a design. The proof is general in that it incorporates most wiring models that have been studied in the literature or implemented in design systems. He has also discovered polynomial-time algorithms for optimally compacting VLSI layouts with automatic jog introduction, and he has improved on the basic algorithm that is typically used to solve constraint systems in normal compaction.

C. Phillips and I have discovered a new algorithm for determining connected components of rectangles in the plane which provably works well in a system environment where rectangles are stored in secondary memory. This algorithm easily generalizes to multiple layers, and hence can be used to determine electrical connectivity in a VLSI design rule checker. Cindy has also discovered new data structures for representing overlapping intervals which are considerably simpler than previous data structures.

We have designed and implemented two switches for bit-serial messages. One, built by T. Cormen, is a hyperconcentrator switch which is capable of mapping messages from many incoming lines onto fewer outgoing lines. The other is a pseudorandom permuter, built by C. Phillips, is a pseudorandom permuter which scrambles the order of messages.

2.7. Albert R. Meyer

My research has mainly been on axiomatic and denotational semantics as a guide to programming language understanding and design. Emphasis has been on metamathematical results such as completeness and decidability of formal theories. Principal subtopics include

- semantics and proof theory of ALGOL-like languages,
- generalized type structures, especially polymorphism, in programming languages,
- model theory and decision problems for λ-calculi

I have been supervising six Ph.D students (Mitchell, Breazu, Cosmadakis, McAllester, Bercovici, Gasarch), one M.S. Student (Reinhold), and two Senior Theses (Kilian, Hailperin).

My latest grant proposal is being generously funded for the next three years by the standards of the NSF CS Theory Section -- which is to say, I must seek substantial further funding, probably within the LCS DARPA grant.

2.8. Alan Sherman

A. Sherman has been investigating the relationship between algebraic and security properties of cryptosystems. Sherman and Rivest have shown that any finite, deterministic cryptosystem that generates a small group is vulnerable to a known-plaintext that runs in | K time on the average, where K is the size of the keyspace. In addition, they have developed a related statistical test to determine whether or not a deterministic cryptosystem generates a small group. With B. Kaliski, they have applied their test to the Data Encryption Standard (DES) using a combinations of software and special-purpose hardware. Initial trials of their experiment show, with high confidence, that DES does not generate a small group. Additional experiments are in progress to test DES for other related algebraic weaknesses. Currently, Sherman is investigating lower bounds on the time required to test a cryptosystem for various types of algebraic weaknesses.

Sherman expects to complete his Ph.D. by August 31, 1985 and, in September, will join the Computer Science faculty at Tufts University.

2.9. Peter Shor

P. Shor has been investigating the average-case behavior of bin packing algorithms. In the case where the item sizes are uniformly distributed, he has derived much tighter bounds on the wasted space produced by the algorithm First Fit than were previously known, and has the exact answer, up to a constant, for the wasted space produced by the algorithm Best Fit. He has also derived a lower bound for any on-line that shows that on-line algorithms cannot do as well as off-line algorithms, and that Best Fit comes within a small factor of being optimal among on-line algorithms. The same techniques seem like they should be applicable to the grid matching problem, and Peter Shor and Tom Leighton are investigating this.

2.10. Laura Yedwab

L. Yedwab will be completing her master thesis this summer. This culminates a year's work on the theory of disjunctive sum of games and how it relates to the endgame of GO. She plans to spend this fall trekking through Nepal. She will return to MIT in the spring to start the Ph.D. program.

Publications

1. Baldwin, R. and Sherman, A.T. "How we solved the $100,000 Decipher Puzzle," in preparation.

2. Berman, F., Leighton, F.T., Shor P. and Snyder, L. "Generalized Planar Matching," MIT-VLSI Memo, Cambridge, MA, March 1985.

3. Breazu-Tannen, V. and Meyer, A.R. "Lambda Calculus with Constrained Types," *Proceedings of the Logic of Programs Workshop*, June 1985, Springer-Verlag, Lecture Notes in Computer Science, to appear.

4. Bruce, K.B., Meyer, A.R. and Mitchell, J.C. "The Semantics of Second-Order Lambda Calculus," invited for special issue of Information and Control (1985).

5. Bui, T., Chaudhuri, S., Leighton F.T. and Sipser, M. "Graph Bisection Algorithms with Good Average Case Behavior," *Proceedings of the 25th Conference on the Foundations of Computer Science*, October 1984, 181-192.

6. Buss J. and Shor, P. "On the Pagenumber of Planar Graphs," *16th Annual Symposium on Theory of Computing*, 98-100, 1984.

7. Chandra, A., Halpern, J.Y., Meyer, A.R. and Parikh, P. "Equations between Regular Terms and an Application to Process Logic," to appear.

8. Chung, F.R.K., Leighton, F.T. and Rosenberg, A.L. "Embedding Graphs in Books: A Layout Problem with Applications to VLSI Design," MIT-VLSI Memo, Cambridge, MA, March 1985.

9. Kaliski, B.S., Rivest, R.L. and Sherman, A.T. "Is the Data Encryption Standard a Group?" *Proceedings of Eurocrypt '85*, to appear.

10. Kaliski, B.S., Rivest, R. L. and Sherman, A.T. Is DES a Pure Cipher? (Results of more cycling experiments on DES)," submitted to Crypto '85.

11. Leighton, F.T. and A.L. Rosenberg, A.L. "Three-Dimensional Circuit Layouts," MIT/LCS/TM-262, MIT Laboratory for Computer Science, Cambridge, MA, September 1984.

12. Loui, M.C., Meyer, A.R., Halpern, J.Y. and Weise, D. "On Time Versus Space III," submitted to *Mathematical Systems Theory*, (1985).

13. Meyer, A.R. and Tiuryn, J. "Equivalences Among Logics of Programs," *Journal Computer and System Sciences,* 29,2 (1984), 160-169.

14. Meyer, A.R. and Wand, M. "Continuation Semantics in Typed Lambda Calculi," *Proceedings of the Logic of Programs Workshop*, June 1985, Springer-Verlag, Lecture Notes in Computer Science, to appear.

15. Meyer, A.R. "Thirteen Puzzles in Programming Logic," *Proceedings of the Workshop on Formal Software Development: Combining Specification Methods,* Nyborg, Denmark, (May, 1984), D. Bjorner, (Ed.), Springer-Verlag, Lecture Notes in Computer Science, to appear (1985).

16. Mitchell, J.C. and Meyer, A.R. "Second-order Logical Relations," Proceedings of the Logic of Programs Workshop, June 1985, Springer-Verlag, Lecture Notes in Computer Science, to appear.

17. Sherman, A. T. "Cryptography and VLSI (a two-part dissertation): I. Foundations for Secure Communications II. Placing Modules on a VLSI Chip," thesis proposal, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 7, 1984.

18. Shor, P. "The Average-case Analysis of Some On-line Algorithms for Bin Packing," *25th Symposium on Foundations of Computer Science*, 193-200, 1984.

19. Sistla, A.P., Clarke, E.M., Francez, N. and Meyer, A.R. "Can Message Buffers be Axiomatized in Linear Temporal Logic?", to appear, *Information and Control* (1985).

Talks

1. Bui, T.

   -- 25th FOCS, Florida, October. 1984.
   -- MIT VLSI Research Review, MIT, Dec. 1984.

2. Leighton, F.T "Networks, Parallel Computation and VLSI," Special 2-lecture series presented to the computer science department at the University of Chicago in January, 1985, and to the Annual Southeastern Conference on Combinatorics, Graph Theory and Computing in Boca Raton, February 1985.

3. Leighton, F.T. "The Average Case Behavior of Algorithms for VLSI, Bin Packing and Dynamic Allocation," presented to the Oberwolfach Workshop on Algorithms in Germany in November, 1984, to the IBM T.J. Watson research center in November and to the MIT Math department in December.

4. Meyer, A.R. "The Complexity of Flow-Analysis: Application of a Fundamental Theorem of Denotational Semantics," Logic and Semantics of Programs Workshop, Bad Honnef, West Germany, March, 1985; MIT, Lab. for Computer Science, April, 1985; Stanford University, Department of Computer Science, April, 1985.

5. Meyer, A.R. "Logical Puzzles in Programming," Carnegie-Mellon University, Distinguished Lecture in Computer Science, September, 1984; University of Maryland, Distinguished Lecture in Computer Science, October, 1984; Brandeis University, Department of Computer Science, November. 1984; Yale University, Department of Computer Science, Dec. 1984; University of California, Santa Cruz, Department of Computer Science, Jan. 1985; University of Texas, Austin, Department of Computer Science, May, 1985.

6. Meyer, A.R. "The State of the Logic of State," Workshop on Formal Software Development: Combining Specification Methods, Nyborg, Denmark, May, 1984.

7. Meyer, A.R. "Lambda-calculus and Computer Science," University of Pisa, Italy, Computer Science Department, June, 1984; University of Maryland, Department of Mathematics, October, 1984.

8. Meyer, A.R. "What Makes the Free-list free?," Workshop on Formal Software Development: Combining Specification Methods, Nyborg, Denmark, May 1984; SRI, Palo Alto, CA, April, 1985.

9. Sherman, A.T. "Is the Data Encryption Standard a Group?" Eurocrypt 85, April 10, 1985, Linz, Austria.

10. Sherman, A.T. "Two Recent Cryptologic Results: I. Is the Data Encryption Standard a Group? II. How we solved the $100,000 Decipher Puzzle," invited talk, Brandeis University, April 18, 1985, Waltham, Massachusetts.

THEORY OF DISTRIBUTED SYSTEMS

Academic Staff

N. A. Lynch, Group Leader

Graduate Students

B. Bloom	E. Stark
B. Coan	M. Tuttle
J. Lundelius	

Support Staff

E. Pothier

Visitors

B. Awerbuch	P. Kanellakis
C. Dwork	M. Merritt

1. OVERVIEW

Members of the Theory of Distributed Systems group worked on a variety of problems related to distributed computing. The activities of its members are summarized in the following individual reports.

2. INDIVIDUAL PROGRESS REPORTS

2.1. Baruch Awerbuch

My research has been focused on the field of distributed computation in communication networks. The main emphasis was put on the following problems:

1) distributed shortest paths computation

2) coin-flipping in unreliable networks

3) detection and resolution of deadlocks

A more detailed description is given below.

1) The research on shortest paths in networks was conducted jointly with R. Gallager. This problem arises in relation with the problem of efficient routing of messages towards a certain destination node. We have proposed a number of algorithms, whose complexities are close to optimal.

2) A problem of flipping a fair common coin in unreliable networks was investigated jointly with B. Chor, S. Goldwasser and S. Micali. This is a very fundamental problem since many existing protocols rely heavily upon existence of such a coin, while no satisfactory solution has been proposed so far. In this research, we have proposed a new efficient protocol for coin flipping that produces a provably fair coin in presence of malicious adversaries. Our solution strongly uses cryptography. However, cryptographic assumptions needed in the protocol are very weak. Namely, we assume that trapdoor functions exist. This is the weakest possible assumptions because otherwise cryptography is useless.

3) Together with S. Micali, we have investigated the problem of detection and resolution of deadlocks in communication networks. Efficient solution of this problem will result in improved performances of communication networks. However, no such efficient solution is currently known. The main contribution of our research is a new protocol for deadlock resolution which is much more efficient than the existing protocols.

Work in progress includes: "How Local is Maximal Matching?" with O. Goldreich.

2.2. Bard Bloom

Multi-Writer Atomic Registers: Shared-variable multiprocessor algorithms often assume that any processor can read from and write to any shared variable; and that writes and reads cannot overlap. Unfortunately, it is hard to make real memory with these properties. It is somewhat easier to make memory which one processor can write to and arbitrary numbers can read from safely. I have developed an algorithm to simulate two-writer memory with two sets of one-writer memory, with a fairly small overhead.

Theoretical Artificial Intelligence with R. Rivest: We are trying to develop a theory of AI, hopefully bearing the same relation to AI that automata and complexity theory do to the rest of computer science. At the moment, we are investigating learning. under what circumstances is learning possible? reasonable? a good strategy? We have found several types of worlds in which learning is good, and a few in which it is not.

Denotational Semantics of Concurrency (with N. Lynch and A. Meyer). At the moment, we are trying to catch up with the people who have been working on semantics for the last dozen years.

2.3. Brian A. Coan

I have continued my work on fault-tolerant distributed algorithms. There are three principal areas of progress reported below.

First, the work reported last year on efficient randomized Byzantine agreement algorithms has been concluded. The results of this work are reported in a joint paper with B. Chor [2]. We developed a new randomized Byzantine agreement algorithm. The algorithm operates in a synchronous fully-connected system of n processors, at most t of which may fail. The algorithm terminates after an expected $O(t/\log n)$ rounds of message exchange. This performance is further improved to a constant expected number of rounds if the distribution of processor failures is assumed to be uniform. In either event, the algorithm improves on the known lower bounds for rounds for deterministic algorithms. This algorithm may be of practical interest because it is both simple and efficient.

Second, I have been working with D. Dolev, C. Dwork, and L. Stockmeyer on the distributed firing squad problem. For a discussion of this work see the progress report of Cynthia Dwork.

Third, I have been working on upper bounds on communication requirements for fault-tolerant distributed algorithms. [4] The technique I have devised works for various fault models: fail-stop, failure-by-omission, and Byzantine. Specifically, the technique is a two-step transformation. The first step is a transformation into a well known [7] communication-inefficient canonical form in which each processor, at each round, broadcasts its entire state. The second step is a new transformation from this communication-inefficient canonical form to to a communication-efficient canonical form. For each fault model there is a separate transformation. The transformation in the Byzantine model is fully worked out. I am currently attempting to devise other more efficient transformations for the two less severe fault models.

As a corollary to the results in the Byzantine fault model, I obtained a major new result about the communication requirements of Byzantine agreement. This new result is a polynomial-message Byzantine agreement algorithm that uses about half the rounds of communication used by any other polynomial-message algorithm.

2.4. Cynthia Dwork

Research

(1) Deterministic Distributed Firing Squad (with B. Coan, D. Dolev, and L. Stockmeyer)

Most fault-tolerant distributed algorithms assume a "synchronous system", in which processing is divided into rounds of message exchange. This assumption is justified by the impossibility results of Fischer, Lynch, and Patterson; and Dolev, Dwork, and Stockmeyer, which show that if the system is asynchronous then there is no protocol for distributed agreement tolerant to even one benign processor failure. Another common assumption is that all processors begin the algorithm simultaneously, i.e., in the same round. In an actual distributed system in which different transactions and algorithms may be executed periodically, this may be unrealistic. Typically an algorithm is executed in response to some external request made to a specific processor. If the given processor is correct then all correct processors learn of the request simultaneously, so they can indeed begin the algorithm in unison. However, if the processor is faulty then the correct processors may learn of the requested transaction at different times, if at all. We were able to justify the design assumption of simultaneous starts. Specifically, we obtained algorithms to solve the associated synchronization problem, which we call the "distributed firing squad" problem (abbreviated DFS).

In addition to the two standard fault models of fail-stop and authenticated Byzantine behavior we considered two new types of faulty behavior. In "rushing" a malicious faulty processor can receive, process and re-send messages "between" the synchronous steps of other processors. In the "timing fault" model faulty processors follow their transition functions precisely but may take steps at irregular times and/or may experience slight delays or accelerations in communicating with the other processors. The unauthenticated Byzantine model has been studied by Burns and Lynch, who have found a way of transforming any agreement protocol in this model into a DFS protocol. We obtained the converse reduction for the fail-stop model.

Our principal results are:

tight bounds on the running time required to solve DFS in the fail-stop, authenticated Byzantine, and rushing models;

tight bounds on the number of processors required to solve DFS in the fail-stop, timing fault, and rushing models.

(2) Randomized Distributed Firing Squad (with B. Coan)

DFS appears implicitly in work of Rabin. He had an algorithm for reaching consensus whose expected running time was constant. However, unlike previous consensus algorithms, his allowed correct processors to make their decisions at different rounds. The implicit question was, Does there exist an algorithm for achieving simultaneity that runs in time strictly less than O(t) (the lower bound for agreement in a deterministic algorithm)? We have answered this question negatively: we have shown that not only is there no "fast" deterministic firing squad algorithm, but there is not even a randomized DFS algorithm whose expected running time is less than t+1 rounds, where the expectation is taken over the coin flip sequences.

Work in Progress

(3) Unification and Term Matching (with P. Kanellakis and L. Stockmeyer)

We have strengthened the results in "On the Sequential Nature of Unification" (Dwork, Kanellakis, and Mitchell). We have shown certain interesting special cases of unification to be complete for P with respect to log-space reductions. We are currently searching for a more processor-efficient parallel algorithm for one-way unification or term matching, as well as trying to obtain lower bounds on this measure. We are also searching for tighter lower bounds on congruence closure.

(4) Knowledge and Common Knowledge in the Presence of Faults (with Y. Moses)

We have been extending the work of Halpern and Moses on knowledge and common knowledge in distributed systems to describe states of knowledge in a system in which processors can fail. We have obtained a new, very intuitive proof of the lower bound on rounds required to reach agreement. We are currently examining how states of knowledge are affected by properties of the communications media of the system.

2.5. Paris C. Kanellakis

Research

(1) "Partition Semantics for Relations," with S.S. Cosmadakis, P.C. Kanellakis, N. Spyratos.

We use set-theoretic partitions to provide models for relation schemes, relations and dependencies. Our new point of view of the relational model demonstrates that there is a natural extension of functional dependencies (FD's), which is based on the duality between product and sum of partitions. These partition dependencies (PD's) have the power to express both functional determination and transitive closure of undirected graphs. The inference problem of PD's is shown to be the uniform word problem in a lattice. We provide a polynomial time algorithm for this natural generalization of the FD inference problem. We show how partition semantics justify a number of variants of the weak instance assumption and investigate the expressive power of PD's. We also provide a polynomial time test for consistency of a set of relations with a set of PD's.

(2) "On the Analysis of Cooperation and Antagonism in Networks of Communicating Processes," with P.C. Kanellakis and S.A. Smolka.

We propose a new method for the analysis of cooperative and antagonistic properties of communicating finite state processes (FSP's). This algebraic technique is based on a composition operator and the notion of "possibility equivalence" among FSP's. We demonstrate its utility by showing that potential blocking, lockout, and termination can be efficiently decided for loosely connected networks of tree FSP's. If not all acyclic FSP's are trees, then the cooperative properties become NP-complete and the antagonistic ones PSPACE-complete. For tightly coupled networks of tree FSP's, we also have NP-hardness. For the considerably harder cyclic process case, we provide a natural extension of the method as well as a subcase reducible to integer programming with a constant number of variables.

(3) "Equational Theories and Database Constraints," with S.S. Cosmadakis and P.C. Kanellakis.

We present a novel way to formulate database dependencies as sentences of first-order logic, using equational statements instead of Horn clauses. Dependency implication is directly reduced to equational implication. Our approach is powerful enough to express functional and inclusion dependencies, which are the most common database constraints. We present a new proof procedure for these dependencies. We use our equational formulation to derive new upper and lower bounds for the complexity of their implication problems. The conceptual contribution is the merging of two different subareas of computer science: database logic and equational theories. Our approach also extends to algebraic dependencies, (see Ph.D. thesis by S.S. Cosmadakis).

(4) "Two Applications of Equational Theories to Database Theory," with S.S. Cosmadakis and P.C. Kanellakis.

This is a description of some of the results of 1) above for lattices, and some of the potentially useful transformations of 3) above. We have used 3) to experiment with J. Guttag's REVE system for functional and inclusion statements. We describe our experiments with the Knuth-Bendix procedure, which attempts to compile database constraints into a rewrite-rule system (this is a much more useful and manageable form for describing constraints). Surprisingly REVE compiled every hard example of functional and inclusion dependencies we provided. We believe that conventional equational theorem provers can be used to manipulate logical database constraints in an (practically) efficient fashion.

(5) "ISIS: Interface for a Semantic Information System," with K.J. Goldman, S.A. Goldmann, P.C. Kanellakis, S.B. Zdonik.

ISIS is an experimental system for graphically manipulating a database. The system is based on a simply specified high-level semantic data model. It demonstrates the capabilities of a workstation environment by integrating three aspects of database programming in one graphical setting. Namely, it allows browsing at the schema and data levels, it provides a graphical query language, and it permits direct manipulation of the schema. In all these activities it treats data and schema uniformly.

(6) "On Term-Matching and Unification," with C. Dwork and L. Stockmeyer.

We have obtained a more efficient (under certain representation assumptions) parallel algorithm for one-way unification or term matching. This is the basic operation for term rewriting, and is becoming quite popular with the PROLOG community. Tighter lower bounds on unification on terms, and congruence closure have also been obtained. We are currently working on lower bounds for the matching problem.

(7) "On AC Unification," with A. Chandra.

AC-unification is NP-hard. This is a joint observation with A. Chandra. (AC-matching is NP-complete). For many AC-operators (associative commutative symbols, that do not interact with each other) it is non-trivial to show that the problem is decidable, (this is a result by one of G. Huet's students). At present trying to show that problem is in NP. Also working on showing that for any number of AC and Idempotent operators unification is in PTIME.

(8) "The Classification of Recursive Rules"

An important open problem in database theory is the classification of recursive queries (i.e., queries in PROLOG without function symbols). If we consider each query Q fixed, and the database D as the variable input then evaluation of each Q for any D is in PTIME. The open question is deciding which cases can be evaluated in the parallel class NC. Some preliminary results have been derived in this area.

2.6. Jennifer Lundelius

I finished my master's thesis, which concerned synchronizing clocks in a distributed computer system. We assume that the processes are distributed, fully connected for communication, and communicate only by sending messages. Message delays fall within a known range. Each process has a real-time clock, which is subject to a small rate of drift.

The first result in the thesis is a lower bound on how closely the clocks can be synchronized, even if the clocks all run at a perfect rate, and there are no faulty processes, as long as there is some uncertainty in how long a message takes to be delivered. The intuition behind the proof is that in two executions of any clock synchronization algorithm, clock values can be different, but that fact can be masked by an appropriate change in the message delays. We use this technique of altering executions to obtain constraints on the closeness of synchronization to produce the lower bound. A simple algorithm, which synchronizes the clocks to within the lower bound, is included to show that the bound is tight.

The thesis also includes algorithms to synchronize clocks in the more realistic case where clocks can drift and some percentage of the processes can fail arbitrarily. The maintenance algorithm, used when clocks are initially synchronized, works by correcting for drift every so often. At each round, processes exchange clock values, compute a function of the values received, and update their clocks. The choice of function is crucial. If the algorithm is to tolerate f faulty processes, then there must be more than 3f processes altogether. The function consists of discarding the f highest and f lowest values and taking the midpoint of the rest. The use of this algorithm allows the clock values to stay close together, keeps the clock values close to real time, and makes it easy for a repaired process to become synchronized with the rest of the system.

The second algorithm presented in the thesis can be used when clock values begin arbitrarily far apart. The same basic idea as in the maintenance algorithm is employed. Since each round of the maintenance algorithm cuts the differences between the clock values roughly in half, successive rounds cause repeated halving of the differences, and will bring the clocks closer and closer. The only problem is knowing when to start each round. In order to solve this problem, an extra phase is inserted in each round to synchronize the beginning of the next round.

Work currently in progress concerns studying the capabilities that time gives in a distributed computation. For example, when clocks are used to provide timeouts in a communication protocol, the failure of a channel to deliver a message can be detected. Essentially, time is being used to implement the failure-detection mechanism. Similarly, in many distributed transaction systems, timestamps provide a total order on some set of messages. Further study may show if it is easier to argue the correctness of protocols using the abstraction rather than the implementation.

A related area to investigate is that of modeling distributed systems that fall between the purely synchronous and purely asynchronous. Each system that uses time can be considered an example of a semi-synchronous model. What are the relationships between different models? Are there natural problems that separate the models into stronger and weaker ones?

2.7. Nancy Lynch

During this year, I worked on the following problems:

(1) Electing a leader in a synchronous ring.

With G. Frederickson, I developed a lower bound for the number of messages required to distinguish a unique processor in a synchronous ring of processors [8]. This new result shows that a large amount of communication is required to break symmetry, for arbitrary ring sizes, even if the size of the ring is known to all of the processors. There are two parts to this work: (a) a general theory which shows that certain information flow is necessary to distinguish processors whose sets of neighbors have "similar" identifiers, and (b) a combinatorial result that shows that rings of arbitrary sizes can be constructed in which many processors have such "similar" neighborhoods.

(2) Easy impossibility proofs for distributed consensus problems.

With M. Fischer and Michael Merritt, I developed easy new proofs of the impossibility of solving several consensus problems (Byzantine agreement, weak agreement, Byzantine firing squad, approximate agreement and clock synchronization) in certain communication graphs [12]. We showed that, in the presence of m faults, no solution to these problems exists for communication graphs with fewer than 3m+1 nodes or less than 2m+1 connectivity. While some of these results had previously been proved, the new proofs are much simpler, provide considerably more insight, apply to more general models of computation, and (particularly in the case of clock synchronization) significantly strengthen the results.

(3) Byzantine firing squad.

With J. Burns, I developed a fault-tolerant algorithm for a distributed synchronization problem [1]. Our "algorithm" is really a family of algorithms, derived via a simple transformation from arbitrary Byzantine agreement algorithms.

(4) Consensus in partially synchronous systems.

C. Dwork, L. Stockmeyer and I defined some very simple models for partially synchronous distributed systems [6]. (Partially synchronous models are between the usual synchronous and asynchronous models.) We were able to use these models directly for describing solutions to consensus problems. Results that we had previously obtained for many other models then follow via simulation results. Thus, these models provide a very useful and simplifying intermediate step in describing a class of apparently complicated distributed algorithms.

(5) Reliable broadcast protocols.

With H. Garcia-Molina, I designed a collection of reliable broadcast protocols for use in unreliable networks [9]. One of our protocols is currently being implemented at

Computer Corporation of America for use in their new highly-available distributed system.

(6) Completing old work.

(a) Jennifer Lundelius and I reformulated our results on limitations of the closeness within which software clocks can be synchronized [9]. The reformulation is in terms of a simpler and more general model than we had previously used.

(b) B. Weihl and I completed our work on the problem of reaching approximate agreement in a distributed system [5]. B. Coan and A. Fekete (a student from my Distributed Algorithms course) helped us to solve some of the remaining problems.

(c) N. Griffeth, L. Guibas, M. Fischer and I finally completed our difficult work on probabilistic analysis of a network resource allocation algorithm [11].

(d) I completed my paper on concurrency control for resilient nested transactions[10].

Several projects are now in progress:

I have been thinking about correctness conditions for distributed systems of the new type being proposed by Computer Corporation of America. These systems are designed to be highly available, fault-tolerant and fast. Unfortunately, they do not satisfy the usual strong correctness conditions which are guaranteed by conventional concurrency control algorithms. I am trying to determine what interesting correctness conditions this system does satisfy, so that an application designer will be able to make effective use of such a system.

With M. Merritt, P. Kanellakis and B. Weihl, I have been trying to develop a framework for understanding concurrency control algorithms. The framework we have in mind uses simulations of alternative executions. This work has proved to be very difficult.

M. Tuttle and I have been carrying out a proof of an interesting distributed resource allocation algorithm using levels of abstraction in a very natural and fruitful way.

B. Awerbuch, M. Fischer and I have been examining consensus problems in networks in which the communication links are unreliable.

Other work during the year included continued development of my new graduate course in "Distributed Algorithms", service on the STOC Symposium Program Committee, and service on the NSF Division of Computer Research Advisory Committee.

2.8. Michael Merritt

(1) Easy Impossibility Proofs for Distributed Consensus.

M. Fischer (Yale), N. Lynch and I examined impossibility results for a variety of agreement problems in unreliable environments. Our paper, "Easy Impossibility Proofs for Distributed Consensus Problems," makes use of properties of graph coverings to obtain a number of results in a clear, simple way. Many of these results were previously known, but with awkward and lengthy proofs. We are able to unify this work, and obtain some important new results in the area of clock synchronization.

(2) Expected Time to Reach Agreement.

Probabilistic algorithms have been found to have excellent average behavior for a variety of applications. Work at MIT, Harvard and Cornell has applied this technique to the problem of reaching agreement in unreliable systems. B. Chor, D. Shmoys (Harvard) and I have found several simple algorithms which reach agreement in constant expected time, and which work in different failure models.

(3) The Orphan Problem.

In nested transactions systems (such as ARGUS, here at MIT), 'orphans' can be created when high-level actions abort, and their subtransactions continue processing. Managing these orphans involves detecting and aborting them before they have a chance to see inconsistent data. Work in ARGUS has focused on implementing such management, although the problem has not been well understood. N. Lynch and I are developing a model for nested transaction systems which will allow a clear formulation of the correctness issues involved. We expect to argue the correctness of a number of algorithms developed at MIT, and hope to be able to address performance issues.

(4) Diagnosis Problems.

T. Dahbura (Bell Telephone Laboratories) and I have been exploring a generalization of the diagnosis problem he addressed in his thesis. We are exploring the problem of agreeing on a set of faulty processors in a network in which some processors test others, and the rest communicate over a communication network of varying topology. We both have a strong feeling that the diagnosis and agreement problems are related, and hope that this work will allow us to state the precise relationship underlying this intuition.

2.9. Mark R. Tuttle

Tuttle has spent the majority of his first year studying techniques for verifying the correct behavior of distributed algorithms. The vast majority of the techniques he has studied have tried to create a logical framework in which to reason about the correctness of the algorithm. Most of the techniques use predicate calculus to make statements about the state of each process at certain points in their execution. More recently Tuttle has been looking for ways to extend work done by N. Lynch, G. Stark, and J. Goree in which the algorithm is studied at varying levels of abstraction. At each level of abstraction one defines an algebra consisting of a set of states and a set of partial operations on the states. Algebras lower on the scale of abstraction are mapped up to more abstract algebras in a way that ensures that any behavior that can be exhibited in the lower algebra can be simulated at the higher level until the level of abstraction is high enough that one may reason about the behavior of the algorithm at a very conceptual level without concerning himself with the details of implementation. This technique has the advantage of conceptual simplicity which allows a person to use her intuition to construct a formal proof. The investigator may satisfy herself that the algorithm exhibits certain local properties and then use these properties to verify the algorithm while avoiding the tyranny of detail seen in some of the techniques in the literature. Currently, Tuttle is using this technique to verify the correctness of an interesting arbiter algorithm. In the future, Tuttle intends to continue the study of verification and to look into resource allocation and communications problems.

Tuttle spent the second semester of this year as a teaching assistant in 6.004 Computational Structures, the undergraduate computer architecture course, and as a laboratory assistant in the digital logic lab for the same course.

References

1. Burns, J. and Lynch, N. "The Byzantine Firing Squad Problem"

2. Chor, B. and Coan, B. A. "A Simple and Efficient Randomized Byzantine Agreement Algorithm," *Proceedings 4th Symposium on Reliability in Distributed Software and Database Systems* 1984, 98-106, selected for republication in *IEEE Transactions on Software Engineering*, June 1985.

3. Coan, B. A., Dolev, D., Dwork, C. and Stockmeyer, L. "The Distributed Firing Squad Problem," *Proceedings of the 17th ACM Symposium on Theory of Computing*, May 1985, 335-345.

4. Coan, B. A. "Communication-Efficient Canonical Forms for Fault-Tolerant Distributed Algorithms," manuscript in progress.

5. Dolev, D., Lynch, N., Pinter, S., and Stark, W. "Reaching Approximate Agreement in the Presence of Faults," to appear *Journal of the ACM.*

6. Dwork, C., Lynch, N. and Stockmeyer, L. "Consensus in the Presence of Partial Synchrony", *Proceedings of 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Vancouver, B.C., Canada, August 1984, 103-118.

7. Fischer, M. and Lynch, N. "A Lower Bound for the Time to Assure Interactive Consistency," *Information Processing Letters* 14(4) (1982).

8. Frederickson, G. and Lynch, N. "A General Lower Bound For Electing a Leader In a Ring"

9. Garcia-Molina, H. and Lynch, N. "Reliable Broadcast Protocols for Unreliable Networks," Computer Corporation of America internal memo. Lundelius, J. and Lynch, N. "An Upper and Lower Bound for Clock Synchronization", to appear in *Information and Control.*

10. Lynch, N. "Concurrency Control for Resilient Nested Transactions," to appear in *The Theory of Databases.*

11. Lynch, N., Griffeth, N., Fischer, M. and Guibas, L. "Probabilistic Analysis of a Network Resource Allocation Algorithm," to appear in *Information and Control.*

12. Merritt, M., Fischer, M. and Lynch, N. "Easy Impossibility Proofs for Distributed Consensus Problems," has been accepted at the Principles of

Distributed Computation conference this August, and was invited to appear in the first issue of a new journal, *Distributed Computing.*

Publications

1. Awerbuch, B. "A New Distributed Depth-First Search Algorithm," to appear in *Information Processing Letters.*

2. Awerbuch, B. "An Efficient Network Synchronization Protocol," to appear in *Journal of the ACM.*

3. Awerbuch, B., Chor, B., Goldwasser, S. and Micali, S. "Provably Secure Coin in a Byzantine Environment," to appear.

4. Awerbuch, B. and Gallager, R. "Distributed Breadth-First-Search Algorithms," to appear.

5. Awerbuch, B. and Micali, S. "Complexity of Resolution and Detection of Deadlocks," to appear

6. Awerbuch, B. "Reducing complexities of distributed Maximum Flow and Breadth-First Search Algorithms by means of Network Synchronization," to appear in *Networks.*

7. Awerbuch, B. and Gallager, R. "Communication Complexity of Distributed Shortest Path Algorithms," submitted to the *IEEE Transactions on Information Theory.*

8. Awerbuch, B. "An Efficient Network Synchronization Protocol," *Proceedings of the 16th Annual ACM Symposium on Theory of Computing,* Washington, DC, April 1984, 522-525.

9. Awerbuch, B., Chor, B., Goldwasser, S. and Micali, S. "Provably Secure Coin in a Byzantine Environment," to appear in Publication to Conference.

10. Awerbuch, B. and Even, S. "Efficient and Reliable Broadcast is Achievable in an Eventually-Connected Network," *Proceedings of 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing,* Vancouver, B.C., Canada, August 1984, 278-281.

11. Awerbuch, B., Israeli, A. and Shiloach, Y. "Finding Euler Circuits in Logarithmic Parallel Time," *Proceedings of the 16th Annual ACM Symposium on Theory of Computing,* Washington, DC, April 1984, 249-257.

12. Burns, J. and Lynch, N. "The Byzantine Firing Squad Problem," to appear.

13. Chor, B. and Coan, B. A. "A Simple and Efficient Randomized Byzantine Agreement Algorithm," *Proceedings 4th Symposium on Reliability in Distributed Software and Database Systems* 1984, 98-106, selected for republication in *IEEE Transactions on Software Engineering*, June 1985.

14. Coan, B. A., Dolev, D., Dwork, C. and Stockmeyer, L. "The Distributed Firing Squad Problem," *Proceedings of the 17th ACM Symposium on Theory of Computing*, May 1985, 335-345.

15. Dolev, D., Dwork, C. and Stockmeyer, L. "On the Minimal Synchronism Needed for Distributed Consensus," submitted to *JACM.*

16. Dolev, D., Lynch, N., Pinter, S., and Stark, W. "Reaching Approximate Agreement in the Presence of Faults," to appear *Journal of the ACM.*

17. Dwork, C., Lynch, N. and Stockmeyer, L. "Consensus in the Presence of Partial Synchrony," *Proceedings of 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Vancouver, B.C., Canada, August 1984, 103-118.

18. Dwork, C., and Skeen, D. "Patterns of Communication in Consensus Protocols," *Proceedings of 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Vancouver, B.C., Canada, August 1984, 143-153.

19. Frederickson, G. and Lynch, N. "A General Lower Bound For Electing a Leader In a Ring," to appear.

20. Kanellakis, P. C. and Papadimitriou, C. H. "The Complexity of Distributed Concurrency Control," *SIAM Journal of Computing*, 14, 1, February 1985, 52-75.

21. Kanellakis, P. C. and Smolka, S. A. "On the Analysis of Cooperation and Antagonism in Networks of Communicating Processes," *Proceedings 4th ACM SIGACT-SIGOPS Principles of Distributed Computing*, (August 1985, to appear).

22. Kanellakis, P. C. and Cosmadakis, S. S. "Equational Theories and Database Constraints," *Proceedings 17th Annual ACM Symposium on Theory of Computing*, Providence, RI, May 1985.

23. Kanellakis, P. C. and Cosmadakis, S. S. "Two Applications of Equational Theories to Database Theory," *Proceedings First International Conference on Rewriting Techniques and Applications*, May 1985.

24. Kanellakis, P. C., Cosmadakis, S. S. and Spyratos, N. "Partition Semantics for Relations," *Proceedings 4th ACM SIGACT-SIGMOD PODS*, March 1985. Submitted to *JCSS*, special issue on the 4th PODS conference (J.D. Ullman, editor).

25. Kanellakis, P. C., Goldman, K. J., Goldman, S. A. and Zdonik, S. B. "ISIS: Interface for a Semantic Information System," *Proceedings ACM SIGMOD*, May 1985.

26. Lundelius, J. and Lynch, N. "A New Fault-Tolerant Algorithm for Clock Synchronization," *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, Vancouver, B.C., Canada, August 1984, 75-88. Lundelius, J. and Lynch, N. "An Upper and Lower Bound for Clock Synchronization", to appear in *Information and Control.*

27. Lynch, N. "Concurrency Control for Resilient Nested Transactions," to appear in *The Theory of Databases.*

28. Lynch, N., Griffeth, N., Fischer, M. and Guibas, L. "Probabilistic Analysis of a Network Resource Allocation Algorithm," to appear in *Information and Control.*

29. Merritt, M., Chor, B. and Shmoys, D. "Simple Constant-Time Consensus Protocols in Realistic Failure Models: Extended Abstract," has been accepted at the *Proceedings 4th ACM SIGACT-SIGOPS Principles of Distributed Computation* conference this August.

30. Merritt, M. "Elections in the Presence of Faults," *Proceedings of 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing,* Vancouver, B.C., Canada, August 1984, 134-142.

31. Merritt, M. J. and Mitchell, D. P. "A Distributed Algorithm for Deadlock Detection and Resolution," *Proceedings of 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, Vancouver, B.C., Canada, August 1984, 282-284.

32. Merritt, M., Fischer, M. and Lynch, N. "Easy Impossibility Proofs for Distributed Consensus Problems," has been accepted at the Principles of Distributed Computation conference this August, and was invited to appear in the first issue of a new journal, *Distributed Computing.*

Theses Completed

1. Lundelius, J. "Synchronizing Clocks in a Distributed System," MIT/LCS/TR-335, MIT Laboratory for Computer Science, Computer Science, Cambridge, MA, August 1984.

2. McKay, E. N. "A Methodology For Software Implemented Transient Error Recovery In Spacecraft Computation," S.B. and S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1985.

3. Stark, E. W. "Foundations of a Theory of Specification for Distributed Systems," Ph.D dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1984.

Thesis in Progress

1. Coan, B. A. "Fundamental Problems in Fault-Tolerant Distributed Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1986.

Talks

1. Awerbuch, B. "Complexity of Network Synchronization,"
   University of California, Berkeley, August 1984
   Stanford University, August 1984
   IBM San Jose, CA., August 1984
   IBM Yorktown, NY, September 1984
   MIT, Cambridge, MA. October 1984
   GTE Labs, October 1984
   University of Toronto, November 1984
   Harvard University, December 1984
   Yale University, January 1985

2. Awerbuch, B. "Distributed BFS Algorithms,"
   GTE Labs, February 1985
   MIT, Cambridge, MA, March 1985
   University of California Berkeley, April 1985

3. Coan, B. A. "A Simple and Efficient Randomized Byzantine Agreement Algorithm,"
   17th Annual ACM Symposium on Theory of Computing,
   Providence, RI, May 1985

4. Coan, B. A. "The Distributed Firing Squad Problem," Symposium on

Reliability in Distributed Software and Database Systems, Silver Spring, MD, October 1985

5. Dwork, C. "On the Sequential Nature of Unification,"
   Bay Area Theory Seminar (Berkeley), December 1984
   IBM San Jose Research Laboratory, December 1984
   Singapore National University, January 1985
   Chinese University of Hong Kong, January 1985
   Stanford University, April 1985

6. Dwork, C. "The Distributed Firing Squad Problem,"
   University of California, Berkeley, January 1985
   University of British Columbia, January 1985
   University of Washington, Seattle, January 1985
   Digital Equipment Corp. Systems Res. Center., February 1985
   IBM San Jose Research Laboratory, February 1985
   Bell Telephone Laboratories, February 1985
   Cornell University, February 1985
   IBM T.J. Watson Research Center, February 1985
   University of Texas, Austin, February 1985
   University of Wisconsin, Madison, March 1985
   University of Toronto, March 1985
   MIT, March 1985

7. Kanellakis, P. C. "On the Parallel Complexity of Unification,"
   University of Toronto, December 1984
   Micro-Electronics Computer, Texas, January 1985

8. Kanellakis, P. C. "Database Theory and Reasoning about Equations,"
   GE Research and Development Labs, April 1985
   MIT Seminar April 1985

9. Kanellakis, P. C. "ISIS: Interface for a Semantic Information System," Computer Corporation of America, May 1985

10. Lundelius, J. (joint work with N. Lynch) "A New Fault-Tolerant Algorithm for Clock Synchronization," ACM Symposium on Principles of Distributed Computing, Vancouver, BC, Canada, August 1984

11. Lynch, N. "Distributed Agreement in Asynchronous Systems,"
    University of Texas, Austin, October 1984
    University of Wisconsin, Madison, February 1985
    Brooklyn College, March 1985

12. Lynch, N. "Electing a Leader in a Synchronous Ring,"
    University of Texas, Austin, October 1984
    University of Wisconsin, Madison, February 1985

13. Lynch, N. "Easy Impossibility Proofs for Distributed Consensus Problems," University of Wisconsin, Madison, February 1985

14. Lynch, N. "Recent Results on Reaching Approximate Agreement in a Distributed System," Columbia University, Symposium on Approximately Solved Problems, April 1985

15. Merritt, Michael "Easy Impossibility Proofs for Distributed Consensus Problems,"
    Harvard University, October 1984
    Georgia Institute of Technology, January 1985

PUBLICATIONS

Technical Memoranda

TM-10[3]	Jackson, J.N. Interactive Design Coordination for the Building Industry, June 1970, AD 708-400
TM-11	Ward, P.W. Description and Flow Chart of the PDP-7/9 Communications Package, July 1970, AD 711-379
TM-12	Graham, R.M. File Management and Related Topics June 12, 1970, September 1970, AD 712-068
TM-13	Graham, R.M. Use of High Level Languages for Systems Programming, September 1970, AD 711-965
TM-14	Vogt, C.M. Suspension of Processes in a Multi-processing Computer System, September 1970, AD 713-989
TM-15	Zilles, S.N. An Expansion of the Data Structuring Capabilities of PAL, October 1970, AD 720-761
TM-16	Bruere-Dawson, G. Pseudo-Random Sequences, October 1970, AD 713-852
TM-17	Goodman, L.I. Complexity Measures for Programming Languages, September 1971, AD 729-011
TM-18	Reprinted as TR-85
TM-19	Fenichel, R.R. A New List-Tracing Algorithm, October 1970, AD 714-522

[3] TMs 1-9 were never issued.

PUBLICATIONS

TM-20	Jones, T.L. A Computer Model of Simple Forms of Learning, January 1971, AD 720-337
TM-21	Goldstein, R.C. The Substantive Use of Computers For Intellectual Activities, April 1971, AD 721-618
TM-22	Wells, D.M. Transmission Of Information Between A Man-Machine Decision System And Its Environment, April 1971, AD 722-837
TM-23	Strnad, A.J. The Relational Approach to the Management of Data Bases, April 1971, AD 721-619
TM-24	Goldstein, R.C. and Strnad, A.J. The MacAIMS Data Management System, April 1971, AD 721-620
TM-25	Goldstein, R.C. Helping People Think, April 1971, AD 721-998
TM-26	Iazeolla, G.G. Modeling and Decomposition of Information Systems for Performance Evaluation, June 1971, AD 733-965
TM-27	Bagchi, A. Economy of Descriptions and Minimal Indices, January 1972, AD 736-960
TM-28	Wong, R. Construction Heuristics for Geometry and a Vector Algebra Representation of Geometry, June 1972, AD 743-487
TM-29	Hossley, R. and Rackoff, C. The Emptiness Problem for Automata on Infinite Trees, Spring 1972, AD 747-250
TM-30	McCray, W.A. SIM360: A S/360 Simulator, October 1972, AD 749-365
TM-31	Bonneau, R.J. A Class of Finite Computation Structures Supporting the Fast Fourier Transform, March 1973, AD 757-787
TM-32	Moll, R. An Operator Embedding Theorem for Complexity Classes of Recursive Functions, May 1973, AD 759-999

TM-33	Ferrante, J. and Rackoff, C.	A Decision Procedure for the First Order Theory of Real Addition with Order, May 1973, AD 760-000
TM-34	Bonneau, R.J.	Polynomial Exponentiation: The Fast Fourier Transform Revisited, June 1973, PB 221-742
TM-35	Bonneau, R.J.	An Interactive Implementation of the Todd-Coxeter Algorithm, December 1973, AD 770-565
TM-36	Geiger, S.P.	A User's Guide to the Macro Control Language, December 1973, AD 771-435
TM-37	Schonhage, A.	Real-Time Simulation of Multidimensional Turing Machines by Storage Modification Machines, December 1973, PB 226-103/AS
TM-38	Meyer, A.R.	Weak Monadic Second Order Theory of Successor is not Elementary-Recursive, December 1973, PB 226-514/AS
TM-39	Meyer, A.R.	Discrete Computation: Theory and Open Problems, January 1974, PB 226-836/AS
TM-40	Paterson, M.S., Fischer, M.J. and Meyer, A.R.	An Improved Overlap Argument for On-Line Multiplication, January 1974, AD 773-137
TM-41	Fischer, M.J. and Paterson, M.S.	String-Matching and Other Products, January 1974, AD 773-138
TM-42	Rackoff, C.	On the Complexity of the Theories of Weak Direct Products, January 1974, PB 228-459/AS

PUBLICATIONS

TM-43	Fischer, M.J. and Rabin, M.O. Super-Exponential Complexity of Presburger Arithmetic, February 1974, AD 775-004
TM-44	Pless, V. Symmetry Codes and their Invariant Subcodes, May 1974, AD 780-243
TM-45	Fischer, M.J. and Stockmeyer, L.J. Fast On-Line Integer Multiplication, May 1974, AD 779-889
TM-46	Kedem, Z.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, V. Mathematical Foundations of Flip-Flops, June 1974, AD 780-901
TM-48	Kedem, Z.M. The Reduction Method for Establishing Lower Bounds on the Number of Additions, June 1974, PB 233-538/AS
TM-49	Pless, V. Complete Classification of (24,12) and (22,11) Self-Dual Codes, June 1974, AD 781-335
TM-50	Benedict, G.G. An Enciphering Module for Multics, S.B. Thesis, EE Dept., July 1974, AD 782-658
TM-51	Aiello, J.M. An Investigation of Current Language Support for the Data Requirements of Structured Programming, S.M. & E.E. Thesis, EE Dept., September 1974, PB 236-815/AS
TM-52	Lind, J.C. Computing in Logarithmic Space, September 1974, PB 236-167/AS
TM-53	Bengelloun, S.A. MDC-Programmer: A Muddle-to Datalanguage Translator for Information Retrieval, S.B. Thesis, EE Dept., October 1974, AD 786-754
TM-54	Meyer, A.R. The Inherent Computation Complexity of Theories of Ordered Sets: A Brief Survey, October 1974, PB 237-200/AS

TM-55	Hsieh, W.N., Harper, L.H. and Savage, J.E. A Class of Boolean Functions with Linear Combinatorial Complexity, October 1974, PB 237-206/AS
TM-56	Gorry, G.A. Research on Expert Systems, December 1974
TM-57	Levin, M. On Bateson's Logical Levels of Learning, February 1975
TM-58	Qualitz, J.E. Decidability of Equivalence for a Class of Data Flow Schemas, March 1975, PB 237-033/AS
TM-59	Hack, M. Decision Problems for Petri Nets and Vector Addition Systems, March 1975 PB 231-916/AS
TM-60	Weiss, R.B. CAMAC: Group Manipulation System, March 1975, PB 240-495/AS
TM-61	Dennis, J.B. First Version of a Data Flow Procedure Language, May 1975
TM-62	Patil, S.S. An Asynchronous Logic Array, May 1975
TM-63	Pless, V. Encryption Schemes for Computer Confidentiality, May 1975, AD A010-217
TM-64	Weiss, R.B. Finding Isomorph Classes for Combinatorial Structures, S.M. Thesis, EE Dept., June 1975
TM-65	Fischer, M.J. The Complexity Negation-Limited Networks - A Brief Survey, June 1975

TM-66	Leung, C. Formal Properties of Well-Formed Data Flow Schemas, S.B., S.M. & E.E. Thesis, EE Dept., June 1975
TM-67	Cardoza, E.E. Computational Complexity of the Word Problem for Commutative Semigroups, S.M. Thesis, EE & CS Dept., October 1975
TM-68	Weng, K-S. Stream-Oriented Computation in Recursive Data Flow Schemas, S.M. Thesis, EE & CS Dept., October 1975
TM-69	Bayer, P.J. Improved Bounds on the Costs of Optimal and Balanced Binary Search Trees, S.M. Thesis, EE & CS Dept., November 1975
TM-70	Ruth, G.R. Automatic Design of Data Processing Systems, February 1976, AD A023-451
TM-71	Rivest, R. On the Worst-Case of Behavior of String-Searching Algorithms, April 1976
TM-72	Ruth, G.R. Protosystem I: An Automatic Programming System Prototype, July 1976, AD A026-912
TM-73	Rivest, R. Optimal Arrangement of Keys in a Hash Table, July 1976
TM-74	Malvania, N. The Design of a Modular Laboratory for Control Robotics, S.M. Thesis, EE & CS Dept., September 1976, AD A030-418
TM-75	Yao, A.C. and Rivest, R.I. K+1 Heads are Better than K, September 1976, AD A030-008
TM-76	Bloniarz, P.A., Fischer, M.J. and Meyer, A.R. A Note on the Average Time to Compute Transitive Closures, September 1976

TM-77	Mok, A.K. Task Scheduling in the Control Robotics Environment, S.M. Thesis, EE & CS Dept., September 1976, AD A030-402
TM-78	Benjamin, A.J. Improving Information Storage Reliability Using a Data Network, S.M. Thesis, EE & CS Dept., October 1976, AD A033-394
TM-79	Brown, G.P. A System to Process Dialogue: A Progress Report, October 1976, AD A033-276
TM-80	Even, S. The Max Flow Algorithm of Dinic and Karzanov: An Exposition, December 1976
TM-81	Gifford, D.K. Hardware Estimation of a Process' Primary Memory Requirements, S.B. Thesis, EE & CS Dept., January 1977
TM-82	Rivest, R.L., Shamir, A. and Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, April 1977, AD A039-036
TM-83	Baratz, A.E. Construction and Analysis of Network Flow Problem which Forces Karzanov Algorithm to $O(n^3)$ Running Time, April 1977
TM-84	Rivest, R.L. and Pratt, V.R. The Mutual Exclusion Problem for Unreliable Processes, April 1977
TM-85	Shamir, A. Finding Minimum Cutsets in Reducible Graphs, June 1977, AD A040-698
TM-86	Szolovits, P., Hawkinson, L.B. and Martin, W.A. An Overview of OWL, A Language for Knowledge Representation, June 1977, AD A041-372
TM-87	Clark, D., editor Ancillary Reports: Kernel Design Project, June 1977

PUBLICATIONS

TM-88	Lloyd, E.L. On Triangulations of a Set of Points in the Plane, S.M. Thesis, EE & CS Dept., July 1977
TM-89	Rodriguez, H. Jr. Measuring User Characteristics on the Multics System, S.B. Thesis, EE & CS Dept., August 1977
TM-90	d'Oliveira, C.R. An Analysis of Computer Decentralization, S.B. Thesis, EE & CS Dept., October 1977, AD A045-526
TM-91	Shamir, A. Factoring Numbers in O(log n) Arithmetic Steps, November 1977, AD A047-709
TM-92	Misunas, D.P. Report on the Workshop on Data Flow Computer and Program Organization, November 1977
TM-93	Amikura, K. A Logic Design for the Cell Block of a Data-Flow Processor, S.M. Thesis, EE & CS Dept., December 1977
TM-94	Berez, J.M. A Dynamic Debugging System for MDL, S.B. Thesis, EE & CS Dept., January 1978, AD A050-191
TM-95	Harel, D. Characterizing Second Order Logic with First Order Quantifiers, February 1978
TM-96	Harel, D., Amir P. and Stavi, J. A Complete Axiomatic System for Proving Deductions about Recursive Programs, February 1978
TM-97	Harel, D., Meyer, A.R. and Pratt, V.R. Computability and Completeness in Logics of Programs, February 1978

TM-98	Harel, D. and Pratt, V.R. Nondeterminism in Logics of Programs, February 1978
TM-99	LaPaugh, A.S. The Subgraph Homeomorphism Problem, S.M. Thesis, EE & CS Dept., February 1978
TM-100	Misunas, D.P. A Computer Architecture for Data-Flow Computation, S.M. Thesis, EE & CS Dept., March 1978, AD A052-538
TM-101	Martin, W.A. Descriptions and the Specialization of Concepts, March 1978, AD A052-773
TM-102	Abelson, H. Lower Bounds on Information Transfer in Distributed Computations, April 1978
TM-103	Harel, D. Arithmetical Completeness in Logics of Programs, April 1978
TM-104	Jaffe, J. The Use of Queues in the Parallel Data Flow Evaluation of "If-Then-While" Programs, May 1978
TM-105	Masek, W.J. and Paterson, M.S. A Faster Algorithm Computing String Edit Distances, May 1978
TM-106	Parikh, R. A Completeness Result for a Propositional Dynamic Logic, July 1978
TM-107	Shamir, A. A Fast Signature Scheme, July 1978, AD A057-152
TM-108	Baratz, A.E. An Analysis of the Solovay and Strassen Test for Primality, July 1978
TM-109	Parikh, R. Effectiveness, July 1978

PUBLICATIONS

TM-110	Jaffe, J.M. An Analysis of Preemptive Multiprocessor Job Scheduling, September 1978
TM-111	Jaffe, J.M. Bounds on the Scheduling of Typed Task Systems, September 1978
TM-112	Parikh, R. A Decidability Result for a Second Order Process Logic, September 1978
TM-113	Pratt, V.R. A Near-optimal Method for Reasoning about Action, September 1978
TM-114	Dennis, J.B., Fuller, S.H., Ackerman, W.B., Swan, R.J. and and Weng, K-S. Research Directions in Computer Architecture, September 1978, AD A061-222
TM-115	Bryant, R.E. and Dennis, J.B. Concurrent Programming, October 1978, AD A061-180
TM-116	Pratt, V.R. Applications of Modal Logic to Programming, December 1978
TM-117	Pratt, V.R. Six Lectures on Dynamic Logic, December 1978
TM-118	Borkin, S.A. Data Model Equivalence, December 1978, AD A062-753
TM-119	Shamir, A. and Zippel, R.E. On the Security of the Merkle-Hellman Cryptographic Scheme, December 1978, AD A063-104
TM-120	Brock, J.D. Operational Semantics of a Data Flow Language, S.M. Thesis, EE & CS Dept., December 1978, AD A062-997

TM-121	Jaffe, J. The Equivalence of R.E. Programs and Data Flow Schemes, January 1979
TM-122	Jaffe, J. Efficient Scheduling of Tasks Without Full Use of Processor Resources, January 1979
TM-123	Perry, H.M. An Improved Proof of the Rabin-Hartmanis-Stearns Conjecture, S.M. & E.E. Thesis, EE & CS Dept., January 1979
TM-124	Toffoli, T. Bicontinuous Extensions of Invertible Combinatorial Functions, January 1979, AD A063-886
TM-125	Shamir, A., Rivest, R.L. and Adleman, L.M. Mental Poker, February 1979, AD A066-331
TM-126	Meyer, A.R. and Paterson, M.S. With What Frequency Are Apparently Intractable Problems Difficult?, February 1979
TM-127	Strazdas, R.J. A Network Traffic Generator for Decnet, S.B. & S.M. Thesis, EE & CS Dept., March 1979
TM-128	Loui, M.C. Minimum Register Allocation is Complete in Polynomial Space, March 1979
TM-129	Shamir, A. On the Cryptocomplexity of Knapsack Systems, April 1979, AD A067-972
TM-130	Greif, I. and Meyer, A.R. Specifying the Semantics of While-Programs: A Tutorial and Critique of a Paper by Hoare and Lauer, April 1979, AD A068-967
TM-131	Adleman, L.M. Time, Space and Randomness, April 1979
TM-132	Patil, R.S. Design of a Program for Expert Diagnosis of Acid Base and Electrolyte Disturbances, May 1979

PUBLICATIONS

TM-133	Loui, M.C. The Space Complexity of Two Pebble Games on Trees, May 1979
TM-134	Shamir, A. How to Share a Secret, May 1979, AD A069-397
TM-135	Wyleczuk, R.H. Timestamps and Capability-Based Protection in a Distributed Computer Facility, S.B. & S.M. Thesis, EE & CS Dept., June 1979
TM-136	Misunas, D.P. Report on the Second Workshop on Data Flow Computer and Program Organization, June 1979
TM-137	Davis, E. and Jaffe, J.M. Algorithms for Scheduling Tasks on Unrelated Processors, June 1979
TM-138	Pratt, V.R. Dynamic Algebras: Examples, Constructions, Applications, July 1979
TM-139	Martin, W.A. Roles, Co-Descriptors, and the Formal Representation of Quantified English Expressions (Revised May 1980), September 1979, AD A074-625
TM-140	Szolovits, P. Artificial Intelligence and Clinical Problem Solving, September 1979
TM-141	Hammer, M. and McLeod, D. On Database Management System Architecture, October 1979, AD A076-417
TM-142	Lipski, W., Jr. On Data Bases with Incomplete Information, October 1979
TM-143	Leth, J.W. An Intermediate Form for Data Flow Programs, S.M. Thesis, EE & CS Dept., November 1979
TM-144	Takagi, A. Concurrent and Reliable Updates of Distributed Databases, November 1979

TM-145 Loui, M.C.
A Space Bound for One-Tape Multidimensional Turing Machines, November 1979

TM-146 Aoki, D.J.
A Machine Language Instruction Set for a Data Flow Processor, S.M. Thesis, EE & CS Dept., December 1979

TM-147 Schroeppel, R. and Shamir, A.
A $T = 0(2^{n/2})$, $S = 0(2^{n/4})$ Algorithm for Certain NP-Complete Problems, January 1980, AD A080-385

TM-148 Adleman, L.M. and Loui, M.C.
Space-Bounded Simulation of Multitape Turing Machines, January 1980

TM-149 Pallottino, S. and Toffoli, T.
An Efficient Algorithm for Determining the Length of the Longest Dead Path in an "Lifo" Branch-and-Bound Exploration Schema, January 1980, AD A079-912

TM-150 Meyer, A.R.
Ten Thousand and One Logics of Programming, February 1980

TM-151 Toffoli, T.
Reversible Computing, February 1980, AD A082-021

TM-152 Papadimitriou, C.H.
On the Complexity of Integer Programming, February 1980

TM-153 Papadimitriou, C.H.
Worst-Case and Probabilistic Analysis of a Geometric Location Problem, February 1980

TM-154 Karp, R.M. and Papadimitriou, C.H.
On Linear Characterizations of Combinatorial Optimization Problems, February 1980

TM-155 Atai, A., Lipton, R.J., Papadimitriou, C.H. and Rodeh, M.
Covering Graphs by Simple Circuits, February 1980

TM-156 Meyer, A.R. and Parikh, R.
Definability in Dynamic Logic, February 1980

TM-157	Meyer, A. R. and Winklmann, K. On the Expressive Power of Dynamic Logic, February 1980
TM-158	Stark, E. W. Semaphore Primitives and Starvation-Free Mutual Exclusion, S.M. Thesis, EE & CS Dept., March 1980
TM-159	Pratt, V.R. Dynamic Algebras and the Nature of Induction, March 1980
TM-160	Kanellakis, P. C. On the Computational Complexity of Cardinality Constraints in Relational Databases, March 1980
TM-161	Lloyd, E.L. Critical Path Scheduling of Task Systems with Resource and Processor Constraints, March 1980
TM-162	Marcum, A.M. A Manager for Named, Permanent Objects, S.B. & S.M. Thesis, EE & CS Dept., April 1980, AD A083-491
TM-163	Meyer, A. R. and Halpern, J.Y. Axiomatic Definitions of Programming Languages: A Theoretical Assessment, April 1980
TM-164	Shamir, A. The Cryptographic Security of Compact Knapsacks (Preliminary Report), April 1980, AD A084-456
TM-165	Finseth, C.A. Theory and Practice of Text Editors or A Cookbook for an Emacs, S.B. Thesis, EE & CS Dept., May 1980
TM-166	Bryant, R. E. Report on the Workshop on Self-Timed Systems, May 1980
TM-167	Pavelle, R. and Wester, M. Computer Programs for Research in Gravitation and Differential Geometry, June 1980
TM-168	Greif, I. Programs for Distributed Computing: The Calendar Application, July 1980, AD A087-357

TM-169	Burke, G. and Moon, D. LOOP Iteration Macro, (revised January 1981) July 1980, AD A087-372
TM-170	Ehrenfeucht, A., Parikh R. and Rozenberg, G. Pumping Lemmas for Regular Sets, August 1980
TM-171	Meyer, A.R. What is a Model of the Lambda Calculus?, August 1980
TM-172	Paseman, W. G. Some New Methods of Music Synthesis, S.M. Thesis, EE & CS Dept., August 1980, AD A090-130
TM-173	Hawkinson, L.B. XLMS: A Linguistic Memory System, September 1980, AD A090-033
TM-174	Arvind, Kathail V. and Pingali, K. A Dataflow Architecture with Tagged Tokens, September 1980
TM-175	Meyer, Albert R., Weise, D. and Loui, M.C. On Time Versus Space III, September 1980
TM-176	Seaquist, C.R. A Semantics of Synchronization, S.M. Thesis, EE & CS Dept., September 1980, AD A091-015
TM-177	Sinha, M. K. TIMEPAD - A Performance Improving Synchronization Mechanism for Distributed Systems, September 1980
TM-178	Arvind and Thomas, R.E. I-Structures: An Efficient Data Type for Functional Languages, September 1980
TM-179	Halpern J. Y. and Meyer, A.R. Axiomatic Definitions of Programming Languages, II, October 1980
TM-180	Papadimitriou, C. H. A Theorem in Database Concurrency Control, October 1980
TM-181	Lipski, W., Jr. and Papadimitriou, C.H. A Fast Algorithm for Testing for Safety and Detecting Deadlocks in Locked Transaction Systems, October 1980

TM-182	Itai, A., Papadimitriou C.H. and Szwarefiter, J.L. Hamilton Paths in Grid Graphs, October 1980
TM-183	Meyer, A. R. A Note on the Length of Craig's Interpolants, October 1980
TM-184	Lieberman, H. and Hewitt, C. A Real Time Garbage Collector that can Recover Temporary Storage Quickly, October 1980
TM-185	Kung, H-T. and Papadimitriou, C.H. An Optimality Theory of Concurrency Control for Databases, November 1980, AD A092-625
TM-186	Szolovits, P. and Martin, W.A. BRAND X Manual, November 1980, AD A093-041
TM-187	Fischer, M. J., Meyer, A.R. and Paterson, M.S. CapOmega()(n log n) Lower Bounds on Length of Boolean Formulas, November 1980
TM-188	Mayr, E. An Effective Representation of the Reachability Set of Persistent Petri Nets, January 1981
TM-189	Mayr, E. Persistence of Vector Replacement Systems is Decidable, January 1981
TM-190	Ben-Ari, M., Halpern, J.Y. and Pnueli, A. Deterministic Propositional Dynamic Logic: Finite Models, Complexity, and Completeness, January 1981.
TM-191	Parikh, R. Propositional Dynamic Logics of Programs: A Survey, January 198l.
TM-192	Meyer, A. R., Streett, R.S. and Mirkowska, G. The Deducibility Problem in Propositional Dynamic Logic, February 1981
TM-193	Yannakakis, M. and Papadimitriou, C.H. Algebraic Dependencies, February 1981

TM-194	Barendregt, H. and Longo, G. Recursion Theoretic Operators and Morphisms on Numbered Sets, February 1981
TM-195	Barber, G. R. Record of the Workshop on Research in Office Semantics, February 1981
TM-196	Bhatt, S.N. On Concentration and Connection Networks, S.M. Thesis, EE & CS Dept., March 1981
TM-197	Fredkin, E. and Toffoli, T. Conservative Logic, May 1981
TM-198	Halpern, J. and Reif, J. The Propositional Dynamic Logic of Deterministic Well-Structured Programs, March 1981
TM-199	Mayr, E. and Meyer, A. The Complexity of the Word Problems for Communative Semigroups and Polynomial Ideals, June 1981
TM-200	Burke, G. LSB Manual, June 1981
TM-201	Meyer, A. What is a Model of the lambda Calculus? Expanded Version, July 1981.
TM-202	Saltzer, J. H. Communication Ring Initialization without Central Control December 1981
TM-203	Bawden, A., Burke, G. and Hoffman, C. Maclisp Extensions, July 1981
TM-204	Halpern, J.Y. On the Expressive Power of Dynamic Logic, II, August 1981
TM-205	Kannon, R. Circuit-Size Lower Bounds and Non-Reducibility to Sparce Sets, October 1981.
TM-206	Leiserson, C. and Pinter, R. Optimal Placement for River Routing, October 1981

TM-207	Longo, G. Power Set Models For Lambda-Calculus: Theories, Expansions, Isomorphisms, November 1981
TM-208	Cosmadakis, S. and Papadimitriou, C. The Traveling Salesman Problem with Many Visits to Few Cities, November 1981
TM-209	Johnson, D. and Papadimitriou, C. Computational Complexity and the Traveling Salesman Problem, December 1981
TM-210	Greif, I. Software for the 'Roiels" People Play, February 1982
TM-211	Meyer, A. and Tiuryn, J. A Note on Equivalences Among Logics of Programs, December 1981
TM-212	Elias, P. Minimax Optimal Universal Codeword Sets, January 1982
TM-213	Greif, I PCAL: A Personal Calendar, January 1982
TM-214	Meyer, A. and Mitchell, J. Terminations for Recursive Programs: Completeness and Axiomatic Definability, March 1982
TM-215	Leiserson, C. and Saxe J. Optimizing Synchronous Systems, March 1982
TM-216	Church, K. and Patil, R. Coping with Syntactic Ambiguity or How to Put the Block in the Box on the Table, April 1982.
TM-217	Wright, D. A File Transfer Program for a Personal Computer, April 1982
TM-218	Greif, I. Cooperative Office Work, Teleconferencing and Calendar Management: A Collection of Papers, May 1982
TM-219	Jouannaud, J.-P., Lescanne, P and Reinig, F. Recursive Decomposition Ordering and Multiset Orderings, June 1982
TM-220	Chu, T.-A. Circuit Analysis of Self-Times Elements for NMOS VLSI Systems, May 1982

TM-221	Leighton, F., Lepley, M. and Miller, G. Layouts for the Shuffle-Exchange Graph Based on the Complex Plane Diagram, June 1982
TM-222	Meier zu Sieker, F. A Telex Gateway for the Internety, S.B. Thesis, Electrical Engineering Dept., May 1982
TM-223	diSessa, A.A. A Principled Design for an Integrated Computation Environment, July 1982
TM-224	Barber, G. Supporting Organizational Problem Solving with a Workstation, July 1982
TM-225	Barber, G. and Hewitt, C. Foundations for Office Semantics, July 1982
TM-226	Bergstra, J., Chmielinska, A. and Tiuryn, J. Hoares's Logic Not Complete When it Could Be, August 1982
TM-227	Leighton, F.T. New Lower Bound Techniques for VLSI, August 1982
TM-228	Papadimitriou, C. and Zachos, S. Two Remarks on the Power of Counting, August 1982
TM-229	Cosmadakis, S. The Complexity of Evaluation Relational Queries, August 1982
TM-230	Shamir, A. Embedding Cryptographic Trapdoors in Arbitrary Knapsack Systems, September 1982
TM-231	Kleitman, D., Leighton, F.T., Lepley, M. and Miller G. An Asymptotically Optimal Layout for the shuffle-exchange Graph, October 1982
TM-232	Yeh, A. PLY: A System of Plausibility Inference with a Probabilistic Basis, December 1982
TM-233	Konopelski, L. Implementing Internet Remote Login on a Personal Computer, S.B. Thesis, Electrical Engineering Dept., December 1982

TM-234	Rivest, R. and Sherman, A. Randomized Encryption Techniques, January 1983.
TM-235	Mitchell, J. The Implication of Problem for Functional and Inclusion Dependencies, February 1983
TM-236	Leighton, F.T. and Leiserson, C.E. Wafter-Scale Integration of Systolic Arrays, February 1983
TM-237	Dolev, D., Leighton, F.T. and Trickey, H. Planar Embedding of Planar Graphs, February 1983
TM-238	Baker, B.S., Bhatt, S.N. and Leighton, F.T. An Approximation Algorithm for Manhattan Routing, February 1983
TM-239	Sutherland, J.B. and Sirbu, M. Evaluation of an Office Analysis Methodology, March 1983
TM-240	Bromley, H. A Program for Therapy of Acid-Base and Electrolyte Disorders, S.B. Thesis, Electrical Engineering Dept., June 1983
TM-241	Arvind and Iannucci, R.A. Two Fundamental Issues in Multiprocessing: The Dataflow Solution, September 1983
TM-242	Pingali, K. and Arvind. Efficient Demand-driven Evaluation (I), September 1983
TM-243	Pingali, K. and Arvind. Efficient Demand-driven Evaluation (II), September 1983
TM-244	Goldreich, O., Goldwasser, S. and Micali, S. How to Construct Random Functions, November 1983
TM-245	Meyer, A. Understanding Algol: The View of the Recent Convert to Denotational Semantics, October 1983
TM-246	Trakhtenbrot, B.A., Halpern, J.Y. and Meyer, A.R. From Denotational to Operational and Axiomatic Semantics for Algol-Like Languages: An Overview, October 1983
TM-247	Leighton, T. and Lepley, M. Probabilistic Searching in Sorted Linked Lists, November 1983

TM-248	Leighton, F.T. and Rivest, R.L. Estimating a Probability Using Finite Memory, November 1983
TM-249	Leighton, F.T. and Rivest, R.L. The Markov Chain Tree Theorem, December 1983
TM-250	Goldreich, O. On Concurrent Identification Protocols, December 1983
TM-251	Dolev, D., Lynch, N., Pinter, S. Stark, E. and Weihl, W. Reaching Approximate Agreement in the Presence of Faults, December 1983
TM-252	Zachos, S. and Heller, H. On BPP, December 1983
TM-253	Chor, B., Leiserson, C., Rivest, R. and Shearer, J. An Application of Number Theory to the Organization of Raster Graphics Memory, April 1984
TM-254	Feldmeier, D.C. Empirical Analysis of a Token Ring Network, April 1984
TM-255	Bhatt, S. and Leiserson, C. How to Assemble Tree Machines, April 1984
TM-256	Goldreich, O. On the Number of Close-and Equal Pairs of Bits in a String (With Implications on the Security of RSA's L.S.B., April 1984
TM-257	Dwork, C., Kanellakis, P. and Mitchell, J. On the Sequential Nature of Unification, April 1984
TM-258	Halpern, M., Meyer A. and Trakhtenbrot, B. The Semantics of Local Storage, or What Makes the Free-list Free?, April 1984
TM-259	Lynch, N. and Fredrickson, G. The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring, April 1984
TM-260	Chor, B. and Goldreich, O. RSA/Rabin Least Significant Bits are 1/2 + poly(logn) Secure, May 1984
TM-261	Zaks, S. Optimal Distributed Algorithms for Sorting and Ranking, May 1984

PUBLICATIONS

TM-262	Leighton, T. and Rosenberg, A. Three-dimensional Circuit Layouts, June 1984
TM-263	Sirbu, M.S. and Sutherland, J.B. Naming and Directory Issues in Message Transfer Systems, July 1984
TM-264	Sarin, S.K. and Greif, I. Software for Interactive On-Line Conferences, July 1984
TM-265	Lundelius, J. and Lynch, N. A New Fault-Tolerant Algorithm for Clock Synchronization, July 1984
TM-266	Chor, B. and Coan, B.A. A Simple and Efficient Randomized Byzantine Agreement Algorithm, August 1984
TM-267	Schooler, R. and Stamos, J.W. Proposal for a Small Scheme Implementation, October 1984
TM-268	Awerbuch, B. Complexity of Network Synchronization, January 1985
TM-269	Fisher, M., Lynch, N.A., Burns, J. and Borodin, A. The Colored Ticket Algorithm, August 1983
TM-270	Dwork, C., Lynch, C. and Stockmeyer, L. Consensus in the Presence of Partial Synchrony (Preliminary Version), July 1984
TM-271	Dershowitz, N. and Zaks, S. Patterns in Trees, January 1985
TM-272	Leighton, T. Tight Bounds on the Complexity of Parallel Sorting, April 1985
TM-273	Berman, F., Leighton, T., Shor, P.W. and Shor, L. Generalized Planar Matching, April 1985
TM-274	Kuipers, B. Qualitative Simulation of Mechanisms, April 1985
TM-275	Burns, J.E. and Lynch, N.A. The Byzantine Firing Squad Problem, April 1985
TM-276	Dolev, D., Lynch., N.A., Pinter, S.S., Stark, E.W. and Weihl, W.E. Reaching Approximate Agreement in the Presence of Faults, May 1985

TM-277	Frederickson, G.N. and Lynch, N.A. A General Lower Bound for Electing a Leader in a Ring, March 1985
TM-278	Fisch, M.J., Griffeth, N.D., Guibas, L.J. and Lynch, N.A. Probabilistic Analysis of a Network Resource Allocation Algorithm, June 1985
TM-279	Fischer, M.J., Lynch, N.A. and Merritt, M. Easy Impossibility Proofs for Distributed Consensus Problems, June 1985
TM-280	Kuipers, B. and Kassirer, J.P. Qualitative Simulation in Medical Physiology: A Progress Report, June 1985
TM-281	Hailperin, M. What Price for Eliminating Expression Side-Effects?, June 1985
TM-285	Kilian, J.J Two Undecidability Results in Probabilistic Automata Theory, S.B. Thesis/June 1985
TM-288	Chung, J.C. Dscribe: A Scribe Server, May 1985
TM-290	Fisher, M.J., Lynch, N.A., Burns, J.E. and Borodin, A. Distributed FIFO Allocation of Identical Resources Using Small Shared Space, June 1985

Technical Reports

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

[4]TRs 5, 9, 10, 15 were never issued

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. Dissertation, EE Dept., June 1965, AD 470-715
TR-19	Russo, Francis John A Heuristic Approach to Alternate Routing in a Job Shop, S.B. & S.M. Thesis, Sloan School, June 1965, AD 474-018
TR-20	Wantman, Mayer Elihu CALCULAID: An On-Line System for Algebraic Computation and Analysis, S.M. Thesis, Sloan School, September 1965, AD 474-019
TR-21	Denning, Peter James Queueing Models for File Memory Operation, S.M. Thesis, EE Dept., October 1965, AD 624-943
TR-22	Greenberger, Martin The Priority Problem, November 1965, AD 625-728
TR-23	Dennis, Jack B. and Earl C. Van Horn Programming Semantics for Multi-programmed Computations, December 1965, AD 627-537
TR-24	Kaplow, Roy, Stephen Strong and John Brackett MAP: A System for On-Line Mathematical Analysis, January 1966, AD 476-443
TR-25	Stratton, William David Investigation of an Analog Technique to Decrease Pen-Tracking Time in Computer Displays, S.M. Thesis, EE Dept., March 1966, AD 631-396

PUBLICATIONS

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. Dissertation, EE Dept., June 1966, AD 636-275
TR-30	Saltzer, Jerome Howard TRaffic Control in a Multiplexed Computer System, Sc.D. Thesis, EE Dept., July 1966, AD 635-966
TR-31	Smith, Donald L. Models and Data Structures for Digital Logic Simulation, S.M. Thesis, EE Dept., August 1966, AD 637-192
TR-32	Teitelman, Warren PILOT: A Step Toward Man-Computer Symbiosis, Ph.D. Dissertation, Math. Dept., September 1966, AD 638-446
TR-33	Norton, Lewis M, ADEPT - A Heuristic Program for Proving Theorems of Group Theory, Ph.D. Dissertation, Math. Dept., October 1966, AD 645-660
TR-34	Van Horn, Earl C., Jr. Computer Design for Asynchronously Reproducible Multiprocessing, Ph.D. Dissertation, EE Dept., November 1966, AD 650-407
TR-35	Fenichel, Robert R. An On-Line System for Algebraic Manipulation, Ph.D. Dissertation, Appl. Math. (Harvard), December 1966, AD 657-282
TR-36	Martin, William A. Symbolic Mathematical Laboratory, Ph.D. Dissertation, EE Dept., January 1967, AD 657-283
TR-37	Guzman-Arenas, Adolfo Some Aspects of Pattern Recognition by Computer, S.M. Thesis, EE Dept., February 1967, AD 656-041

TR-38	Rosenberg, Ronald C., Daniel W. Kennedy and Roger A. Humphrey A Low-Cost Output Terminal For Time-Shared Computers, March 1967, AD 662-027
TR-39	Forte, Allen Syntax-Based Analytic Reading of Musical Scores, April 1967, AD 661-806
TR-40	Miller, James R. On-Line Analysis for Social Scientists, May 1967, AD 668-009
TR-41	Coons, Steven A. Surfaces for Computer-Aided Design of Space Forms, June 1967, AD 663-504
TR-42	Liu, Chung L., Gabriel D. Chang and Richard E. Marks Design and Implementation of a Table-Driven Compiler System, July 1967, AD 668-960
TR-43	Wilde, Daniel U. Program Analysis by Digital Computer, Ph.D. Dissertation, EE Dept., August 1967, AD 662-224
TR-44	Gorry, G. Anthony A System for Computer-Aided Diagnosis, Ph.D. Dissertation, 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. Dissertation, Math. Dept., December 1967, AD 662-666
TR-48	Jones, Malcolm M. Incremental Simulation on a Time-Shared Computer, Ph.D. Dissertation, Sloan School, January 1968, AD 662-225
TR-49	Luconi, Fred L. Asynchronous Computational Structures, Ph.D. Dissertation, EE Dept., February 1968, AD 667-602

TR-50	Denning, Peter J. Resource Allocation in Multiprocess Computer Systems, Ph.D. Dissertation, 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. Dissertation, 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 An Integrated Hardware-Software System for Computer Graphics in Time-Sharing, December 1968, AD 685-202
TR-57	Morris, James H. Lambda-Calculus Models of Programming Languages, Ph.D. Dissertation, 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. Dissertation, 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. Dissertation, EE Dept., April 1969, AD 689-305
TR-61	Baecker, Ronald M. Interactive Computer-Mediated Animation, Ph.D. Dissertation, EE Dept., June 1969, AD 690-887
TR-62	Tillman, Coyt C., Jr. 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. 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. Dissertation, EE Dept., October 1969, AD 699-501
TR-66	Beyer, Wendell T. Recognition of Topological Invariants by Iterative Arrays, Ph.D. Dissertation, Math. Dept., October 1969, AD 699-502

TR-67	Vanderbilt, Dean H. Controlled Information Sharing in a Computer Utility, Ph.D. Dissertation, 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. Dissertation, Sloan School, June 1970, AD 710-011
TR-69	Gertz, Jeffrey L. Hierarchical Associative Memories for Parallel Computation, Ph.D. Dissertation, EE Dept., June 1970, AD 711-091
TR-70	Fillat, Andrew I. and Leslie A. Kraning Generalized Organization of Large Data-Bases: A Set-Theoretic Approach to Relations, S.B. & S.M. Thesis, EE Dept., June 1970, AD 711-060
TR-71	Fiasconaro, James G. A Computer-Controlled Graphical Display Processor, S.M. Thesis, EE Dept., June 1970, AD 710-479
TR-72	Patil, Suhas S. Coordination of Asynchronous Events, Sc.D. Thesis, EE Dept., June 1970, AD 711-763
TR-73	Griffith, Arnold K. Computer Recognition of Prismatic Solids, Ph.D. Dissertation, Math. Dept., August 1970, AD 712-069
TR-74	Edelberg, Murray Integral Convex Polyhedra and an Approach to Integralization, Ph.D. Dissertation, EE Dept., August 1970, AD 712-070
TR-75	Hebalkar, Prakash G. Deadlock-Free Sharing of Resources in Asynchronous Systems, Sc.D. Thesis, EE Dept., September 1970, AD 713-139
TR-76	Winston, Patrick H. Learning Structural Descriptions from Examples, Ph.D. Dissertation, EE Dept., September 1970, AD 713-988
TR-77	Haggerty, Joseph P. Complexity Measures for Language Recognition by Canonic Systems, S.M. Thesis, EE Dept., October 1970, AD 715-134

TR-78	Madnick, Stuart E. Design Strategies for File Systems, S.M. Thesis, EE Dept. & Sloan School, October 1970, AD 714-269
TR-79	Horn, Berthold K. Shape from Shading: A Method for Obtaining the Shape of a Smooth Opaque Object from One View, Ph.D. Dissertation, EE Dept., November 1970, AD 717-336
TR-80	Clark, David D., Robert M. Graham, Jerome H. Saltzer and Michael D. Schroeder The Classroom Information and Computing Service, January 1971, AD 717-857
TR-81	Banks, Edwin R. Information Processing and Transmission in Cellular Automata, Ph.D. Dissertation, ME Dept., January 1971, AD 717-951
TR-82	Krakauer, Lawrence J. Computer Analysis of Visual Properties of Curved Objects, Ph.D. Dissertation, EE Dept., May 1971, AD 723-647
TR-83	Lewin, Donald E. In-Process Manufacturing Quality Control, Ph.D. Dissertation, 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. Dissertation, 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. Dissertation, EE Dept., June 1971, AD 725-859
TR-87	Thomas, Robert H. A Model for Process Representation and Synthesis, Ph.D. Dissertation, EE Dept., June 1971, AD 726-049

TR-88	Welch, Terry A. Bounds on Information Retrieval Efficiency in Static File Structures, Ph.D. Dissertation, 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. Thesis, EE Dept., September 1971, AD 730-521
TR-91	Smoliar, Stephen W. A Parallel Processing Model of Musical Structures, Ph.D. Dissertation, Math. Dept., September 1971, AD 731-690
TR-92	Wang, Paul S. Evaluation of Definite Integrals by Symbolic Manipulation, Ph.D. Dissertation, 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. Dissertation), 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. Dissertation, EE Dept., March 1972, AD 739-678
TR-98	Stockmeyer, Larry Joseph Bounds on Polynomial Evaluation Algorithms, S.M. Thesis, EE Dept., April 1972, AD 740-328

TR-99	Lynch, Nancy Ann Relativization of the Theory of Computational Complexity, Ph.D. Dissertation, 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. Dissertation, EE Dept., September 1972, AD 749-949
TR-104	Schroeder, Michael D. Cooperation of Mutually Suspicious Subsystems in a Computer Utility, Ph.D. Dissertation, EE Dept., September 1972, AD 750-173
TR-105	Smith, Burton J. An Analysis of Sorting Networks, Sc.D. Thesis, EE Dept., October 1972, AD 751-614
TR-106	Rackoff, Charles W. The Emptiness and Complementation Problems for Automata on Infinite Trees, S.M. Thesis, EE Dept., January 1973, AD 756<-248
TR-107	Madnick, Stuart E. Storage Hierarchy Systems, Ph.D. Dissertation, EE Dept., April 1973, AD 760-001
TR-108	Wand, Mitchell Mathematical Foundations of Formal Language Theory, Ph.D. Dissertation, Math. Dept., December 1973.
TR-109	Johnson, David S. Near-Optimal Bin Packing Algorithms, Ph.D. Dissertation, Math. Dept., June 1973, PB 222-090

PUBLICATIONS

TR-110	Moll, Robert Complexity Classes of Recursive Functions, Ph.D. Dissertation, Math. Dept., June 1973, AD 767-730
TR-111	Linderman, John P. Productivity in Parallel Computation Schemata, Ph.D. Dissertation, EE Dept., December 1973, PB 226-159/AS
TR-112	Hawryszkiewycz, Igor T. Semantics of Data Base Systems, Ph.D. Dissertation, EE Dept., December 1973, PB 226-061/AS
TR-113	Herrmann, Paul P. On Reducibility Among Combinatorial Problems, S.M. Thesis, Math. Dept., December 1973, PB 226-157/AS
TR-114	Metcalfe, Robert M. Packet Communication, Ph.D. Dissertation, Applied Math., Harvard University, December 1973, AD 771-430
TR-115	Rotenberg, Leo Making Computers Keep Secrets, Ph.D. Dissertation, 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. Thesis, EE Dept., January 1974, AD 774-141
TR-117	Clark, David D. An Input/Output Architecture for Virtual Memory Computer Systems, Ph.D. Dissertation, 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. Dissertation, EE Dept., February 1974, AD 775-545
TR-120	Ramchandani, Chander Analysis of Asynchronous Concurrent Systems by Timed Petri Nets, Ph.D. Dissertation, EE Dept., February 1974, AD 775-618

TR-121	Yao, Foong F. On Lower Bounds for Selection Problems, Ph.D. Dissertation, 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. Thesis, EE Dept., March 1974, PB 231-365/AS
TR-125	Mark, William S. A Model-Debugging System, S.B. & S.M. Thesis, EE Dept., April 1974, AD 778-688
TR-126	Altman, Vernon E. A Language Implementation System, S.B. & S.M. Thesis, 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. Thesis, 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. Dissertation, 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. Dissertation, EE Dept., July 1974, PB 235-283/AS
TR-134	Ellis, David J. Semantics of Data Structures and References, S.M. & E.E. Thesis, EE Dept., August 1974, PB 236-594/AS
TR-135	Pfister, Gregory F. The Computer Control of Changing Pictures, Ph.D. Dissertation, EE Dept., September 1974, AD 787-795
TR-136	Ward, Stephen A. Functional Domains of Applicative Languages, Ph.D. Dissertation, EE Dept., September 1974, AD 787-796
TR-137	Seiferas, Joel I. Nondeterministic Time and Space Complexity Classes, Ph.D. Dissertation, Math. Dept., September 1974. PB 236-777/AS
TR-138	Yun, David Y. Y. The Hensel Lemma in Algebraic Manipulation, Ph.D. Dissertation, Math. Dept., November 1974, AD A002-737
TR-139	Ferrante, Jeanne Some Upper and Lower Bounds on Decision Procedures in Logic, Ph.D. Dissertation, Math. Dept., November 1974. PB 238-121/AS
TR-140	Redell, David D. Naming and Protection in Extendable Operating Systems, Ph.D. Dissertation, EE Dept., November 1974, AD A001-721
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. Thesis, 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. Dissertation, EE Dept., February 1975.
TR-145	Henderson, D. Austin The Binding Model: A Semantic Base for Modular Programming Systems, Ph.D. Dissertation, 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. Dissertation, 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.
TR-148	Johnson, Jerry Program Restructuring for Virtual Memory Systems, Ph.D. Dissertation, EE Dept., March 1975, AD A009-218
TR-149	Snyder, Alan A Portable Compiler for the Language C, S.B. & S.M. Thesis, EE Dept., May 1975, AD A010-218
TR-150	Rumbaugh, James E. A Parallel Asynchronous Computer Architecture for Data Flow Programs, Ph.D. Dissertation, EE Dept., May 1975, AD A010-918
TR-151	Manning, Frank B. Automatic Test, Configuration, and Repair of Cellular Arrays, Ph.D. Dissertation, EE Dept., June 1975, AD A012-822
TR-152	Qualitz, Joseph E. Equivalence Problems for Monadic Schemas, Ph.D. Dissertation, 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. Dissertation, EE & CS Dept., September 1975, AD A016-302
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. Dissertation, EE & CS Dept., November 1975, AD A018-997
TR-158	Grossman, Richard W. Some Data-base Applications of Constraint Expressions, S.M. Thesis, EE & CS Dept., February 1976, AD A024-149
TR-159	Hack, Michel Petri Net Languages, March 1976.
TR-160	Bosyj, Michael A Program for the Design of Procurement Systems, S.M. Thesis, EE & CS Dept., May 1976, AD A026-688
TR-161	Hack, Michel Decidability Questions, Ph.D. Dissertation, EE & CS Dept., June 1976.
TR-162	Kent, Stephen T. Encryption-Based Protection Protocols for Interactive User-Computer Communication, S.M. Thesis, EE & CS Dept., June 1976, AD A026-911
TR-163	Montgomery, Warren A. A Secure and Flexible Model of Process Initiation for a Computer Utility, S.M. & E.E. Thesis, EE & CS Dept., June 1976.
TR-164	Reed, David P. Processor Multiplexing in a Layered Operating System, S.M. Thesis, EE & CS Dept., July 1976.

TR-165	McLeod, Dennis J. High Level Expression of Semantic Integrity Specifications in a Relational Data Base System, S.M. Thesis, EE & CS Dept., September 1976, AD A034-184
TR-166	Chan, Arvola Y. Index Selection in a Self-Adaptive Relational Data Base Management System, S.M. Thesis, EE & CS Dept., September 1976, AD A034-185
TR-167	Janson, Philippe A. Using Type Extension to Organize Virtual Memory Mechanisms, Ph.D. Dissertation, EE & CS Dept., September 1976.
TR-168	Pratt, Vaughan R. Semantical Considerations on Floyd-Hoare Logic, September 1976.
TR-169	Safran, Charles, James F. Desforges and Philip N. Tsichlis Diagnostic Planning and Cancer Management, September 1976.
TR-170	Furtek, Frederick C. The Logic of Systems, Ph.D. Dissertation, EE & CS Dept., December 1976.
TR-171	Huber, Andrew R. A Multi-Process Design of a Paging System, S.M. & E.E. Thesis, EE & CS Dept., December 1976.
TR-172	Mark, William S. The Reformulation Model of Expertise, Ph.D. Dissertation, EE & CS Dept., December 1976, AD A035-397
TR-173	Goodman, Nathan Coordination of Parallel Processes in the Actor Model of Computation, S.M. Thesis, EE & CS Dept., December 1976.
TR-174	Hunt, Douglas H. A Case Study of Intermodule Dependencies in a Virtual Memory Subsystem, S.M. & E.E. Thesis, EE & CS Dept., December 1976.
TR-175	Goldberg, Harold J. A Robust Environment for Program Development, S.M. Thesis, EE & CS Dept., February 1977.
TR-176	Swartout, William R. A Digitalis Therapy Advisor with Explanations, S.M. Thesis, EE & CS Dept., February 1977.

TR-177	Mason, Andrew H. A Layered Virtual Memory Manager, S.M. & E.E. Thesis, EE & CS Dept., May 1977.
TR-178	Bishop, Peter B. Computer Systems with a Very Large Address Space and Garbage Collection, Ph.D. Dissertation, EE & CS Dept., May 1977, AD A040-601
TR-179	Karger, Paul A. Non-Discretionary Access Control for Decentralized Computing Systems, S.M. Thesis, EE & CS Dept., May 1977, AD A040-804
TR-180	Luniewski, Allen W. A Simple and Flexible System Initialization Mechanism, S.M. & E.E. Thesis, EE & CS Dept., May 1977.
TR-181	Mayr, Ernst W. The Complexity of the Finite Containment Problem for Petri Nets, S.M. Thesis, EE & CS Dept., June 1977 .
TR-182	Brown, Gretchen P. A Framework for Processing Dialogue, June 1977, AD A042-370
TR-183	Jaffe, Jeffrey M. Semilinear Sets and Applications, S.M. Thesis, EE & CS Dept., July 1977.
TR-184	Levine, Paul H. Facilitating Interprocess Communication in a Heterogeneous Network Environment, S.B. & S.M. Thesis, EE & CS Dept., July 1977, AD A043-901
TR-185	Goldman, Barry Deadlock Detection in Computer Networks, S.B. & S.M. Thesis, EE & CS Dept., September 1977, AD A047-025
TR-186	Ackerman, William B. A Structure Memory for Data Flow Computers, S.M. Thesis, EE & CS Dept., September 1977, AD A047-026
TR-187	Long, William J. A Program Writer, Ph.D. Dissertation, EE & CS Dept., November 1977, AD A047-595
TR-188	Bryant, Randal E.

Simulation of Packet Communication Architecture Computer Systems, S.M. Thesis, EE & CS Dept., November 1977, AD A048-290

TR-189 Ellis, David J.
Formal Specifications for Packet Communication Systems, Ph.D. Dissertation, EE & CS Dept., November 1977, AD A048-980

TR-190 Moss, J. Eliot B.
Abstract Data Types in Stack Based Languages, S.M. Thesis, EE & CS Dept., February 1978, AD A052-332

TR-191 Yonezawa, Akinori
Specification and Verification Techniques for Parallel Programs Based on Message Passing Semantics, Ph.D. Dissertation, EE & CS Dept., January 1978, AD A051-149

TR-192 Niamir, Bahram
Attribute Partitioning in a Self-Adaptive Relational Database System, S.M. Thesis, EE & CS Dept., January 1978, AD A053-292

TR-193 Schaffert, J. Craig
A Formal Definition of CLU, S.M. Thesis, EE & CS Dept., January 1978

TR-194 Hewitt, Carl and Henry Baker, Jr.
Actors and Continuous Functionals, February 1978, AD A052-266

TR-195 Bruss, Anna R.
On Time-Space Classes and Their Relation to the Theory of Real Addition, S.M. Thesis, EE & CS Dept., March 1978

TR-196 Schroeder, Michael D., David D. Clark, Jerome H. Saltzer and Douglas H. Wells
Final Report of the Multics Kernel Design Project, March 1978

TR-197 Baker, Henry Jr.
Actor Systems for Real-Time Computation, Ph.D. Dissertation, EE & CS Dept., March 1978, AD A053-328

TR-198 Halstead, Robert H., Jr.
Multiple-Processor Implementation of Message-Passing Systems, S.M. Thesis, EE & CS Dept., April 1978, AD A054-009

TR-199 Terman, Christopher J.
The Specification of Code Generation Algorithms, S.M. Thesis, EE & CS Dept., April 1978, AD A054-301

PUBLICATIONS

TR-200	Harel, David Logics of Programs: Axiomatics and Descriptive Power, Ph.D. Dissertation, EE & CS Dept., May 1978
TR-201	Scheifler, Robert W. A Denotational Semantics of CLU, S.M. Thesis, EE & CS Dept., June 1978
TR-202	Principato, Robert N., Jr. A Formalization of the State Machine Specification Technique, S.M. & E.E. Thesis, EE & CS Dept., July 1978
TR-203	Laventhal, Mark S. Synthesis of Synchronization Code for Data Abstractions, Ph.D. Dissertation, EE & CS Dept., July 1978, AD A058-232
TR-204	Teixeira, Thomas J. Real-Time Control Structures for Block Diagram Schemata, S.M. Thesis, EE & CS Dept., August 1978, AD A061-122
TR-205	Reed, David P. Naming and Synchronization in a Decentralized Computer System, Ph.D. Dissertation, EE & CS Dept., October 1978, AD A061-407
TR-206	Borkin, Sheldon A. Equivalence Properties of Semantic Data Models for Database Systems, Ph.D. Dissertation, EE & CS Dept., January 1979, AD A066-386
TR-207	Montgomery, Warren A. Robust Concurrency Control for a Distributed Information System, Ph.D. Dissertation, EE & CS Dept., January 1979, AD A066-996

TR-208	Krizan, Brock C. A Minicomputer Network Simulation System, S.B. & S.M. Thesis, EE & CS Dept., February 1979
TR-209	Snyder, Alan A Machine Architecture to Support an Object-Oriented Language, Ph.D. Dissertation, EE & CS Dept., March 1979, AD A068-111
TR-210	Papadimitriou, Christos H. Serializability of Concurrent Database Updates, March 1979
TR-211	Bloom, Toby Synchronization Mechanisms for Modular Programming Languages, S.M. Thesis, EE & CS Dept., April 1979, AD A069-819
TR-212	Rabin, Michael O. Digitalized Signatures and Public-Key Functions as Intractable as Factorization, March 1979
TR-213	Rabin, Michael O. Probabilistic Algorithms in Finite Fields, March 1979
TR-214	McLeod, Dennis A Semantic Data Base Model and Its Associated Structured User Interface, Ph.D. Dissertation, EE & CS Dept., March 1979, AD A068-112
TR-215	Svobodova, Liba, Barbara Liskov and David Clark Distributed Computer Systems: Structure and Semantics, April 1979, AD A070-286
TR-216	Myers, John M. Analysis of the SIMPLE Code for Dataflow Computation, June 1979
TR-217	Brown, Donna J. Storage and Access Costs for Implementations of Variable - Length Lists, Ph.D. Dissertation, EE & CS Dept., June 1979
TR-218	Ackerman, William B. and Jack B. Dennis VAL--A Value-Oriented Algorithmic Language: Preliminary Reference Manual, June 1979, AD A072-394

TR-219	Sollins, Karen R. Copying Complex Structures in a Distributed System, S.M. Thesis, EE & CS Dept., July 1979, AD A072-441
TR-220	Kosinski, Paul R. Denotational Semantics of Determinate and Non-Determinate Data Flow Programs, Ph.D. Dissertation, EE & CS Dept., July 1979
TR-221	Berzins, Valdis A. Abstract Model Specifications for Data Abstractions, Ph.D. Dissertation, EE & CS Dept., July 1979
TR-222	Halstead, Robert H., Jr. Reference Tree Networks: Virtual Machine and Implementation, Ph.D. Dissertation, EE & CS Dept., September 1979, AD A076-570
TR-223	Brown, Gretchen P. Toward a Computational Theory of Indirect Speech Acts, October 1979, AD A077-065
TR-224	Isaman, David L. Data-Structuring Operations in Concurrent Computations, Ph.D. Dissertation, EE & CS Dept., October 1979
TR-225	Liskov, Barbara, Russ Atkinson, Toby Bloom, Eliot Moss, Craig Schaffert, Bob Scheifler and Alan Snyder CLU Reference Manual, October 1979, AD A077-018
TR-226	Reuveni, Asher The Event Based Language and Its Multiple Processor Implementations, Ph.D. Dissertation, EE & CS Dept., January 1980, AD A081-950
TR-227	Rosenberg, Ronni L. Incomprehensible Computer Systems: Knowledge Without Wisdom, S.M. Thesis, EE & CS Dept., January 1980
TR-228	Weng, Kung-Song An Abstract Implementation for a Generalized Data Flow Language, Ph.D. Dissertation, EE & CS Dept., January 1980

TR-229	Atkinson, Russell R. Automatic Verification of Serializers, Ph.D. Dissertation, EE & CS Dept., March 1980, AD A082-885
TR-230	Baratz, Alan E. The Complexity of the Maximum Network Flow Problem, S.M. Thesis, EE & CS Dept., March 1980
TR-231	Jaffe, Jeffrey M. Parallel Computation: Synchronization, Scheduling, and Schemes, Ph.D. Dissertation, EE & CS Dept., March 1980
TR-232	Luniewski, Allen W. The Architecture of an Object Based Personal Computer, Ph.D. Dissertation, EE & CS Dept., March 1980, AD A083-433
TR-233	Kaiser, Gail E. Automatic Extension of an Augmented Transition Network Grammar for Morse Code Conversations, S.B. Thesis, EE & CS Dept., April 1980, AD A084-411
TR-234	Herlihy, Maurice P. TRansmitting Abstract Values in Messages, S.M. Thesis, EE & CS Dept., May 1980, AD A086-984
TR-235	Levin, Leonid A. A Concept of Independence with Applications in Various Fields of Mathematics, May 1980
TR-236	Lloyd, Errol L. Scheduling Task Systems with Resources, Ph.D. Dissertation, EE & CS Dept., May 1980
TR-237	Kapur, Deepak Towards a Theory for Abstract Data Types, Ph.D. Dissertation, EE & CS Dept., June 1980, AD A085-877
TR-238	Bloniarz, Peter A. The Complexity of Monotone Boolean Functions and an Algorithm for Finding Shortest Paths in a Graph, Ph.D. Dissertation, EE & CS Dept., June 1980
TR-239	Baker, Clark M. Artwork Analysis Tools for VLSI Circuits, S.M. & E.E. Thesis, EE & CS Dept., June 1980, AD A087-040

TR-240	Montz, Lynn B. Safety and Optimization Transformations for Data Flow Programs, S.M. Thesis, EE & CS Dept., July 1980
TR-241	Archer, Rowland F., Jr. Representation and Analysis of Real-Time Control Structures, S.M. Thesis, EE & CS Dept., August 1980, AD A089-828
TR-242	Loui, Michael C. Simulations Among Multidimensional Turing Machines, Ph.D. Dissertation, EE & CS Dept., August 1980
TR-243	Svobodova, Liba Management of Object Histories in the Swallow Repository, August 1980, AD A089-836
TR-244	Ruth, Gregory R. Data Driven Loops, August 1980
TR-245	Church, Kenneth W. On Memory Limitations in Natural Language Processing, S.M. Thesis, EE & CS Dept., September 1980
TR-246	Tiuryn, Jerzy A Survey of the Logic of Effective Definitions, October 1980
TR-247	Weihl, William E. Interprocedural Data Flow Analysis in the Presence of Pointers, Procedure Variables, and Label Variables, S.B.& S.M.Thesis, EE & CS Dept., October 1980
TR-248	LaPaugh, Andrea S. Algorithms for Integrated Circuit Layout: An Analytic Approach, Ph.D.Dissertation, EE & CS Dept., November 1980
TR-249	Turkle, Sherry Computers and People: Personal Computation, December 1980
TR-250	Leung, Clement Kin Cho Fault Tolerance in Packet Communication Computer Architectures, Ph.D. Dissertation, EE & CS Dept., December 1980

TR-251	Swartout, William R. Producing Explanations and Justifications of Expert Consulting Programs, Ph.D. Dissertation, EE & CS Dept., January 1981
TR-252	Arens, Gail C. Recovery of the Swallow Repository, S.M. Thesis, EE & CS Dept., January 1981, AD A096-374
TR-253	Ilson, Richard An Integrated Approach to Formatted Document Production, S.M. Thesis, EE & CS Dept., February 1981
TR-254	Ruth, Gregory, Steve Alter and William Martin A Very High Level Language for Business Data Processing, March 1981
TR-255	Kent, Stephen T. Protecting Externally Supplied Software in Small Computers, Ph.D. Dissertation, EE & CS Dept., March 1981
TR-256	Faust, Gregory G. Semiautomatic Translation of COBOL into HIBOL, S.M. Thesis, EE & CS Dept., April 1981
TR-257	Cisari, C. Application of Data Flow Architecture to Computer Music Synthesis, S.B./S.M. Thesis, EE & CS Dept., February 1981
TR-258	Singh, N. A Design Methodology for Self-Timed Systems, S.M. Thesis, EE & CS Dept., February 1981
TR-259	Bryant, R.E. A Switch-Level Simulation Model for Integrated Logic Circuits, Ph.D. Dissertation, EE & CS Dept., March 1981
TR-260	Moss, E.B. Nested Transactions: An Approach to Reliable Distributed Computing, Ph.D. Dissertation, EE & CS Dept., April 1981
TR-261	Martin, W.A., Church, K.W., Patil, R.S. Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results, EE & CS Dept., June 1981

TR-262	Todd, K.W. High Level Val Constructs in a Static Data Flow Machine, S.M. Thesis, EE & CS Dept., June 1981
TR-263	Street, R.S. Propositional Dynamic Logic of Looping and Converse, Ph.D. Dissertation, EE & CS Dept., May 1981
TR-264	Schiffenbauer, R.D. Interactive Debugging in a Distributed Computational Environment, S.M. Thesis, EE & CS Dept., August 1981
TR-265	Thomas, R.E. A Data Flow Architecture with Improved Asymptotic Performance, Ph.D. Dissertation, EE & CS Dept., April 1981
TR-266	Good, M. An Ease of Use Evaluation of an Integrated Editor and Formatter, S.M. Thesis, EE & CS Dept., August 1981
TR-267	Patil, R.S. Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis, Ph.D. Dissertation, EE & CS Dept., October 1981
TR-268	Guttag, J.V., Kapur, D., Musser, D.R. Derived Pairs, Overlap Closures, and Rewrite Dominoes: New Tools for Analyzing Term Rewriting Systems, EE & CS Dept., December 1981
TR-269	Kanellakis, P.C. The Complexity of Concurrency Control for Distributed Data Bases, Ph.D. Dissertation, EE & CS Dept., December 1981
TR-270	Singh, V. The Design of a Routing Service for Campus-Wide Internet Transport, S.M. Thesis, EE & CS Dept., January 1982
TR-271	Rutherford, C.J., Davies, B., Barnett, A.I., Desforges, J.F. A Computer System for Decision Analysis in Hodgkins Disease, EE & CS Dept., February 1982
TR-272	Smith, B.C. Reflection and Semantics in a Procedural Language, Ph.D. Dissertation, EE & CS Dept., January 1982

TR-273	Estrin, D.L. Data Communications via Cable Television Networks: Technical and Policy Considerations, S.M. Thesis, EE & CS Dept., May 1982
TR-274	Leighton, F.T. Layouts for the Shuffle-Exchange Graph and Lower Bound Techniques for VLSI, Ph.D. Dissertation, EE & CS Dept., August 1981
TR-275	Kunin, J.S. Analysis and Specification of Office Procedures, Ph.D. Dissertation, EE & CS Dept., February 1982
TR-276	Srivas, M.K. Automatic Synthesis of Implementations for Abstract Data Types from Algebraic Specifications, Ph.D. Dissertation, EE & CS Dept., June 1982
TR-277	Johnson, M.G. Efficient Modeling for Short Channel Mos Circuit Simulation, S.M. Thesis, EE & CS Dept., August 1982
TR-278	Rosenstein, L.S. Display Management in an Integrated Office, S.M. Thesis, EE & CS Dept., January 1982
TR-279	Anderson, T.L. The Design of a Multiprocessor Development System, S.M. Thesis, EE & CS Dept., September 1982
TR-280	Guang-Rong, G. An Implementation Scheme for Array Operations in Static Data Flow Computers, S.M. Thesis, EE & CS Dept., May 1982
TR-281	Lynch, N.A. Multilevel Atomicity - A New Correctness Criterion for Data Base Concurrency Control, EE & CS Dept., August 1982
TR-282	Fischer, M.J., Lynch, N.A., Paterson, M.S. Impossibility of Distributed Consensus with One Faulty Process, EE & CS Dept., September 1982

TR-283	Sherman, H.B. A Comparative Study of Computer-Aided Clinical Diagnosis, S.M. Thesis, EE & CS Dept., January 1981
TR-284	Cosmadakis, S.S. Translating Updates of Relational Data Base Views, S.M. Thesis, EE & CS Dept., February 1983
TR-285	Lynch, N.A. Concurrency Control for Resilient Nested Transactions, EE & CS Dept., February 1983
TR-286	Goree, J.A. Internal Consistency of a Distributed Transaction System with Orphan Detection, S.M. Thesis, EE & CS Dept., January 1983
TR-287	Bui, T.N. On Bisecting Random Graphs, S.M. Thesis, EE & CS Dept., March 1983
TR-288	Landau, S.E. On Computing Galois Groups and its Application to Solvability by Radicals, Ph.D. Dissertation, EE & CS Dept., March 1983
TR-289	Sirbu, M., Schoichet, S.R., Kunin, J.S., Hammer, M.M., Sutherland, J.B., Zarmer, C.L. Office Analysis: Methodology and Case Studies, EE & CS Dept., March 1983
TR-290	Sutherland, J.B. An Office Analysis and Diagnosis Methodology, S.M. Thesis, EE & CS Dept., March 1983
TR-291	Pinter, R.Y. The Impact of Layer Assignment Methods on Layout Algorithms for Integrated Circuits, Ph.D. Dissertation, EE & CS Dept., August 1982
TR-292	Dornbrook, M., Blank, M. The MDL Programming Language Primer, EE & CS Dept., June 1980
TR-293	Galley, S.W., Pfister, G. The MDL Programming Language, EE & CS Dept., May 1979

TR-294	Lebling, P.D. The MDL Programming Environment, EE & CS Dept., May 1980
TR-295	Pitman, K.M. The Revised Maclisp Manual, EE & CS Dept., June 1983
TR-296	Church, K.W. Phrase-Structure Parsing: A Method for Taking Advantage of Allophonic Constraints, Ph.D. Dissertation, EE & CS Dept., June 1983
TR-297	Mok, A.K. Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment, Ph.D. Dissertation, EE & CS Dept., June 1983
TR-298	Krugler, K. Video Games and Computer Aided Instruction, EE & CS Dept., June 1983
TR-299	Wing, J. A Two Tiered Approach to Specifying Programs, June 1983
TR-300	Cooper, G. An Argument for Soft Layering of Protocols, May 1983
TR-301	Valente, J.A. Creating a Computer-based Learning Environment for Physically Handicapped Children, Ph.D. Dissertation, EE & CS Dept., September 1983
TR-302	Arvind, Dertouzos, M.L. and Iannucci, R.A. A Multiprocessor Emulation Facility, October 1983
TR-303	Bloom T. Dynamic Module Replacement in a Distributed Programming System, Ph.D. Dissertation, EE & CS Dept., September 1983
TR-304	Terman, C.J. Simulation Tools for Digital LSI Design, Ph.D. Dissertation, EE & CS Dept., September 1983
TR-305	Bhatt, S.N. and Leighton, F.T. A Framework for Solving VLSI Graph Layout Problems, Ph.D. Dissertation, EE & CS Dept., October 1983
TR-306	Leung, K.C. and Lim, W. Y-P.

	PADL -- A Packet Architecture Description Language: A Preliminary Reference Manual, October 1983
TR-307	Guttag, J.V. and Horning, J.J. Preliminary Report on the Larch Shared Language, October 1983
TR-308	Oki, B.M. Reliable Object Storage to Support Atomic Actions, M.S. Thesis, EE & CS Dept., November 1983
TR-309	Brock, J.D. A Formal Model of Non-determinate Dataflow Computation, Ph.D. Dissertation, EE & CS Dept., November 1983
TR-310	Granville, R. Cohesion in Computer Text Generation: Lexical Substitution, M.S. Thesis, EE & CS Dept., December 1983
TR-311	Burke, G.G., Carrette, G.J. and Eliot, C.R. NIL Reference Manual, M.S. Thesis, EE & CS Dept., December 1983
TR-312	Landcaster, J. Naming in a Programming Support Environment, M.S. Thesis, EE & CS Dept., April 1984
TR-313	Koile, K. The Design and Implementation of an Online Directory Assistance System, M.S. Thesis, EE & CS Dept., April 1984
TR-314	Weihl, W. Specification and Implementation of Atomic Data Types, Ph.D. Dissertation, EE & CS Dept., April 1984
TR-315	Coan, B. and Turpin, R. Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement, April 1984
TR-316	Comer, M.H. Loose Consistency in a Personal Computer Mail System, S.B. & M.S. Thesis, May 1984
TR-317	Traub, K.R. An Abstract Architecture for Parallel Graph Reduction, S.B. Thesis, May 1984
TR-324	Schooler, R. Partial Evaluation as a Means of Language Extensibility, S.M. Thesis/August 1984

TR-327	Chiu, S.Y. Debugging Distributed Computations in a Nested Atomic Action System, Ph.D. Dissertation/December 1984
TR-328	Carnese, D.J. Multiple Inheritance in Contemporary Programming Languages, September 1984
TR-329	Sacks, E. Qualitative Mathematical Reasoning, November 1984
TR-330	Sarin, S.K. Interactive On-Line Conferences, Ph.D. Dissertation/June 1984
TR-331	Sollins, K.R. Distributed Name Management, Ph.D. Dissertation/February 1985
TR-332	Culler, D.E. Resource Management for the Tagged Token Dataflow Architecture, S.M. Thesis/January 1985
TR-333	Arnold, J.M. Parallel Simulation of Digital LSI Circuits, S.M. Thesis/April 1984
TR-335	Lundelius, J. Synchronizing Clocks in a Distributed System, S.M. Thesis/August 1984
TR-336	Trilling, S. Some Implications of Complexity Theory on Pseudo-Random Bit Generation, S.M. Thesis/January 1985
TR-337	Seiler, L.D. A Hardware Assisted Methodology for VLSI Design Rule Checking, Ph.D. Dissertation/February 1985
TR-338	Koton, P.A. Towards a Problem Solving System for Molecular Genetics, May 1985
TR-339	Soley, R.M. Generic Software for Emulating Multiprocessor Architectures, A Thesis/ May 1985
TR-340	Wellman, M.P. Reasoning About Preference Models, S.M. Thesis/May 1985
TR-341	Boughton, G.A.

	Routing Networks for Packet Communication Systems, Ph.D. Dissertation/ August 1984
TR-342	Stark, E.W. Foundations of a Theory of Specification for Distributed Systems, Ph.D. Dissertation/August 1984
TR-343	Forgaard, R. A Program for Generating and Analyzing Term Rewriting Systems, S.M. Thesis/ September 1984
TR-351	Bhatt, S.N. The Complexity of Graph Layout and Channel routing for VLSI, Ph.D. Disserta- tion/February 1984
TR-355	Guharoy, B. Data Structure Management in a Data Flow Compter System, S.M. Thesis/May 1985

Progress Reports

1. Project MAC Progress Report I, to July 1964, AD 465-088
2. Project MAC Progress Report II, July 1964-July 1965, AD 629-494
3. Project MAC Progress Report III, July 1965-July 1966, AD 648-346
4. Project Mac Progress Report IV, July 1966-July 1967, AD 681-342
5. Project MAC Progress Report V, July 1967-July 1968, AD 687-770
6. Project MAC Progress Report VI, July 1968-July 1969, AD 705-434
7. Project MAC Progress Report VII, July 1969-July 1970, AD 732-767
8. Project MAC Progress Report VIII, July 1970-July 1971, AD 735-148
9. Project MAC Progress Report IX, July 1971-July 1972, AD 756-689
10. Project MAC Progress Report X, July 1972-July 1973, AD 771-428
11. Project MAC Progress Report XI, July 1973-July 1974, AD A004-966
12. Laboratory for Computer Science Progress Report XII, July 1974-July 1975, AD A024-527
13. Laboratory for Computer Science Progress Report XIII, July 1975-July 1976, AD A061-246
14. Laboratory for Computer Science Progress Report XIV, July 1976-July 1977, AD A061-932
15. Laboratory for Computer Science Progress Report 15, July 1977-July 1978, AD A073-958

16. Laboratory for Computer Science Progress Report 16, July 1978-July 1979, AD A088-355

17. Laboratory for Computer Science Progress Report 17, July 1979-July 1980, AD A093-384

18. Laboratory for Computer Science Progress Report 18, July 1980-June 1981, A 127586

19. Laboratory for Computer Science Progress Report 19, July 1981-June 1982, A 143429

20. Laboratory for Computer Science Progress Report 20, July 1982-June 1983, A 145134

21. Laboratory for Computer Science Progress Report 21, July 1983-June 1984, A 154810

Copies of all reports with A, AD, or PB numbers listed in Publications may be secured from the National Technical Information Service, U.S. Department of Commerce, Reports Division, 5285 Port Royal Road, Springfield, Virginia 22161 (tel: 703-487-4650). Prices vary. The reference number must be supplied with the request.

OFFICIAL DISTRIBUTION LIST

DIRECTOR	2 copies
Information Processing Techniques Office	
Defense Advanced Research Projects Agency (DARPA)	
1400 Wilson Boulevard	
Arlington, VA 22209	
OFFICE OF NAVAL RESEARCH	2 copies
800 North Quincy Street	
Arlington, VA 22217	
Attn: Dr. Gary Koop, Code 433	
DIRECTOR, CODE 2627	6 copies
Naval Research Laboratory	
Washington, DC 20375	
DEFENSE TECHNICAL INFORMATION CENTER	12 copies
Cameron Station	
Alexandria, VA 22314	
NATIONAL SCIENCE FOUNDATION	2 copies
Office of Computing Activities	
1800 G. Street, N.W.	
Washington, DC 20550	
Attn: Program Director	
HEAD, CODE 38	1 copy
Research Department	
Naval Weapons Center	
China Lake, CA 93555	