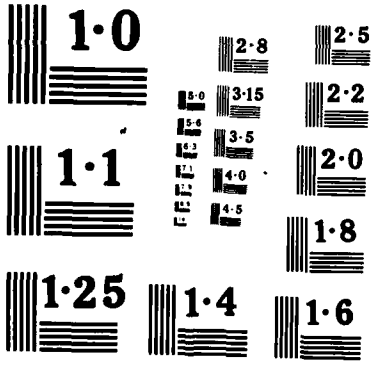
AD-A158 299 LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 21 JULY 1/4
83-JUNE 84(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB
FOR COMPUTER SCIENCE M DERTOUZOS JUN 84

UNCLASSIFIED F/G 9/2 NL


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5

Massachusetts Institute of Technology

Laboratory for Computer Science Progress Report

July 1983-
June 1984

Massachusetts
Institute of
Technology

July 1983-
June 1984

Laboratory for Computer Science Progress Report

21

The work reported herein was carried out within the Laboratory for Computer Science, an MIT interdepartmental laboratory. During 1983-84 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 during the last 21 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, Office of Scientific Research;
- MIT controlled IBM funds under an IBM/MIT joint study contract;

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

Final assembly and production of this report was done by Paula Vancini with assistance from the support staff of each research group.

TABLE OF CONTENTS

ADMINISTRATION

Academic Staff

M. Dertouzos	Director
M. Rivest	Associate Director
A. Vezza	Associate Director

Administrative Staff

P. Anderegg	Assistant Administrative Officer
J. Hynes	Administrative Officer
M. Jones	Fiscal Officer

Support Staff

G. Brown	C. Morin
J. Coleman	E. Profirio
L. Cavallaro	M. Sensale
R. Cinq-Mars	J. Spillane
R. Donahue	C. Stevens
T. LoDuca	P. Vancini

INTRODUCTION

The MIT Laboratory for Computer Science (LCS) is an interdepartmental laboratory whose 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 Systems (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 objectives expanded, leading to research across a broad front of activities.

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 systems 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 are studied by the Computer Systems and Communications Group, while distributed file servers are pursued by the Distributed Computer Systems Group. Finally a key application, involving the tailoring of news and other community information to individual needs, is pursued by the Imaginative Systems Group.

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

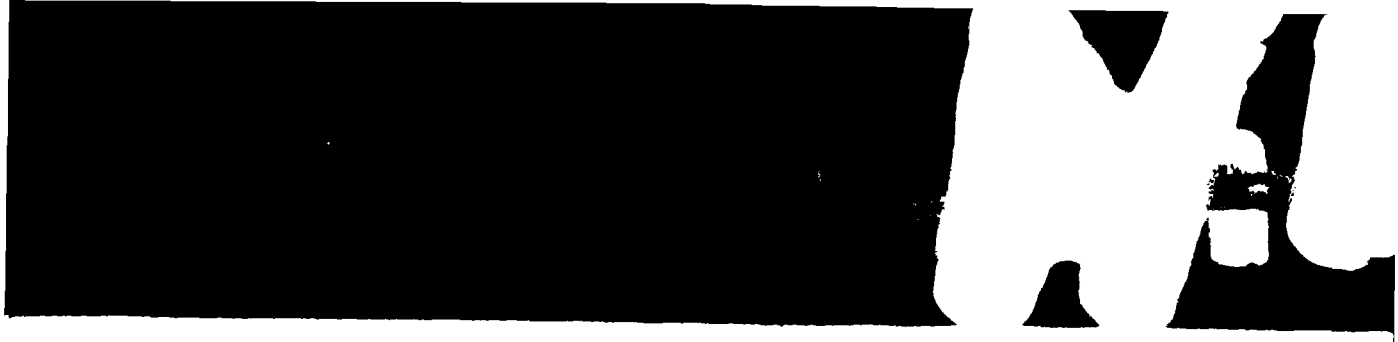
programming languages from both analytical and synthetic view- points; the logic of programs; and the links between mathematics and the privacy/authentication of computer-to-computer messages. Other examples of work in this area involve the study of distributed systems, by the Theory of Distributed Systems Research Group, and routing algorithms for VLSI circuits.

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 1983-1984, the Laboratory embarked on the ambitious project of constructing Project Tanglewood, an emulation facility consisting of 64 interconnected large computers, 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 continuing development of the MultiLisp multiprocessor language by Professor Robert Halstead of the Real Time Systems Group. A multiprocessor applications workshop sponsored by members of the Laboratory was held in Spring 1984 to establish the amount of parallelism that can be expected in a variety of applications.

Another growth activity during 1983-84 has been the newly established Educational Computing Group which is now headed by Dr. Andrea diSessa and includes Professors Harold Abelson, Seymour Papert, and Dr. Sylvia Weir. This group, which in the last 12 years developed the widely used language LOGO, is currently focusing its efforts on the development of Boxer, a successor to LOGO that encompasses new concepts in the computer and cognitive sciences and in educational innovation.

During this reporting period we have also made substantial progress in distributed systems research. The NuBus architecture that we developed was successfully transferred to industry (Texas Instruments) and we took delivery of 30 Texas Instruments Nu Machines supplied to us in exchange for our contributions. These and other related machines (single-user Vaxes and Lisp Machines) are being interconnected into prototype interconnected systems within the Laboratory thereby forming an experimental basis for the study of distributed systems. During the Spring of 1984, key researchers in distributed systems presented their results to some 400 attendees in an ILP-sponsored conference.



During 1983-84, the Laboratory formed two new entities -- the Distributed Computer Systems Research Group and the VLSI Design Project. The first entity, headed by Senior Research Scientist Dr. David D. Clark is concerned with the architecture of distributed systems and in particular with file servers and communication protocols. The second entity, headed by Professors Charles Leiserson and Richard Zippel, is intended to coalesce and focus the various VLSI design activities within LCS. A key research activity of the VLSI Design Project is the study and development of an expert VLSI design system, Schema, which will be used as a common basis for all MIT VLSI design research.

Other events in 1983-84 were the arrival of three IBM engineers who will help us construct the Laboratory's Emulation Facility; and the launching of the Laboratory's bimonthly newsletter -- The Gateway.

In 1984, the Laboratory issued the LCS Achievement Award to Professor Joel Moses for his pioneering work on MACSYMA; and the one-time LCS Founder's Award to the founder of LCS (then Project MAC) Professor Robert M. Fano. Professor Fano will be retired effective July 1984, but will remain a part time member of the Laboratory. Other departures during 1983-84 included Professor Christos Papadimitriou (to Stanford) and Professor Michael Hammer (to his own company).

Arrivals in the same period were Assistant Professors Shafi Goldwasser, Silvio Micali, Ramesh Patil, Christopher Terman, and Research Associates William Ackerman and Benjamin Kuipers. Our Laboratory consisted of 340 members -- 53 faculty and academic research staff, 30 visitors and visiting faculty, 57 professional and support staff, 110 graduate and 90 undergraduate students -- organized into 16 research groups. Laboratory research during 1983-84 was funded by 16 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 1983-84 were disseminated through publications in technical literature, through Technical Reports (TR299-TR317), and through Technical Memoranda (TM238-TM262).

CLINICAL DECISION MAKING

Academic Staff

P. Szolovits, Group Leader
R. Patil

Collaborating Investigators

M. Criscitiello, M.D., Tufts-New England Medical Center Hospital
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.P. Kassirer, M.D., Tufts-New England Medical Center Hospital
M. Klein, M.D., University Hospital, Boston University
J. Lau, M.D., Tufts University-New England Medical Center Hospital
A. Moskowitz, M.D., Tufts-New England Medical Center Hospital
S. Naimi, M.D., Tufts-New England Medical Center Hospital
S.G. Pauker, M.D., Tufts-New England Medical Center
W.B. Schwartz, M.D., Tufts-New England School of Medicine
L. Widman, M.D., Case Western Reserve University Medical School,

Research Staff

G. Burke
C. Eliot
W. Long

Graduate Students

R. Granville
P. Koton
R. Kunstaetter
T. Russ
E. Sachs
M. Wellman

Undergraduate Students

S. Ferguson
R. Goldberg
M. Harvey
C. Kim
C. Park
V. Simonaitis

Visitors

M. Feinberg
M. Fieschi
J. Hunter
I. Kohane
B. Kuipers

Support Staff

R. Hegg

communication. People use models of their conversational partners, models that contain ideas of the other's knowledge and beliefs, in order to determine the correct context of the conversation. Any computer system that engages in intelligent discourse will have to have a similar model of the user, and it will have to be able to derive this model. Our plan is to build a prototype system that forms this model much in the same way that people do, through past experiences with the user, making assumptions about common knowledge where specific knowledge fails.

3.4. Research Plan

We plan to continue work on both the Heart Failure Project and the Ventricular Arrhythmia Management Advisor Project. Also, our current work on user modeling will continue, and we hope to have a more complete definition of the initial model and to begin a computer implementation during the coming year. In addition, we plan to begin development of an experimental environment where we can test our theories on actual student users. We are currently trying to select the appropriate application area for this; initially, it must be a program whose medical expertise is more limited and easier to build than the two major projects described above, so that we may have a testbed in the relatively near future.

4. DEVELOPMENT OF TOOLS FOR CLINICAL DECISION ANALYSIS

4.1. Executive Summary

In this subproject, the investigators have been developing and modifying a microcomputer program to perform clinical decision analysis. The program presumes the user is familiar with the basic principles of decision analysis (formulating problems as decision trees, assigning likelihoods [probabilities] and relative values [utilities], and interpreting the results of systematic variations in the values of single parameters or sets of parameters [sensitivity analyses]. The program allows the users to efficiently specify and analyze such decision problems. In the course of the first twelve months of the project, the program has been rewritten in several different environments to explore its extensibility and to improve it ease of use. We have directed our efforts entirely at the IBM-PC personal computer because it has the lion's share of the microcomputer market. The original program is written in UCSD Pascal. That system, with its compact P-code representation was necessary to "fit" the program into a microcomputer. Two more recently developed programming environments that can use larger amounts of memory (IQLISP and Turbo Pascal) have been used to provide versions that do not require the user to purchase the expensive P-system and version that are compiled, producing a five to twenty-fold increase in processing speed. In the past year we have also enlarged our

depth, it is going to have to use context in both understanding the user and in order to provide appropriate responses. Then it follows that the system is going to need a model of the user's knowledge and beliefs in order to provide the required context [5]. Two currently open problems with user models are how must the system's model of the user change as the discourse progresses, and where does the information in the model come from in the first place?

A good deal of active research is currently attacking the first of these open questions, especially from the end of understanding user statements. Examples of ongoing work include that of Pollack[9], Carberry[1] and Granger[2]. However, no one has proposed a theory with which a system can initially create or derive a model of the user. This is the problem that is being addressed in this project.

The solution proposed here is based on a theory of the way people derive their models of others for discourse. In general, we know what we know and believe, and more importantly, we have an idea how common this knowledge and these beliefs are. When I meet an adult, I assume he knows the difference between red and green, since I believe this is common knowledge learned while very young, but I would not assume he is necessarily familiar with Fillmore's case grammar theory, because I believe only people with a special interests in linguistics have studied case grammar, and that not everyone is that interested in the subject. On the other hand, if I am introduced to someone and am told as part of the introduction that the person is a linguist, I would expect him to be at least aware of Fillmore's work. If I believe that there is a little green frog in the bottom drawer of my desk that created the universe, I might tenaciously hold this belief, but I would also recognize that the vast majority of people, for whatever reason, do not share this view.

In other words, it seems that the initial model we form of a person for discourse purposes is the first impression (for the present meeting) we have of that person. Obviously, the more information we have, the more accurate the impression. Upon meeting an intimate friend, our model would be fairly elaborate and accurate. Meeting a stranger for the first time would allow us to form only a vague model based on what we believe is common to the average person.

Computationally, this suggests a hierarchical default mechanism for generating a user model. Each specific piece of information known about the user, either from previous encounters or from being told, can be entered directly into the model. Other pieces can receive default values from what the system believes to be common knowledge or average values. Obviously, the system would need to keep track of what is actually known and what is being assumed about the user. As more information is learned about the user through discourse, default values can be replaced with actual facts.

In conclusion, we have seen that context is essential for meaningful

utterances will be understood. Telling a patient suspected of having leukemia that the test results were positive, meaning that the results were positively conclusive or that they were favorable, would be incorrect because in this context positive means that the illness being tested for is present.

But context is more important than avoiding the misinterpretation of statements. If somehow the speaker and hearer can agree on the context for the discourse, both people's jobs are made easier. Whatever is in the agreed context can be taken for granted in that both participants know of its existence and understand it. Then conversation can be restricted to those statements that somehow provide information that is not yet in the agreed context. This allows the speaker to follow the tenets of conversation observed by Grice[3]. And if we needed to describe every concept each time we mentioned it, we would literally never be able to finish a thought, since every concept used to describe the initial idea would itself have to be described, and each of the items used in those descriptions would have to be described, and so forth.

Accepting that context is essential for discourse, we might ask exactly what do we require to be in the context? As a speaker, we need not only every piece of information we know that might affect what we want to say, but also how each piece will affect the interpretations of our statements for the hearer. And as a hearer, we need to know what effects the speaker meant to have by his statements. In other words, in addition to knowing what we know and believe concerning the topic of conversation, we have to know (or least have an idea of) what the other person knows and believes. Since we cannot infallibly know what is in another's head, at best we can have a model of the knowledge and beliefs of our partner in conversation.

We use models of others' knowledge and beliefs at all stages when we engage in communication. It is used when we are determining what is appropriate to say. For example, if you were stopped in the streets by someone in a car with out of state plates and the driver asked you directions to a local famous site, you would probably not give directions based on the location of other sites that would be well known to residents, but not to tourists. This is because in this situation you believe the driver generally doesn't know her way around town. On the other hand, if you were giving directions to a friend whom you know to have lived in this city for years, you would use local landmarks because you believe she is familiar with them.

Models are also used in determining an appropriate way to make a statement. To use the same example, one might say to a resident of Boston, "Go to the Hancock and turn left," whereas to a tourist a more appropriate statement would be "Go to that tall glass building and turn left."

If a computer system is ever going to engage in a discussion of any significant

determination of the appropriate recommendation depends on the assessment of costs of the two conditions and the assessment of their likelihoods. The reasoning that leads to the actual therapy recommendation can be made more general by including explicitly the risk-benefit analysis that supports it. This strategy simplifies several problems. Inclusion of new therapies involves changing the assessment of costs to reflect the particular characteristics of the therapy. Changes in understanding about the properties of a therapy either because of new medical knowledge or a different school of thought are similarly incorporated as changes in the assessments or as changes in the strategies for optimizing the achievement of therapeutic goals.

Over the past year our work on this project has centered on the gathering of case material to test both an initial trial program for arrhythmia advice and for the development of the appropriate design for the Advisor. The case material includes a complete record of the minute-by-minute arrhythmia data from an arrhythmia monitor as well as the clinical information needed to follow the disease state and the record of therapeutic interventions. On a few cases we have also been able to obtain drug level information for verifying pharmacokinetic models. So far we have collected data on 44 cases. This case material is not as detailed as we will ultimately need for verifying particular assessment algorithms, but it is serving us well for developing the initial design.

3.3. User Models in Discourse

The problem of modeling what the user knows already and wants to know is difficult. During the past year we have begun to concentrate on one aspect of this problem: the role of conversational context in generating appropriate text.

When engaging in communication, it is important to know the context in which statements are being made. This is equally true for the speaker and the hearer. (Or the writer and the reader. We will not distinguish between oral and written communication here unless we do so specifically by exception.) It is necessary for the hearer to know the context in which statements are being made in order to fully understand them. For example, consider the sentence "The test result was positive." In the context of a qualifying exam, this statement made to the candidate would be cause for celebration. On the other hand, this statement made to a patient being tested for leukemia has a completely different meaning. And the statement made to two brothers who are trying to see if one can donate a kidney to the other has yet a third meaning. It is easy to come up with many contexts for which this statement has a new meaning. It is obviously important for the hearer to know the context in which an utterance is made.

It is equally important for the speaker to be aware of the context in which his

alternative relations that might exist in the patient. This kind of flexibility is possible because the model provides a consistent physiological explanation of what is known about the patient. Modifications to that representation are supported by a Truth Maintenance System that propagates the implications throughout the model. Thus, if there are inconsistencies in the deduced state, these will be pointed out by the system along with the alternatives for correcting them. These mechanism makes it possible for the user to realistically consider changes in the conclusions of the system and to tailor the reasoning to whatever the user knows about the patient.

Over the past year we have made significant strides in the development of the heart failure program. To explore the important representational issues, we are initially developing the model on a subset of the domain—the problem of managing angina in the context of possible heart failure. We have put together a causal model for this subdomain that includes about fifty nodes related as definite and possible causal factors and worsening (predisposing) factors. With this model we are developing general strategies for the assessment of evidence from the patient findings, strategies for pursuing a diagnosis, and strategies for finding and assessing therapy choices. A working version of this program has given us new understanding of some of the needs for explanation and the necessary ingredients for a useful explanation over the last few months.

3.2. The Ventricular Arrhythmia Management Advisor

The problem of generalizing the strategies for therapy management are being addressed by the Ventricular Arrhythmia Management Advisor. The domain of the program is the control of ventricular arrhythmias in the context of the intensive care unit. Many of these arrhythmias occur as a result of ischemia but there are also a number of other causes as well as factors that may worsen the problem. The therapy for these arrhythmias is of two types. Either it is directed at the arrhythmia itself or it is directed at the cause. The arrhythmia therapies commonly employed include lidocaine, procainamide, quinidine, bretylium, and others. The determination of which drug is appropriate in an individual is primarily done by trial and error. Thus, several may be tried before an acceptable patient response is achieved. These are all drugs with multiple compartment pharmacokinetic models, high interpatient variability, narrow therapeutic windows, and high incidence of toxicity. Thus, it is an excellent domain in which to consider the issues of generalization of patient management strategies.

Our approach to generalization is to look for the underlying unifying concepts that run through the patient management process. For example, in adjusting the dosage of a drug for anticipated sensitivities, the physician is actually going through a particular instance of risk-benefit analysis in which the risk is the likelihood of toxicity and the benefit is the likelihood of control of the arrhythmia. The

3.1. The Heart Failure Project

The Heart Failure Project has been particularly fruitful for developing a model of user access and control over the reasoning process. The central observation in the development of a design for this program is that much of the reasoning that supports both diagnosis and therapeutic decisions is based on the particular complex of disease processes and physiological compensations exhibited by the patient. The most obvious way to produce similar kinds of reasoning in a computer program is to do the reasoning from a physiological representation of the understanding of the patient's state. Thus we are developing a program that has at the center a qualitative causal physiological model of the factors related to heart failure, which can represent what is known about the patient state. This model is used for understanding the significance of the findings, reasoning about the possible diagnoses, reasoning about appropriate therapies, and providing explanations for all of these kinds of reasoning.

A physiological model centered approach to the patient management process gives a user access not only to the reasoning of the program but also to the justifications for the reasoning. That is, it is possible for the user to examine and explore the physiological relationships concluded by the program and used to support the recommendations. For example, if the program suggests the use of an arterial vasodilator to increase the cardiac output, the physiological model would provide a consistent picture of why the recommendation is appropriate. This would include facts such as evidence of vasoconstriction caused by a high sympathetic state and therefore the likelihood that the cardiac output would increase rather than causing the arterial pressure to decrease. In addition the model provides the user with the reasons for being cautious in giving such a therapy since a possible alternative result is for the arterial pressure to drop.

The most important consideration in developing a physiological model for supporting this kind of reasoning is to provide the relationships at the appropriate physiological level. From our exploration of the heart failure domain it appears that the level needed for reasoning and for explanation are the same. It also appears that a single level which includes the hemodynamic relationships in the cardiovascular system is adequate for this domain. This is not so in other domains, as pointed out by Patil in the acid-base domain, but the single level in this domain simplifies the problem for exploring other issues of diagnosis and management reasoning.

The physiological model also makes it possible for the user to exert control over the reasoning process. We have observed that the user many times is better able to assess physiological states of the patient than a program that has a limited perspective on the problem and only the ability to ask questions of the user. Thus, our intention is to collaborate with the user in the reasoning process and give the user the ability to make hypotheses, change conclusions, and in general to test the

The diagnostic system at Tufts employs an Evoker module that is modeled after the categorical aspects of the Present Illness Program developed here [7]. Its purpose is to evoke a collection of elementary hypotheses, which are then assembled by a build module, into patient-specific hypotheses. Given a set of active hypotheses the system must choose diagnostic testing for the confirmation of these hypotheses. We will be investigating the implementation of a test planning module that will employ the reasoning processes that expert physicians use in their pursuit of medical diagnostic and therapeutic strategies.

3. EXPLANATION AND JUSTIFICATION BY EXPERT PROGRAMS

This section focuses on the development of improved explanation and justification methods for the Digitalis Therapy Program and the extension of these methods to other problem domains. Our views of both the Digitalis Therapy Advisor and of how best to accomplish the objectives outlined in the proposal have evolved over the last four years. With the increased use of a large variety of drugs in addition to digitalis in many conditions for which digitalis was used almost exclusively, it has become obvious that a successful program must deal with a larger, coherent management problem rather than concentrate on the use of a single drug. Therefore, the context of this research has broadened to encompass two logical outgrowths of our early work on digitalis. The first of these is the Heart Failure Project. The objective of this project is to help the user reason about the diagnosis and management of heart failure by relating the findings to the possible pathophysiology that might be responsible. This project is an outgrowth of the observation in the Digitalis Therapy Advisor that it is difficult to relate the changes in the digitalis levels to the changes in the heart failure manifestations without taking into account the other therapies that might be involved and the particular nature of the heart failure in the patient. The second project is the Ventricular Arrhythmia Management Advisor. This project extends the idea of computer assisted therapy management to the management of a class of disease situations where any of a number of therapies might be applicable.

These two projects have given us a clearer understanding of the problems of giving the user useful explanations of the reasoning and advice of such programs. In particular we have been making significant strides over the past year to design programs where the user has access to the disease, physiological, and therapeutic relationships behind the advice given by the program. One important benefit of the explicit representation of these relationships is that it promises to let us build systems wherein the user has a much larger degree of control over the reasoning process and the conclusions reached by the program. The program thus becomes more a "reactive blackboard" on which the user can try out various ideas, rather than the more traditional expert decision maker. Finally, the changes needed to allow our programs to deal with larger disease classes, in which there are significant therapy choices to be made, also simplify the addition of new therapy modalities.

2.3. Plans For The Coming Year

Completing the Validation of the Model of the Reasoning Process and Additional Transcript Analysis: We plan to complete the evaluation of the model's completeness and generality utilizing an existing rule system to simulate the decisions being made in the transcript. EMYCIN, [8] a domain independent rule system will accept the set of rules that we derived from the transcript and systematically apply them until a goal attribute is established, the system exhausts the rules, or the rules that we derived fail. Our goal attribute is the same choice of action that our expert physician chose. The adequacy of the knowledge encoded by these rules will be assessed by applying them to similar management problems but with a slightly different context. The degree to which we must append the set of rules to establish our goal will be a measure of their adequacy.

As a means of establishing empiric validity of our model of an expert physician's reasoning processes, we will be analyzing transcripts of the same physician, solving problems on a case in a less familiar clinical domain. We suspect that similar reasoning processes are employed for decisions with clinical problems that can be structured in a similar fashion.

We also have collected several other transcripts from other expert level physicians with other areas of expertise than the present physician-subject. We will be evaluating these transcripts in the manner described above. The process will allow us to assess the similarities and differences in the reasoning processes employed by expert physicians with different areas of expertise, for problems both in and out of their domain of expertise. We will be looking to establish whether the same theoretical model of the reasoning process found for the pulmonologist is employed by other physicians facing the same decision.

We are interested in characterizing the development of clinical cognition. Specifically we would like to follow the ontogeny of probabilistic reasoning and the development of reasoning tools for making tradeoffs in the risks imposed by diagnostic tests and therapy. We will start to collect transcripts of physicians at various stages of their training (3rd year and 4th year medical students, 2nd year Internal Medicine residents and medical sub-specialty fellows), as they solve the same problems posed to our expert level physicians.

Developing a Computational Model of the Decision Process: Understanding the structure of knowledge and problem-solving methods that underlie clinical expertise will have a significant impact on the design of knowledge-based artificial intelligence systems in medicine. We have begun to investigate putting together results of our study of expert clinical reasoning with a hypothesis-directed diagnostic system that is currently in working model form at Tufts University, Medford campus (B. J. Kuipers).

section. In this manner, we reviewed the transcript and were able to characterize many of the reasoning processes employed by the physician-subject.

Validation of the model of the reasoning process: The model's completeness and consistency was assessed by two means, its ability to account for each referring phrase in a selection of the transcript and its ability to specify a computer simulation of the decision process.

The first step accomplished in the specification of a computer simulation of the reasoning process was to encode the knowledge employed by the expert physician in making his decision. Sections of the transcript containing decision making material were reviewed and the knowledge that they contained was translated into a rule language. The basic structure of the language is the rule, which acts to link antecedent conditions to actions based on the rules of those antecedent conditions. The rules take the form of

```
IF <ANTECEDENT CONDITIONS> THEN <ACTION>.
```

The process was completed for the each of the decisions focused on by the physician-subject. The next step in the simulation process is described below.

Comparison to decision analysis: We have compared the decision process employed by our physician-subject 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 decision analysis but to employ a significantly different processing algorithm. Based on this work, we have submitted an abstract for presentation to the Society for Medical Decision Making at its sixth annual meeting in November 1984 [6].

Application of Results to the Teaching of Clinical Medicine: The research we are undertaking as part of this effort might enhance our ability to teach expertise to clinicians at all levels. In a recent publication [4] we reported just such an effort, based on the principles identified in descriptive research on clinical problem solving. In this paper, we showed that clinical medicine can be taught to junior and senior medical students and to house officers by a method that follows an iterative, hypothesis-based approach. This method is a direct derivative of earlier research of ours and others on human problem solving and on clinical problem solving in particular. In recent years this prospective hypothesis-based method also has been applied to postgraduate education at the annual meeting of the American College of Physicians.

We expect that the efforts of this research will produce similar insights into decision making under conditions of uncertainty, in particular those decisions requiring tradeoffs between the risks and benefits of tests and treatments. We also expect that these insights can be incorporated into the teaching of clinical medicine.

2.2. Completed Work

Transcript Analysis: Our work to date has involved an extensive analysis of the behavior of one expert physician (pulmonologist) making management decisions for a desperately ill patient with preleukeminia, chest pain, fever and an acute pulmonary infiltrative process. The problem involves choosing between empiric treatment, no treatment and employing potentially dangerous testing to guide treatment. The uncertainty in etiology of the underlying disease process, the high risk associated with gathering further information and the urgent need for specific treatment of the immunocompromised patient engenders decision making with probabilistic reasoning and employment of techniques to assess multi-attribute utilities. Research in transcript analysis proceeded as described in the following sub-sections:

Segmenting the transcript: The verbatim transcript was first segmented into lines and paragraphs. This 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. This particular transcript was segmented into over 1100 lines. The transcript was then reviewed to identify the sections that contain problem solving material. Subsequent steps in the analysis deal only with those sections that contained problem solving material.

Determining the conceptual framework of the reasoning process: Each line of the transcript was examined to identify the domain object being referenced. These object phrases are distinct from the wording used to refer to them.

Each object phrase was then categorized, resulting in a broad list of elements that the expert was reasoning about. This list of elements was then grouped and further categorized to establish a list of conceptual framework elements.

Determining the content of the knowledge being used: Each line of the transcript was then reviewed to identify the assertions made about each domain object. We assume that the content of these assertions constitutes at least some of the knowledge employed by the expert physician.

Characterizing the reasoning process: With the transcript analyzed in terms of domain objects and relationships among domain objects, the progress of the decision process was analyzed. The reasoning process employed in a specific problem solving section was then matched against problem solving methods derived from research in Artificial Intelligence (AI). When a specific AI method was seen to fit a particular section, we then examined other sections of the transcript to determine if the same AI method characterized the reasoning process in the new

configurations. Dr. J. Hollenberg has implemented a small prototype artificial intelligence program that can help its user to define new decision trees. That program represents typical decisions, potential outcomes, and probabilities and utilities in a narrow domain of medicine, and applies a frame-instantiation algorithm to guide the user through the process of decision-tree construction. Mr. M. Wellman, working with Prof. P. Szolovits and Dr. Pauker, has been studying the development of multi-attribute utility models, and has begun the design and implementation of an artificial intelligence program that will assist an expert decision-analyst in the construction of new multi-attribute utility models.

1.4. Institutional Arrangements and Plans

As in the past, we have been successful in integrating a complex set of related but distinct research efforts at three separate institutions. This has proven possible because Prof. Kuipers (from Tufts University) is also a Visiting Scientist at the MIT Laboratory for Computer Science, because Prof. Szolovits, Dr. Long and Mr. Wellman regularly visit Drs. Pauker and Kassirer's laboratory and fellows, and Drs. Hollenberg and Moskowitz have taken courses at MIT and interacted frequently with other members of the Clinical Decision Making Group.

We plan to continue in the coming year with very similar arrangements. However, Prof. Kuipers will join the MIT Laboratory for Computer Science as a part-time Research Associate, leaving his position at Tufts University. He will also share an appointment at Tufts University Medical School. In addition, Dr. Long has begun to spend more time at Tufts/New England Medical Center because we have obtained new funding for a project in the management of Congestive Heart Failure, the research on which is to be performed in collaboration with Dr. Pauker and another group of physicians at Tufts/NEMCH. Also, Dr. Robert Kunstaetter has joined the Clinical Decision Making Group at MIT as a Research Assistant, and has begun to develop a collaborative project with Dr. G. Octo Barnett of the Massachusetts General Hospital, wherein we plan to apply the explanation methods developed under this project in teaching students at the Harvard Medical School.

2. CRITICAL DECISION NODE PROJECT

2.1. Objectives

To describe the knowledge representations and problem-solving methods employed by physicians making difficult management decisions, involving considerable risk and uncertainty. To investigate the nature of clinical expertise and to characterize the development of probabilistic reasoning by comparing the knowledge and problem-solving methods used across experimental subjects differing substantially in expertise.

4) Give the user increased control over the system's internal decisions and conclusions.

5) Generalize the models of therapy developed in our Digitalis Therapy Program to other applications areas.

Work during the past year in this subproject has had two foci: First, we have previously concluded that goals 1, 4 and 5 of the above list require not only the creation of new techniques of explanation but also fundamentally-improved representations of the knowledge that is to be explained. Therefore, Dr. W. J. Long has, with the assistance of Mr. T. A. Russ and Mr. I. Kohane, developed a set of new knowledge representations and reasoning strategies that will underlie the new explanation capabilities. This work has been carried out in the context of two new larger projects that derive from the Digitalis Therapy program: a program to assist physicians with reasoning about congestive heart failure, and a program for management of ventricular arrhythmias.

Second, Mr. R. A. Granville has created much improved methods of generating English output from the explanation program, exhibiting conciseness and coherence of text. A good explanation depends, however, not only on stylistically acceptable English. One particular area we have identified as crucial to further improvements is to tailor what the program says to what it believes the user already understands (part of goal 3). During the past year, Mr. Granville has begun to develop user models to help decide what needs to be stated and what can be left implicit.

1.3. Decision Analysis Tools Project

Our group has had many years' experience in developing and applying the formal methods of decision analysis to a large variety of clinical decision tasks. Within the past five years, the widespread availability of small, inexpensive, but powerful computer workstations has made possible the widespread dissemination of these methods and the provision of necessary computer support for their effective application. The further problems facing the use of this method are partly pragmatic—the cost and difficulty of use of the hardware and software support environment—and partly fundamental—the difficulty that potential users without extensive formal decision-analytic training have in building new decision-models for novel clinical problems.

This subproject has concentrated (during the past year) on both the pragmatic and fundamental aspects of the problem. Dr. S. G. Pauker and his colleagues at Tufts/New England Medical Center have improved the Decision Maker computer program, adding new capabilities for cost-effectiveness calculations and Markov modeling, and making it available in less expensive and more powerful

methods of decision analysis may be brought to bear. (Indeed, making practical the application of just this methodology is the focus of our Decision Analysis Tools project.) Often, however, critical decisions involving uncertain outcomes and imprecisely-understood risks must be made, when the knowledge required by our formal techniques is simply not available.

In this subproject, we have turned our attention to a formal analysis of the decision-making strategies and knowledge of human experts faced with just such problems. Our goal is to understand what these experts do in the face of a serious lack of hard data, and to develop a sufficient understanding of their successful strategies that we may develop computational methods that will enable future AIM programs to use similar methods.

Dr. J. P. Kassirer and Dr. A. J. Moskowitz of the Tufts/New England Medical Center and Prof. B. J. Kuipers of Tufts University have collaborated in the extensive analysis of the patient management decisions of one expert physician facing a very seriously ill patient. To do this, they have developed a formal six-step methodology for the analysis of verbal protocols taken from their subject, and they have begun to apply the methods developed therein to improve the teaching of medical expertise to clinicians at all levels.

1.2. Explanation and Justification Project

An innate part of the consultation process is the user's ability to explore the reasoning on which the expert bases his advice and to argue the appropriateness of the data and assumptions on which that advice is based. We outline five research goals that would move the AIM field toward the ability to build programs that meet this requirement:

1) Develop and implement new strategies of explanation by improving the user's access to the program's knowledge of

   - the causal and temporal relations among disease states, pathophysiology, observable signs and symptoms, and drug pharmacokinetics, and

   - internal decisions made by the program in the course of its deliberations.

2) Test the utility and acceptability of explanation strategies.

3) Develop models of the user's present knowledge and what he or she wants to know.

1. OVERVIEW AND SUMMARY

Our overall objective is to develop improved methods of computer-based medical reasoning, to better understand the reasoning of human expert clinicians, to develop means for better explaining the knowledge and methods of the computer to its potential users, and to improve the application of decision-analytic reasoning to clinical problem-solving tasks. Each of these objectives matches a deficiency in current AIM programs; thus, through this research effort we seek to enhance the related capabilities of future AIM programs.

This project consists of three related sub-projects, focusing on the following goals:

1) To study and elucidate the knowledge representations and problem-solving methods employed by physicians making difficult management decisions, involving considerable risk and uncertainty.

2) To develop new methods that allow expert programs to explain and justify their conclusions by arguing from fundamental medical facts and principles and reconstructing the path by which those bases have led to the program's recommendations.

3) To continue development and enhancement of a micro-computer-based decision-analysis and sensitivity analysis system for clinical use by physicians.

A summary of the accomplishments of each of these sub-projects is presented here, and a more thorough discussion of each sub-project and its plans for the coming year follow in the subsequent three sections.

1.1. Critical Decision Node Project

Throughout the past decade of significant progress in the creation of artificial intelligence programs for medical applications, the study of how human expert medical practitioners make clinical decisions has played a major role in suggesting the methods and knowledge representations that could be used by these programs. Equally important is the fact that such studies also help to make explicit just what the knowledge of these expert clinicians is, thus helping to teach their skills to new generations of doctors.

One of the most difficult areas of medical decision-making is that where decisions rest on a complex interplay between competing risks and benefits. If all known options open to the decision-maker are explicitly known, if all possible consequences of each decision can be foreseen, and if the likelihoods and costs and benefits of each of these outcomes can be assessed, then the normative

representation scheme to allow more convenient representation of time series processes and to allow the convenient performance of cost- effectiveness analyses. We have also begun to rewrite our manual and develop a library of sample analyses. The original versions of the program have been distributed to some 30 users and groups.

4.2. Progress Report

The work accomplished in the past year differs slightly from what we originally expected. First our target machine is no longer the Apple III. Since that time the IBM-PC and IBM-XT have become the dominant force in the microcomputer marketplace. We have therefore chosen these machines as our target. Furthermore the microcomputer market has been sufficiently shaken out that we did not feel compelled to maintain machine independence by using UCSD Pascal. Another major reason for exploring the feasibility of abandoning that environment is because its purchase cost ($600) was deemed to be a significant obstacle to program distribution. We initially had difficulty moving to other Pascal environments because the size of the required longest path object code overlay exceed one machine segment (64K). We have recently identified two languages with allow use of extended RAM and were therefore feasible programming environments—IQLISP and Turbo Pascal, version 2.0.

The Turbo and UCSD Pascal versions are nearly identical. The advantage of Turbo, however, is its use of full RAM (up to 640K), allowing far larger trees structures and, more importantly, far deeper recursion depths. In this version, we have incorporated a form of cost-effectiveness analysis which allows the use of dual utility structures with an average of only 15% increase in processing time (compared to a 100% increase under more standard schemes). The concept involves recognition that all features of a cost-effectiveness tree are duplicated, save for the second utility structure. Rather that fold the tree back twice, we developed a new expression processor that evaluates and expression and its shadow, or alternate structure. The sum of probabilities x utilities then becomes two sums: probability x utility1 and probability x utility 2. Thus all linking of tree structure and probability evaluation (70-80% of processing) is performed only once.

We have also developed a representation for Markov processes (submitted for presentation at SMDM meeting, 1984). The concept is that a Markov node is an outcome descriptor (like a terminal node) but is a more complex process than simple expression (utility) evaluation. Each utility is now an incremental utility, in three flavors: initial, tail, and all others. Transition probabilities can be arbitrary time dependent expressions. In the UCSD and Turbo Pascal versions, the number of states can be very large (over 100) but processing is quite slow. In the IQLISP version, all evaluations are compiled into BASIC generated object code, but the

compiled expressions are very large. Compilation is lengthy (minutes to hours) but running time is very fast. The number of states possible in this version is limited to 20-25.

The IQLISP version of the program is basically a compiler that does symbolic tree evaluation. It generates large expressions that are analyzed for common sub-expressions and are then passed to the BASIC compiler for object code generation. The object code is very rapid in execution and includes a variety of advanced graphic displays, but the lengthy compilation-and-test cycle makes it cumbersome for tree development and debugging.

We have also been revising our manual. Preliminary distribution of an earlier program version and manual has pointed out many inadequacies in that manual. Clearly the additional features and new machine environment also requires that a new document be created.

Because the grant award has not included funds for evaluation and testing of the program, we have not been developing formal protocols for data collection by collaborating users. Nevertheless, we have been developing broad experience from program utilization in our own division and at a limited array of other user sites.

4.3. Research Plan

In the next year, we plan to complete the manual and to develop a further array or worked examples and template analyses, including Markov and cost-effectiveness techniques. We shall also continue to refine the IBM-PC versions of the program. At this point, it is not clear whether the IQLISP version will continue to be feasible. That language environment is moderately buggy and the time intensive issue of symbolic compilation is difficult to manage. We shall continue to work in a compiled object code environment, either in Turbo Pascal or in C (Lattice version). We shall support processors with and without the 8087 coprocessor. We will try to develop a set of graphics routines (for use with both monochrome and color graphics display boards) to allow more natural tree display. We shall also explore the utility of using color graphics to display multiway sensitivity analyses and shall try to support a standard four or six color plotter, such as the Hewlett-Packard or the SweatPea. We are also beginning to explored to feasibility of using windowing techniques and the "mouse" as an input device.

At this time, we have not found efficient means of compiling dual utility analyses (such as cost-effectiveness analyses) in the IQLISP version. We shall explore these possibilities. We shall also continue to distribute the developing program to interested collaborators so we can evaluate its utility.

References

1. Carberry. S. "Tracking User Goals in an Information-Seeking Environment," in *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, Washington, DC, August 1983.

2. Granger, R. H., Eiselt, K. P. and Holbrook, J. K. "STRATEGIST: A Program that Models Strategy-Driven and Content-Driven Inference Behavior," in the *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, Washington, DC, August 1983.

3. Grice, H.P. "Logic and Conversation," in Syntax and Semantics: Speech Acts, Volume 3, Cole, P. and Morgan, J.L. (eds.), Academic Press, New York, 1975.

4. Kassirer, J. P. "Teaching Clinical Medicine by Iterative Hypotherapy Testing," *New England Journal of Medicine*, 209, (October 13, 1983), 921-923.

5. Mann, W. C., Bates, M., Grosz, B. J., McDonald, D.D., McKeown, K. R. and Swartout, W. R. "Text Generation: The State of the Art and the Literature," Technical Report ISI/RR-81-101, Information Sciences Institute, Marina del Rey, CA, 1981.

6. Moskowitz, A.J., Kassirer, J.P. and Kuipers, B.J. "Clinical Reasoning versus Decision Analysis," to be presented at Society for Clinical Decision Making Meeting, November 1984.

7. Pauker, S.G., Gorry G.A., Kassirer, J.P. and Schwartz, W.B. "Towards the Simulation of Clinical Cognition. Taking a Present Illness by Computer," *American Journal of Medicine*, 60, (1976), 981-996.

8. van Melle, W. "A Domain-Independent Production-Rule System for Consultation Programs," *Proceedings of Sixth International Joint Conference on Artificial Intelligence (IJCAI-79)*, Tokyo, Japan, August 1983, 923-925.

9. Pollack, M. E., Hirschberg, J. and Webber, B. "User Participation in the Reasoning Processes of Expert Systems,' *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, Washington, DC, August 1983.

Publications

1. Asbell, I.J. "A Constraint Representation and Explanation Facility for Renal Physiology," MIT/LCS/TR-318, MIT Laboratory for Computer Science, Cambridge, MA, June 1984.

2. Burke, C.G., Carrette, G.J. and Eliot, C.R. "NIL Reference Manual," MIT/LCS/TR-311, MIT Laboratory for Computer Science, Cambridge, MA, January 1984.

3. Church, K. and Patil, R.S. "Coping with Syntactic Ambiguity or How to Put the Block on the Box on the Table," *American Journal of Linguistics*, 8, (1983), 139-149.

4. Granville, R.A. "Cohesion in Computer Text Generation: Lexical Substitution," MIT/LCS/TR-310, MIT Laboratory for Computer Science, Cambridge, MA, December 1983.

5. Kuipers, B. "The Cognitive Map: Could It Have Been Any Other Way?" in Pick, H.L., Jr., and Acredolo, L.P. (eds.), Spatial Orientation: Theory, Research, and Application, Plenum Press, New York, 1983.

6. Kuipers, B. "Modeling Human Knowledge of Routes: Partial Knowledge and Individual Variation," in *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, Washington, DC, August 1983.

7. Kuipers, B. and Kassirer, J.P. "How to Discover a Knowledge Representation for Causal Reasoning by Studying an Expert Physician," in *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, Karlesruhe, West Germany, August 1983.

8. Kuipers, B. "Programs that Understand How the Body Works," in *Proceedings of the Second IEEE Computer Society International Conference and 1983 Stocker Symposium on Medical Computer Science and Computational Medicine (MEDCOMP-83)*, September 1983.

9. Long, W. J. and Russ, T. A. "A Control Structure for Time Dependent Reasoning," *Proceedings of International Joint Conference on Artificial Intelligence 1983 (IJCAI-83)*, Karlesruhe, West Germany, August 1983.

10. Long, W.J. "Reasoning about State from Causation and Time in a Medical Domain," *Proceedings of the American Association for Artificial Intelligence 1983 Conference (AAAI-83)*, Washington, DC, August 1983.

11. Long, W.J. Russ, T. A., Locke, W. and Bucke, "Reasoning from Multiple Information Sources in Arrhythmia Management," *Proceedings of IEEE Frontiers of Engineering and Computers in Health Care 1983*, September 1983.

12. Long, W.J. "Causal Reasoning in a Physiological Model as a Computational Paradigm." Proceedings of IEEE Conference on Medical Computer Science (MEDCOMP 83), October 1983.

13. Long, W.J. "Potential of Artificial Intelligence in the Use of Electrocardiographic Data," *Computerized Interpretation of Electrocardiogram IX*, June 1984.

14. Martin, W.A., Church, K. and Patil, R.S. "Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results," in Bolc, L., (ed.), Natural Language Parsing Systems, Macmillan Press, London, 1984.

15. Patil, R.S. "Role of Causal Relations in Formulation and Evaluation of Composite Hypotheses," *Proceedings of the IEEE Conference on Medical Computer Science (MEDCOMP-83)*, Burr Oaks, OH, September 1983.

16. Patil, R.S., Bromley, H. and Widman, L. "Causal Understanding of Patient Illness in Medical Diagnosis," in *Proceedings of the IEEE Conference on Medical Computer Science (MEDCOMP-83)*, Burr Oaks, OH, September 1983.

17. Szolovits, P. "Using Artificial Intelligence Models of Medical Decision Making in Medical Education," in Meeting the Challenge: Informatics and Medical Education, Pages, J.C., Levy A.H., Gremy, F. and Anderson, J. (eds.), Elsevier Science Publishers B.V. (North Holland), 1983, 271-281.

Theses in Progress

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

Talks

1. Kuipers, B. Member of panel, "Deep Models, Qualitative Reasoning, Compiling From Deep Models, Anatomical and Physiological Reasoning," Artificial Intelligence in Medicine Workshop, Ohio State University, Columbus, OH, June 2, 1984.

2. Kuipers, B. Member of panel, "Cognitive Psychology and AIM," Artificial Intelligence in Medicine Workshop, Ohio State University, Columbus, OH, June 1, 1984.

3. Kuipers, B. Member of panel, "Computers and Education: A Technological Fix?" (Respondent to a presentation by Joseph Weizenbaum.) Tufts University All-University Forum, Medford, MA, February 14, 1984.

4. Kuipers, B. "Qualitative Causal Reasoning for Second-Generation Medical Diagnosis Programs,"

   University of Minnesota Computer Science Department, Minneapolis, MN, March 30, 1984.

   School of Computer and Information Sciences, Georgia Institute of Technology, Atlanta, GA, April 5, 1984.

   Department of Computer and Information Science, University of Massachusetts, Amherst, MA, April 12, 1984.

   Department of Computer Science, University of Texas, Austin, TX, April 16, 1984.

   Department of Electrical Engineering and Computer Science, University of California at San Diego, La Jolla, CA, April 18, 1984.

   Department of Computer Science, Boston University, Boston, MA, April 20, 1984.

5. Kuipers, B. "Studying Experts To Learn About Qualitative Causal Reasoning," Stanford University Computer Science Department, Stanford, CA, February 24, 1984.

6. Kuipers, B. "Knowledge Representations for Causal Reasoning," MIT

Center for Policy Alternatives and Technology and Policy Program, Cambridge, MA, October 28, 1983.

7. Long, W. "Medical Applications of Artificial Intelligence," 19th Annual Fall Conference of Suburban School Superintendents, Massachusetts Institute of Technology, Cambridge, MA, November 2, 1983.

8. Patil, R. Panel member, "Feasible and Infeasible Expert System Applications," New England Association for Artificial Intelligence, October 1983.

9. Patil, R. "Reasoning Methods in Medical AI Programs," 6.001 summer course, MIT, Cambridge, MA, January 1984.

10. Patil, R. "Role of Physiology in Medical Reasoning" Annual conference of the Canadian Society for Computational Studies of Intelligence, London, Ontario, May 1984.

11. Patil, R. "Research Progress in the MIT Clinical Decision Making Group," 1984 Workshop on Artificial Intelligence In Medicine, Ohio State University, Columbus, OH, June 1984.

12. Patil, R. "ABEL: Acid-base and Electrolyte Project" 1984 Workshop on Artificial Intelligence In Medicine, Ohio State University, Columbus, OH, June 1984.

13. Russ, T. "A Control Structure for Time Dependent Reasoning," paper presented at International Joint Conference on Artificial Intelligence 1983 (IJCAI-83), Karlesruhe, West Germany, August 12, 1983.

14. Szolovits, P. "Experience with the OWL Knowledge-representation System," Workshop on Knowledge Representation, Santa Barbara, CA, October 1983.

15. Szolovits, P. "Reasoning Methods in Medical Artificial Intelligence Programs," Nihon Digital Equipment Corporation User's Group Meeting, Tokyo, Japan, November 1983.

16. Szolovits, P. "Current Problems in Knowledge Representation Research," Symposium, Musashino Electrical Communication Laboratory, Nippon Telephone and Telegraph Co., Tokyo, Japan, November 1983.

17. Szolovits, P. "Overview of AI Research at MIT," Seminar, Systems Development Laboratory, Hitachi Ltd., Kawasaki, Japan, November 1983.

18. Szolovits, P. "Integrating AI Decision Methods with Medical Database Systems," Symposium on the HELP System, Salt Lake City, UT, January 1984.

COMPUTATION STRUCTURES

Academic Staff

J. B. Dennis, Group Leader

Research Staff

W. B. Ackerman
W. Y-P. Lim
C. K. C. Leung

Graduate Students

N. B. Bauman
G. A. Boughton
J. D. Brock
T. A. Chu
G-R. Gao
B. Guharoy
T. R. Hegg
S. Jaganathan
K. B. Theobald
T. S. Wanuga

Undergraduate Students

J. Holderle
B. Kartz
D. Kravitz
B. Kuszmaul
B. Lim
E. Lyons
D. Marcovitz
S. Markowitz
T. Tran

Support Staff

P. Sedell

1. INTRODUCTION

The work of the Computation Structures Group concerns new concepts of the basic structure of computer systems, and therefore addresses issues ranging from hardware design methodology, through the consistent specification of machine architecture and user language, to the evaluation of applications for execution on proposed system architectures. Two system designs are currently being pursued: the static dataflow architecture for large scale scientific computation; and the VIM Project which is exploring the application of dataflow principles to a general purpose system architecture. In the development of the static architecture, progress has been made in the architectural design of a practical dataflow processing element, in the experimental design of key LSI components, in performance studies, and in the techniques of program transformation essential to an effective compiler for VAL (the functional programming developed by the group for use with dataflow computers). Work on the VIM Project has centered on the design of representations for the structured data types of VIMVAL (the user language for VIM), the philosophy of type inference and type checking to be incorporated in VIM, and study of the problems of maintaining data integrity through automatic backup. The group is also working on an advanced methodology for the design of self-timed logic circuits in continuation of its previous work in this area.

2. DATA FLOW PROCESSING ELEMENT

The principal effort of the Computation Structures Group is work toward construction of a demonstration dataflow machine for high-speed numerical computations. The goal is to demonstrate a machine that can achieve better than 100 megaflops of performance in executing useful programs compiled from VAL. The work is guided by our experience in the analysis of benchmark programs in application areas from weather modeling to plasma dynamics.

The design problem is simplified by the recent availability of a commercial, high performance floating point chip set comprising a multiplier and an adder. Because of their ability to pipeline successive operations, they are well suited to to the construction of a dataflow processor.

The envisioned machine has 64 processing elements, each built around a pair of floating point chips and capable of at least five megaflops of performance. We estimate that each processing element will hold several thousand dataflow instructions and will include a large local memory for holding the arrays of values that make up the database of an application. We expect the machine will be able to achieve its full potential performance of 320 megaflops for many application codes. Several proposals for the organization of the processing element have been developed. These are being compared and evaluated, and a choice of one design for detailed development will be made during the next year.

3. PROGRAM TRANSFORMATION

The success of a practical dataflow computer for high performance scientific computation hinges on the ability of a compiler to produce effective dataflow programs from high-level source programs. The basic problem is to create target program structures that adapt the degree of parallelism exposed in the source program to the storage capacity of the machine. To do this, the compiler must restructure the program to make the trade-off between time (parallelism), and space (memory) that permits full utilization of the machine. The Computation Structures Group is exploring two avenues toward this goal -- a general approach based on program transformations that apply to any program written in VAL, and a second approach that generates a *pipe-structured* dataflow program for any of a certain class of VAL programs.

In his recent doctoral thesis [2], William Ackerman studied the translation and optimization techniques for VAL programs that deal with arrays. The principal transformation technique proposed for achieving high execution speed on a static dataflow computer is the *spatial interleaving* of arrays and the *unfolding* of the iteration loops that manipulate them. This technique allows the transformed program to be distributed over many processing elements, permitting parallel operation with minimal communication among the parts. This technique works for the *forall* expression of the VAL language as well as the normal sequential iteration loop, and it works for nested loops and multidimensional arrays. It should yield good results over a wide range of machine sizes and problem sizes.

Gao Guang-Rong has studied a class of VAL programs in which the body of the program consists of several blocks of code interconnected without cycles. Each block defines one or more array values in terms of its inputs, and has the form of an iteration or a *forall* expression. Gao has shown that such programs can be transformed algorithmically into dataflow machine code in which the data is pipelined through the instructions at the maximum rate permitted by the architecture. This approach permits calculation of the instruction and data storage required, and therefore the right choice of space versus time may be made to utilize as much of the machine's capacity as possible. The method used to construct pipelined code by augmenting dataflow programs with buffering is treated in earlier work by Gao [8].

Incorporating these techniques into a practical compiler is a challenging task. Yet the functional (applicative) nature of the VAL programming language makes the task much easier than would be the case for a more conventional language such as Fortran. Construction of a program-transforming compiler for real-world application programs will be a major project over the next few years.

4. LOGIC DESIGN METHODOLOGY

The Group has contributed to the study of asynchronous design of digital logic systems for many years [4], and specifically to design techniques based on self-timed concepts of logic design [9]. At one time it appeared that self-timed techniques already developed might be competitive in the context of VLSI design [9]. However, our experimental design of a two-by-two router device in LSI using standard self-timed circuit elements showed that this approach is far from achieving competitive component density.

Tam-Anh Chu has conceived a design method that exploits silicon layout and self-timed principles together to achieve designs that are competitive with conventional techniques [5]. The methodology uses a novel system organization. In contrast to the usual data-path/controller partition of synchronous systems, a system is organized into *data-path*, *state-machines*, and *distributed control structures*. Self-timed data-path circuits are built using logic gates, pass-transistors, and matched timing delays. Such an approach yields layouts of almost the same area as for synchronous construction, and is possible because in VLSI systems, delays of data-path components can be easily matched by using identical geometry. Also, data-path components have their own controllers for inter-module timing synchronization. State-machines and distributed control structures are used together to form the control part of the system. State-machines test predicates only to guide control flow in the distributed control, and are simple and fast.

A self-timed router chip [6] has been designed using this approach. The router is a packet switching device with two input and two output ports. It decodes address bits of byte-serial input packets and routes them to designated output ports by setting up and maintaining a link between ports throughout the duration of a packet transmission. Packets going to different output ports can be processed concurrently, and any contention for output ports is resolved using arbiters.

The methodology uses a graph model called the Signal Transition Graph for the direct synthesis of self-timed circuits. A Signal Transition Graph is a directed graph where vertices represent signal-transitions at circuit nodes, and arcs specify static and dynamic relationships between pairs of transitions. Additional constraints are included in the graph to permit determination of the circuit (an interconnection of components), and the logic functions of the components. Two self-timed chips have been successfully designed using this graph model -- a router chip and a ring buffer chip.

5. INTEGRATED CIRCUIT DESIGN AND FABRICATION

Since the use of custom integrated circuit devices is essential to achieving competitive performance in a dataflow processor, we have developed several experimental chips using CAD tools available at MIT, and the DARPA MOSIS fabrication service. The devices chosen for experiment reflect key requirements in the construction of practical dataflow computers. One is the two-by-two router which is a building block for packet switched routing networks. The second is a simple dataflow processing element with 8-bit data words that has given us experience in implementing all the essential mechanisms of a full-scale dataflow processor.

The first device constructed [5] was a scaled down router with a two-bit data-path. It was fabricated in 1983 using a 5-micron NMOS process. We have obtained eight chips from MOSIS, all of which were found to be fully operational at the relatively slow rate of around 0.7MHz. This router was designed using self-timed circuits with extremely conservative expectation of process variations. Dual-rail coding was used in the data-path and control circuits to ensure fully speed-independent operation of all components. Also, a substantial amount of circuitry was included for test purposes, and this partly degraded the performance of the router. In retrospect, this project was successful in that it allowed verification of the design approach and produced valuable feedback.

Our experience with this design inspired development of the new design methodology described above. Application of the methodology to the two-by-two router yielded a device [6] with a full 9 bit data-path and buffer storage for eight bytes at each input port. It has been fabricated through MOSIS using a 3-micron CMOS process, and its speed is estimated to be 5MHz. In this version, control circuits are speed-independent and data-path components are timed by means of matched delays. All modules are designed so that times for the reset phase of the four cycle signaling protocol are reduced to a minimum, yielding a significant overall speed improvement. The layout of the data-path in this circuit is similar to a synchronous one. We believe the layout of the control is superior to that of the synchronous design, as the distributed control structures uses less circuitry than a central controller, and they may be laid out to the same pitch as the data-path.

The simple dataflow processing element was an exploratory design developed as a course project in 1983. It has been redesigned and submitted to MOSIS for fabrication. This chip was implemented in bulk CMOS with 3 micron features. When combined with a commercial 2K by 8 RAM chip it forms a simple processing element that can handle the sequencing of 64 dataflow instructions. Most of the chip area is occupied by an eight-bit arithmetic-logic unit/register array, and several PLA structures that implement the control functions. The unique element is a 64-bit

enable memory having an integral priority encoder that implements the dataflow instruction sequencing rules. In a full-scale dataflow processor, the size of the enable memory will require that it be fabricated as one or more dedicated chips. Such a specialized device will be the next candidate for experimental design.

6. PERFORMANCE STUDIES

Being very different from conventional computer systems, dataflow computers require new approaches to performance evaluation. One important aspect is the manner in which the instructions of a dataflow program are assigned to the processing elements of the machine so that full performance may be achieved. This aspect is addressed primarily as a problem of program structure, and compiler analysis and transformation of programs. The other aspect of performance of the static dataflow multiprocessor is the manner in which the load placed on the routing network by the processing elements interacts with the protocol of the routers to determine the rate at which data flows among the processing elements.

In most of our work we have focused on the packet routing network as an independent subsystem. In his doctoral thesis [3] G. Andrew Boughton examines the design of high throughput packet switched networks for interconnecting the modules of a large digital system such as the processing elements of a dataflow computer. The design of such networks is studied under two different sets of assumptions about the implementation technology. The first set corresponds roughly to present technology where only a small number of network nodes can be placed on a single integrated circuit chip. The second set corresponds to future VLSI technology where a large number of network nodes can be placed on a single chip.

Under the first set of assumptions, network structures based on the indirect n-cube topology are studied to determine the factors limiting the performance of very large networks. For this purpose the load on the network is assumed to be generated by independent packet sources at each input port with uniformly distributed packet destinations. For such sources, the strongest constraint examined still allows the throughput of the network to grow linearly with the number of network inputs. However, we show that conditions can arise in the operation of moderately large networks (around a thousand inputs) such that some network inputs accept packets at less than half the average input rate for a long period of time. The design of networks where the sources generate a nonuniform distribution of packet destinations is examined briefly.

For network implementations where many nodes are built into a VLSI chip, we derive a minimum wire cost proportional to the square of the number of network inputs, shown for networks capable of high performance for certain uniform patterns

3.2. Packet Trains for Network Workload Modeling

For network modeling and simulation, it is essential to know the right model for packet arrivals. The traffic measurements on the LCS token ring show that the packet arrivals do not follow the commonly used model of Poisson arrivals which assumes that packets arrive independently and that the arrival of a packet gives no clue about future arrivals. A more realistic model called *packet trains* has been proposed by Raj Jain, in which all packets of a file transfer form a train. The inter-packet interval between successive packets of a train has a distribution very different from inter-train interval. The former is a characteristic of the higher-level protocols and system configurations while the later depends solely on user behavior. After the arrival of a locomotive (the first packet) of a train subsequent packet arrivals can be predicted with very little variance. Protocols that exploit this dependence of packet arrivals, e.g., reservation sharing, can be designed to use resources more efficiently than current ones which are optimized for random Poisson traffic.

3.3. High-bandwidth Residential Networks

A second network technology effort is just beginning this year: exploring the use of commercial cable television systems as a high-bandwidth local network technology that can reach the home. The primary progress on this topic has been to develop, in concept, techniques that can deal with the analog environment of the typical CATV system. It now appears that a promising approach is the use of spread-spectrum modulation techniques, for four reasons. First, spread-spectrum provides compatibility with miscellaneous services already in place on the same cable. Second, the anti-jamming properties of spread-spectrum modulation provide a counter-measure to deal with extreme interference from short-wave broadcasts in the frequency bands available for two-way data communications. A third property, that spread-spectrum signals are well hidden from non-spread spectrum receivers, may also be useful in allowing data signals to use radio frequencies on the cable that have been set aside to protect nearby aircraft communications. Finally, the spread-spectrum code-division multiple access technique may be an effective alternative to carrier-sense or token-passing as a channel access control technique. The theoretical properties of spread-spectrum modulation for this application all are sufficiently promising that it is time to begin a more detailed study, perhaps undertaking a modem design in the coming year.

review, the proNET ring is a commercial version of a 10 Mbit/sec token ring local area network designed by this research group and reported in detail in previous reports. Its primary innovative feature is a star-shaped topology with a passive wiring center intended to make maintenance easy and availability very high. The LCS installation currently covers three floors of the building using four interconnected wire centers. Although the installation is mostly done with twisted pair, there is one experimental fiber-optic section. There are 30 Vax 11/750 and 11/780 computers, five LSI-11's, a Bridge CS/1, and 3 IBM PC's connected to the ring. The Bridge CS/1 and several of the LSI-11's act as gateways to other local area networks, chiefly an experimental (3 Mbit/sec) Ethernet, a standard Ethernet, the ARPANET, and a serial-line network.

Preliminary observations of the monitoring station appear in a Master's thesis by Feldmeier, also available as an LCS Technical Memorandum. Some of the more interesting results are the following:

- Packet traffic is between 1 and 2 million packets/day. The ring is thus in production use, a vital link in the Laboratory's resources.

- Load distribution is very similar to that reported by Shoch and Hupp at Xerox, with the difference that the individual nodes at MIT seem to generate about three times as many bits/second/node. Two possible explanations of this more intense traffic are that the MIT site has more remote paging, and that the Vax 11/750 computers are more powerful than the Xerox Alto, so they can make more frequent demands on the network.

- Internetwork traffic is some 40% of the total traffic on the ring. This large number may have important implications for gateway design and generally for internet plans, although it is hard to tell whether it would be similar in other environments with less diverse communications facilities.

- The distribution of packet interarrival times is decidedly non-Poisson: shorter interarrival intervals have much greater than Poisson probabilities. This observation has led Raj Jain to explore a network packet arrival model called "packet trains" reported below.

- "Network unavailable time," which accumulates whenever a token is continuously absent for more than one second, averages 5 to 10 seconds per 24-hour day. Since all reconfiguration and repair of the network at LCS is done without turning off the network or the monitor, this small number suggests that the star configuration is extraordinarily effective in providing high availability.

4) Implement discretionary or non-discretionary controls in the accessible internal resources as needed.

In summary, the gateway authenticates, labels, and maintains information on category sets while most of the rest of the world can go on unchanged.

2.3. Experimental Implementations

We have implemented two components of a mail-relay gateway suitable for inter-organization connections.

Don Gillies implemented a secure mail relay on an IBM Personal Computer as an undergraduate thesis project. The mail relay was inspired by the needs of the Laboratory for Computer Science Headquarters, which had a new office automation program for their Unix system. They wanted to connect to the rest of the MIT computer network, but also wanted to protect their sensitive research accounting data. The same basic software design should be usable in the planned MIT-to-IBM mail relay connection.

The mail relay is built on top of the CSC Group's PC/IP protocol implementation for the IBM PC. It contains a new a multi-connection TCP, user and server SMTP, and a reliable spooling program. It also has security mechanisms that leave audit trails, that detect unusually heavy mail traffic, that halt messages with suspicious ascii characters, and that record attempts to speak unauthorized protocols through the relay. The full design is explained in the thesis.

The second component (also implemented by Gillies) is mail-filtering software running on a Vax. This facility can be tailored to implement higher-level policies (e.g., no transit) for mail traffic that enters the network via the gateway.

By connecting the mail relay to an internal local area network on one side, and a telephone line on the other, we can implement a controlled, inter-organization, mail network. All messages that arrive over the telephone line can be sent to the Vax for higher-level filtering. In addition, we can use a link-level protocol that authenticates the origin of packets that arrive over the telephone line.

3. LOCAL AREA NETWORK TECHNOLOGY

3.1. Ring Network Monitoring Results

The primary progress this year on the ring network research project was bringing into service of a ring network monitoring station, built by David Feldmeier, and the beginning of systematic collection of data on the proNET token ring network. In

2) The administration of most internal networks is intentionally decentralized. Consequently, it is very difficult to assure conformance with new policies such as tight controls on accessibility of internal resources to outsiders.

3) Internal networks grow incrementally by adding connections to other internal networks as well as single machines. It is hard to check if such additions introduce resources into the internal network that do not conform to network-wide policy.

4) In order for users to enforce a security policy they must be educated as to its purpose and operation. Educating all users of a decentralized network is hard to accomplish once, let alone every time an external link is established.

These conditions suggest the use of *non-discretionary* access controls to isolate strictly-internal resources without relying on the discretion or explicit action of strictly-internal resource owners.

However, the controls needed differ from traditional non-discretionary controls in two important ways:

1) We want to control *invocation* of strictly-internal computer-based resources. Most of the literature on non-discretionary controls is concerned with control of information flow only.

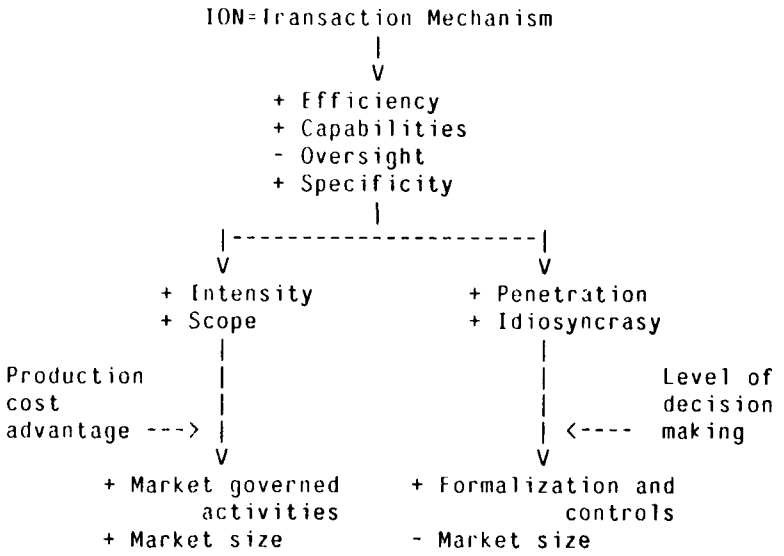
2) We want to protect the *owner* of the service being *invoked*. Most of the literature that does concern non-discretionary controls on invocation is concerned with protecting the integrity of the *invoker* only.

Despite these differences, it seems that with a few modifications, the traditional non-discretionary control policies can be used in this environment:

1) Define special network entry points (gateways).

2) Implement non-discretionary invocation controls on incoming traffic in the gateway(s) using category sets and the following *Intersect* rule: user A can invoke resource B if and only if $\{C\}_a$ *Intersect* $\{C\}_b$ is not equal to $\{\}$.

3) Include authorized outsiders in the category sets of internal resources that the organization wants to make accessible to outsiders. Assign a null-set category set to resources that are to remain strictly internal, so that no one will be able to access them via the gateway (internal users can still get to them since they do not do so via the gateway).



The model is being tested in one domain, industrial and academic research and development laboratories.

2.2. Security Requirements and Technical Mechanisms for Inter-Organization Networks

These interconnections raise new and interesting security requirements. Unlike traditional security requirements, the goal is not to *prohibit* access by outsiders; some outside access is explicitly desired. However, because potential users are outside the boundaries of the organization, they should not be treated as insiders and the potential damage of undesired access is high.

Beefing up security on all internal systems/resources is not an attractive, nor feasible, approach. The set of resources that an organization wants to make accessible is significantly smaller than the set that it wants to remain strictly internal. Therefore, these strictly-internal resources should not be required to take action in order to be protected from external access. Such a requirement is inappropriate for several reasons:

1) Typically, the purpose of an internal network is to facilitate communication and resource sharing. Increased internal usage controls that are tailored to restrict outsiders may interfere with this objective.

1. INTRODUCTION

Rearrangements of Computer Systems Group boundaries occurred in 1983-84, with the result that the area reported under this title is substantially smaller than in past years. The research projects of the group now fall in three categories: inter-organization networks, local area network technology, and network services. The next sections describe these three areas in turn.

2. INTER-ORGANIZATION NETWORKS

During the past year Deborah Estrin continued her doctoral research on the interconnection of computer networks across organization boundaries. This interdisciplinary research encompasses both effects on organizations and new technical requirements.

2.1. Impact on Inter-organization Relationships

When two or more distinct organizations interconnect their internal computer networks to facilitate information and resource exchange, they form an Inter-Organization Network (ION). Ms. Estrin has developed a model of how IONs impact organizations and inter-organization relationships. For a given domain, the model predicts whether IONs favor vertical integration or de-integration (e.g., make or buy, internal development or joint ventures), and market concentration or competition, for example.

Inter-Organization Networks are a new type of transaction mechanism that support new communication and interchange patterns among organizations. The new interchange patterns in turn support new governance structures. Greater intensity and scope allow more activities to be carried out efficiently across the organization boundary (i.e., in the market); in addition, interchange can be managed efficiently with a larger set of organizations (i.e., a larger actual market). However, increased penetration and idiosyncrasy of interchange may lead organizations to adopt more formal governance structures (e.g., contracts) and preclude interchange with an expanded number of market members. The outcome of these antagonistic factors depends upon factors exogenous to the ION: the production cost advantage of carrying out an activity in the market as opposed to internally within the firm; and the level of decision making attention applied to the interconnection. The model is described in the figure below.

COMPUTER SYSTEMS AND COMMUNICATION

Academic Staff

F.J. Corbato
D.P. Reed
J.H. Saltzer, Group Leader
M.V. Wilkes

Graduate Students

D.L. Estrin
D.C. Feldmeier
K. Koile

Undergraduate Students

D.W. Gillies
F.S. Hsu
E. Jaeger
M.L. Lambert
D.J. Karlson
J.L. Romkey

Support Staff

N. Lyall
M.F. Webber

Visitors

R.K. Jain
B.G. Lindsay

11. Lim, W. Y-P. "The MIT Data Flow Engineering Model," IFIP Congress, 9th World Congress, Paris, France, September 1983.

6. 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 January 1985.

7. Wanuga, T. Routing Performance in a Static Data Flow Computer," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected January 1985.

Talks

1. Dennis, J.B. "Design Issues in Functional/Applicative Programming Languages," MIT Data Flow Workshop, Dedham, MA, July 1983.

2. Dennis, J.B. "Non-determinate Computation Using Streams and Guardians," MIT Data Flow Workshop, Dedham, MA, July 1983.

3. Dennis, J.B. "Type-Checking and Inference for VimVal," MIT Data Flow Workshop, Dedham, MA, July 1983.

4. Dennis, J.B. "Architectural Abstraction," Summer Study on Methods for Improving Software Quality and Life Cycle Costs," *NAS Summer Study* Center, Woods Hole, MA, July 1983.

5. Dennis, J.B. "Data Flow Ideas for Supercomputers," Conference on the Frontiers of Supercomputers," Los Alamos, CA, August 1983.

6. Dennis, J.B. "Language-Based Design of Computer Systems," for the short course: "Software-Oriented Computer Architecture," Boston, MA, November 1983.

7. Dennis, J.B. "The TX-O at MIT," The Computer Museum, Marlboro, MA, November 1983.

8. Dennis, J.B. "*Vim*: An Experimental Computer System Supporting Functional Programming," Conference on High-level Language Computer Architecture, Los Angeles, CA, May 1984.

9. Dennis, J.B. Remarks in accepting the 1984 ACM-IEEE Eckert-Mauchly Award, Ann Arbor, MI, June 1984.

10. Gao, G-R. "Maximum Pipelining of Array Operations on Static Data Flow Computers," 1983 International Proceedings on Parallel Processing, Bel Air, MI, August 1984.

Static Data Flow Machine," 1983 International Conference on Parallel Processing, Bel Air, MI, August 1983.

3. Dennis, J.B., Lim, W. Y-P., and W. B. Ackerman, "The MIT Data Flow Engineering Model," *Proceedings of IFIP 9th World Congress*, Paris, France, September 1983.

Theses Completed

1. Ackerman, W.B. "Efficient Implementation of Applicative Languages," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, April 1984.

2. Kuszmaul, B. "Type-checking in VIM-VAL," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

3. Brock, J. D. "Formal Model of Non-determinate Data Flow Computation," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1983.

Theses in Progress

1. Boughton, G.A. "Routing Networks for Packet Communication Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1984.

2. Chu, T-A. "A Design Methodology for Self-timed VLSI Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1986.

3. Gao, G-R. "A Pipelined Code Mapping Scheme for Static Data Flow Computers," Ph.d. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1985.

4. Guharoy, B. "Memory Management in a Static Data Flow Computer System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1985.

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

References

1. Ackerman, W. and Dennis, J.B. "VAL -- A Value-Oriented Algorithmic Language: Preliminary Reference Manual," MIT/LCS/TR-218, MIT Laboratory for Computer Science, Cambridge, MA, June 1979.

2. Ackerman, W.B. "Efficient Implementation of Applicative Languages," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, to appear.

3. Boughton, G.A. "Routing Networks for Packet Communication Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, to appear.

4. Bryant, R.E. "Report on the Workshop on Self-timed Systems," MIT/LCS/TM-166, MIT Laboratory for Computer Science, Cambridge, MA, May 1980.

5. Chu, T-A. "The Design, Implementation and Testing of a Self-timed Two by Two Packet Router," Computation Structures Group Memo 225, MIT Laboratory for Computer Science, Cambridge, MA, February 1983.

6. Chu, T-A. "Design of a Self-Timed Router Using Signal Transition Graphs and VLSI Techniques," to appear.

7. Dennis, J.B. "Data Should not Change: A Model for A Computer System," CSG Memo 209, MIT Laboratory for Computer Science, Cambridge, MA, July 1981.

8. Gao, G.-R. "An Implementation Scheme for Array Operations on Static Data Flow Computers. S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

9. Seitz, C. "System Timing," in An Introduction to VLSI Systems, Chapter 7, Reading, MA, 1980.

Publications

1. Dennis, J.B. "Data Flow Ideas for Supercomputers," *Proceedings of CompCon '84* 28th IEEE Society International Conference, February-March, 1984.

2. Dennis, J.B. and Gao G-R. "Maximum Pipelining of Array Operations on

A program in VIMVAL can be translated into a dataflow graph and the local constraints the program places on the types of its expressions can be described in terms of the class of regular sets which satisfy an operator's type requirements. Type correctness can be defined in terms of the number of regular sets which satisfy every operator's type requirements: a program is type correct if there is exactly one regular set satisfying all the type requirements of all the operators in a program. Type correctness is well defined in terms of mathematical set operations. Type correctness has been shown to be decidable, and an efficient type checking algorithm for VIMVAL has been developed on the basis of this theory.

10. BACKUP AND RECOVERY ISSUES IN VIM

We wish to provide a backup and recovery system for VIM that will guarantee the security of all online data. We envision that incremental backup information will be maintained on some stable storage device -- stable disk -- with older backup information being held on tape storage. The objective is that, following loss of data due to hardware failure, full recovery of the system to the situation prevailing just prior to the fault is possible. That is, users will be able to continue as if no fault had occurred except for the loss of a few characters from terminal input/output buffers.

An interesting approach is made possible by the applicative nature of the VIMVAL language: since reexecution of any procedure that is a function is guaranteed to produce the same results, even in the presence of arbitrary concurrent computations, it is not necessary to back up intermediate states of function evaluation; only the arguments of the invocation need be saved. This concept becomes more intricate as one considers data structures and stream-oriented computation. Of course, *guardians*, the mechanism provided in VIM for implementing nondeterminate computations, must be treated carefully. These issues are the subject of master's thesis research in progress by Suresh Jaganathan.

Therefore we are examining the issues involved in implementing a paging mechanism for VIM with a small page size. The unit of transfer between the main store and the disk is called a *chunk* and its size has been set at 32 words for our present experimental design.

Bhaskar Guharoy has designed representations for the principal structured data types of VIM -- arrays and records -- to permit efficient operation of the storage management system. A large array or nested record structure is represented as a tree of chunks. In this representation, data structures may share common substructures, and the basic operations of accessing an element or creating an altered version of a structure can be done in log(n) steps for a structure of n elements. Furthermore, a full (modified) copy of a data structure may be made in log(n) time.

The design of VIMVAL and its implementation is such that circular structures are never created during system operation. This acyclic property of the data structures permits use of a reference-count based storage reclamation mechanism.

9. THE VIM TYPE SYSTEM

A type system has been developed for the revised version of the VAL programming language (VIMVAL) in a bachelor's thesis completed by Bradley Kuszmaul. The type system is designed so that users of VIMVAL may omit type declarations from their programs whenever the type of an expression can be determined by the compiler. The system combines several ideas that have been found to be worthwhile in advanced language designs. To start with, VIMVAL includes higher order functions, that is, functions are "first class objects" and may be passed as values and built into data structures. The type system allows programs to be written with incomplete type specifications, and the type checker infers the types of expressions from their context. The types of some expressions may remain undetermined even after program compilation. This allows the binding of types to be deferred until the module is linked to other modules to create an executable program. At that point all types must be determined, for we wish to exclude the possibility of discovery of type errors during program execution. This aspect of the type system allows program modules to be "polymorphic" -- a module may perform analogous operations on different types, according to the modules linked to it.

The thesis develops a theory of types which applies to a large class of programming languages, including VIMVAL. The notion of type is defined in terms of regular sets, which describe sequences of legal operations on a value of a given type. Recursive types allow infinite sequences of legal operations, so the definition of types allows infinitely long sequences to be elements of the regular sets describing a type.

of communication. Networks are presented that come close to supporting the desired level of performance using the specified amount of wire. Networks for other patterns of communication are also briefly discussed.

Research is beginning on how the routing network influences performance of the static dataflow machine. The effects of packet traffic flow patterns generated by the program workload on the throughput of the routing network (as well as the rest of the machine) are being examined. From the results of this analysis, possible methods for improving the performance of the routing network are being considered. Initially, the structure of the routing network is being restricted to the indirect binary n-cube, and the program workload is being restricted to the class of pipe-structured programs.

7. GENERAL PURPOSE DATA FLOW COMPUTING: THE VIM PROJECT

The Computation Structures Group is developing an experimental advanced general purpose computing system based on the principles of dataflow computation and functional programming and using the model of computation proposed by Dennis [7]. The prototype for such a system is the VAL Interpretive Machine or VIM. The high-level functional programming language chosen for this project is VIMVAL, a revision and extension of VAL [1]. Our current work on VIM covers three areas: the development of a comprehensive memory system for supporting VIMVAL data structures; development of a simple but general type system for VIMVAL using type inference; and the development of specialized backup and recovery procedures for VIM.

8. VIM STRUCTURES

The physical storage system of VIM consists of a semiconductor main store and a disk. A common virtual address space is provided to all users to facilitate the sharing of programs and data. Ideally, the only information brought from the disk into the main store should be the logical units of information -- such as records and arrays -- which are referenced by the computation. A large page size is undesirable because inefficient memory use results when independent logical units are packed into the same page. In conventional systems a small page size is troublesome because the large page traffic cannot be handled efficiently. In particular, the overlapped execution of several page accesses for one computational process is generally impractical.

In the case of VIM, the concurrency of the dataflow program execution model makes the overlapped handling of page accesses much easier to consider.

4. NETWORK SERVICES

4.1. Personal Computer Networking: PC/IP

Work continued on the development of network programs for the IBM PC. This year, several new programs were written, including one that displays all packets sent on the Ethernet, which is useful for debugging. Other programs placed in service include a file transfer server package, a remote printing user interface, and an interface to the ARPANET Network Information Center name directory. A major program which was finished was an implementation of RVD, the Remote Virtual Disk protocol, for the IBM PC. RVD was developed by LCS to permit shared (read only) disks and large private disks on Vaxes that actually have only small ones. The PC implementation allows ten RVD drives on one machine, making it possible to have tremendous amounts of data accessible. In the current implementation, disk accesses are about as fast as a floppy disk. A drawback of the current implementation is that no network programs (most important, TFTP) can write to an RVD disk because of contention for the network interface. The problems are mostly caused by the unusual structure of the programs and awkward integration of the network with the PC DOS operating system. Another major addition this year was a driver for the proNET ring interface for the IBM PC. John Romkey worked on the programs and the RVD implementation.

A major release of the source language versions of the PC network programs was made, dated February 1. The release included most of the work done prior to RVD, binaries of all the programs, the User's Manual and the Programmer's Manual. It was 2.8 Mbytes long. Copyright messages in the code grant permission to do anything except remove the copyright messages. Over 50 copies have been shipped at a $45 fee that covers the cost of a tape, duplication, and postage. Handling of the distribution was done largely by Muriel Webber.

4.2. On-line Directory System

Kimberly Koile completed a Master's thesis describing the design and implementation of an online directory assistance system called DIRSYS. The system, designed for use by members of the MIT community, has an *incremental* interface that combines features of a paper telephone book with those of a full-screen editor such as Emacs. Each directory entry is displayed in a compact one line per entry format, as are entries in a paper telephone book. Since more information is available for each entry than in a paper telephone book, a command is available for changing entries into a more expanded format in which additional information is displayed.

Users may "browse" through this electronic telephone book by issuing commands

similar to Emacs' cursor motion commands, or they may search for a specific name by typing the name. After each letter that the user types, DIRSYS moves the *highlight*, the means of emphasizing an entry, to the entry whose name string most closely matches what the user has typed so far. This incremental search mechanism is similar to that used in Emacs. The system provides a *help facility* with two levels: the first level reminds users of which commands are available; the second level describes the function of a specified command in detail. A *tutorial* is available for users who want a very detailed description of the system.

Finally, DIRSYS provides a facility for keeping the information in the directory database up-to-date. A user may submit *update requests*, which contain information about modifications to his directory entry, to the DIRSYS manager. When the update requests have been validated, the information contained in them is incorporated into the directory database by an *update daemon*, a program that runs every night to update the database.

A preliminary evaluation of DIRSYS indicates that the system can be used easily by both inexperienced and experienced computer users. Details of the evaluation were described in the thesis. The learning aids guide the novice without encumbering the experienced user. The commands are simple, easy to use, and consistently interpreted. The system provides prompts and polite, informative messages to the user. It also is robust in that it is very difficult to cause DIRSYS to fail to operate. Finally, individuals who used DIRSYS seemed to enjoy using the system, even though the system response was slow at times, and agreed that a directory system such as DIRSYS would provide a convenient service for use both inside and outside the MIT community.

Doris Karlson, in an undergraduate thesis, explored the possibility of a search based on name sounds as an alternative to the incremental search of DIRSYS. A modified version of the Soundex indexing system was tried on the MIT directory database. Statistics from that database suggest two things: 1) A more sophisticated indexing system than Soundex seems necessary for reasonable human engineering, because the Soundex system tends to index too many names that do *not* sound alike in the same category; and 2) The frequency of similar-sounding names in a file of 20,000 entries is large enough that display of more than one name per line, 20 lines per display is required.

4.3. Analysis of Timeout Algorithms for Packet Retransmission

Almost all networking and distributed systems protocols have to cope with the problem of determining when a node should retransmit a packet that has not been acknowledged. A bad timeout algorithm may either flood the network with duplicate copies of packets and lead to unwanted congestion or it may take too long to

retransmit a packet. An analysis of various timeout algorithms shows that during period of congestion (repeated packet loss), most timeout algorithms would either diverge to extremely high timeout intervals, or converge to a rather low timeout value. In either case, the throughput may be reduced to virtually zero or the circuit may be disconnected prematurely. Raj Jain did a simulation study of a variety of timeout algorithms and has compiled a list of guidelines for designing such algorithms and for setting parameter values.

Publications

1. Comer, M.H. "Loose Consistency in a Personal Computer Mail System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1983. Also MIT Laboratory for Computer Science Technical Report, MIT/LCS/TR-316, June 1984.

2. Feldmeier, D.C. "Empirical Analysis of a Token Ring Network," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1984. Also MIT Laboratory for Computer Science Technical Memorandum, MIT/LCS/TM-254, January 1984.

3. Koile, K. "The Design and Implementation of an Online Directory Assistance System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, December 1983. Also MIT Laboratory for Computer Science Technical Report, MIT/LCS/TR-313, December 1983.

4. Estrin, D. and Sirbu, M. "Cable Television Networks as an Alternative to the Local Loop," to appear in *Journal of Telecommunications Networks* (1984).

5. Saltzer, J.H., Reed, D.P. and Clark, D.D. "End-To-End Arguments in System Design," to appear in *ACM Transactions on Computer Systems*, (November 1984).

Theses Completed

1. Comer, M.H. "Loose Consistency in a Personal Computer Mail System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1983.

2. Feldmeier, D.C. "Empirical Analysis of a Token Ring Network," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

3. Frankston, C. "The Amber Operating System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

4. Gillies, D.W. "Improved Network Security with a Trusted Mail Relay," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

5. Karlson, D.J. "Soundex Searching in an Online Directory Assistance System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

6. Koile, K. "The Design and Implementation of an Online Directory Assistance System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, December 1983.

Theses in Progress

1. Estrin, D.L. "Inter-organization Networks," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1985.

2. Jaeger, E. "Third Party Access Control and Accounting Schemes," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1984.

Conference Participation

1. Estrin, D.L. Technology for Meaningful Work, Institute of Policy Studies, Allentown, PA, April 1984.

2. Estrin, D.L. IBM-MIT Workshop on Security, Boston, MA, May 1984.

3. Saltzer, J.H. Chairman, ACM Ninth Symposium on Operating Systems Principles, Bretton Woods, NH, October 1983.

Talks

1. Corbato, F.J. "System Issues in Project Athena," Panel discussion, Ninth ACM Symposium on Operating Systems Principles, Bretton Woods, NH, October 1983.

2. Saltzer, J.H. "Inter-organization Links and Community Networks," presented at ILP Symposium on "Networked Computer Systems," MIT, Cambridge, MA, May 14, 1984.

DISTRIBUTED COMPUTER SYSTEMS

Research Staff

L.W. Allen
D.D. Clark, Group Leader
M.B. Greenwald
E.A. Martin

Graduate Students

R.W. Baldwin
J.C. Gibson
W.C. Gramlich
P. Ng
K.R. Sollins
L. Zhang

Undergraduate Students

D.A. Bridgham
E.D. Crisostomo
T.H. Kim
J. Leschner
M.B.Macaisa
J.L. Romkey
M.A. Rosenstein
H.J. Shinsato
E.H. Siegel
G.D. Skinner
J. Spurlock
C.A. Warack

Support Staff

S.C. Comfort
E.L. Felix

Visitors

A.J. Herbert

1. INTRODUCTION

The Distributed Computer Systems Group is a new group this year, formed from certain members of the Computer Systems and Communications Group and the Computer System Structures Group. The major project of this group has been the development of the Swift Operating System, but there are a number of other projects which are described in the sections to follow.

2. SWIFT

2.1. Project Summary

The Swift Operating System arose out of our earlier research in the implementation of network protocols. Recurring performance problems with protocol software led us to the conclusion that there was an underlying problem more general than specific design flaws in certain protocol suites. Our conclusion was that existing operating systems, such as Unix, failed to provide the correct run-time support for highly interactive parallel software packages such as protocols. Swift is intended to demonstrate that proper support for this kind of software can be easily supplied.

The most novel aspect of Swift is the general structure provided for inter- and intra-process flow of control and information. Swift supports a programming style loosely built around two interrelated concepts, *multiprocess modules* and *upcalls*.

In traditional systems, the supervisor consists of a set of entry points which are invoked by the application program. That is, the application makes a subroutine call to a lower level, which performs some service for the application and then returns. In a network driven environment, most of the actions are initiated, not by the client from above, but by the network from below. Therefore, the most natural flow of control is not down from above but up from below. Most systems support this upward flow of control poorly, using either a very inefficient interprocess message to achieve this upward flow, or a very efficient but unstructured interrupt. Our system permits subroutines to be arranged so that the natural flow of control can either be up from below or down from above. Permitting control to flow in a natural direction eliminates many unnecessary process schedulings within software such as network protocols, and has in some cases permitted a tenfold reduction in the bulk of the code, as well as a tenfold increase in its performance.

The upcall strategy eliminates process switching as a means of invoking a software module such as a protocol layer. Traditionally, a protocol layer would be organized as a separate process, but now it is organized as subroutines which live in a number of processes, each callable as appropriate from above or below. There must thus be a mechanism for these various subroutines to share state in such a way that the

function of a particular module is carried out. For this purpose, Swift uses the operating system mechanism called a "monitor," a shared data object whose lock is managed by the system itself. Subroutines in different processes that collaborate with each other constitute a *multiprocess module*. Swift supports a programming style in which interprocess communication is *never* used as a way for one module to invoke another, but rather interprocess communication is only used within one module, through the mediation of the monitor locks.

Programs written using the philosophy of upcalls and multiprocess modules turn out, if written by sophisticated programmers, to be very simple, short and efficient. However, these tools, in the hands of tasteless programmers, have the possibility of creating the parallel programming equivalent of Fortran "spaghetti code." One of the concerns of our research is developing constraints on the programming style which lead to coherent and readable programs without severely impacting the efficiency and natural structure of the code.

There are two other features to Swift which, along with these new ideas for program structure, distinguish it from other operating systems. The first of these is the support which Swift supplies for real-time support, and the other is the support for management of its address space.

Swift is concerned with tasks such as network protocols, which involves scheduling a number of small computations to run in the 1-10 millisecond region. For this reason, the operating system that supports this task must have something in common with a real-time operating system, rather than a general purpose time-sharing system. Swift contains a task scheduler which assigns a priority to tasks based on the real-time deadline of the task, measured in milliseconds. This rather sophisticated scheduler is integrated into the monitor lock facility, so that a high-priority task encountering the lock set by a low-priority task can promote the low-priority task to run until such time as the lock is released.

The other important feature of Swift is its approach towards management of its address space. There are, historically, three ways toward dealing with the address space in an operating system. The first, typified by Multics, provides a multiple address space environment, with a very rich set of tools for sharing data between these address spaces in a controlled manner. This provided efficient interprocess communication, but required special hardware support which limited the portability of the system. The second, typified by Unix, provided a multiple address space environment with limited tools for sharing between them. The elimination of sophisticated sharing mechanisms meant that there were no special hardware requirements for support of the system, so it could be easily ported onto new machines, but it meant that highly parallel computations were very difficult and inefficient to program on the system. For Swift, neither of these approaches

appealed to us. We could not tolerate the inefficiency of a Unix-like interprocess communication mechanism, and we were not interested in a system with strong requirements for specialized hardware support. Swift thus chose a third option for address space management, which is to put all the computations of the system into a single large address space. Clearly, this requires no special hardware support, and it certainly provides very efficient communication between different tasks. The major drawback of this approach, which is almost overwhelming, is that the broad sharing of a single address space means that program bugs cause corruption of arbitrary parts of storage, leading to crashes which are almost impossible to debug. We thus set ourselves a goal of building a single address space operating system with sufficient restraints that program development and execution would be stable and predictable, even though each task address space was nominally shared with all of the other tasks on the system.

The approach we took to this is to program Swift in a language which uses compile and run-time checking to detect erroneous references to memory and other similar bugs. Specifically, we have implemented Swift in the CLU programming language. CLU performs such tests as bounds checking for array references, and most important it prevents the use of an arbitrary bit-pattern as a pointer, so that references through arbitrary bit-patterns cannot corrupt unexpected locations in memory.

The most insidious bug which can arise in a system such as this is the use of a bad pointer, not because the pointer has been created incorrectly, but because the pointer points to a location in memory which has been de-allocated and reused. This can easily happen in a multi-task environment, if one task believes that an object is no longer needed, while another task continues to use that pointer. The only way to avoid this class of bug is to take de-allocation of storage away from the application and perform it in the system. This is the function called *Garbage Collection*. One of the challenges of the Swift system is to develop a garbage collector suitable for the operating system environment.

2.2. Project Status

A major effort effort over the last year has been the moving of the Swift system from the Vax onto a 68000 processor. This move was necessitated by several architectural limitations of the Vax, in particular the very inefficient I/O structure, by the lack of an all-points-addressable display for the Vax, and by the high cost of the Vax, which prevented the deployment of the machine in suitable numbers. Swift is now running on a 68000-based machine which we have assembled out of commercially purchased cards. In retrospect, a great deal of group effort went into making usable a machine which was somewhat cheaper than would have been ideal: however, Swift is now running and we are proceeding with a number of improvements which were not possible within the Vax version of the system.

Considerable effort has been invested in the design of a suitable garbage collector for Swift. Ideally, our garbage collector would have the following characteristics. First, it is incremental, collecting garbage a little at a time while the system runs, rather than stopping the system while all of memory is examined. Second, it should not move objects around as a part of collecting garbage, because I/O devices may have pointers into memory, which are difficult to find and change arbitrarily. Third, it should not require a large quantity of additional memory to use as the temporary storage area for the garbage collection process.

A number of very creative ideas were proposed for the Swift system, taking into account that the system is intended to run on a desktop workstation or similar node of a networked distributed system. If we truly believed that network communication was extremely rapid, then an obvious way to garbage collect one's environment is to make a snapshot of it and send the snapshot across the net to a cleaning machine which would send back a laundered version of the address space. However, it seems difficult to get the necessary throughput across the net. Certain garbage collection strategies, in particular the one proposed by Dijkstra, seem directly relevant to what we are doing and have been implemented for trial later this year. Perhaps the most novel garbage collector we have proposed is the Probabilistic Parallel Garbage Collector (PPGC). This superficially silly idea involves allowing the garbage collector to run in parallel with the other tasks, without the traditional interlocks. This raises the possibility that, under certain very anomalous circumstances, the garbage collector may declare as garbage something that is not. However, there are ways that this event can be tested after-the-fact, which means that the investment of additional background cycles from the machine can reduce to an acceptable level the probability that we have made an error. Unfortunately, we cannot come up with any analytical model that tells us exactly what the probability of failure is. Thus, we are experimenting with PPGC, under a variety of program loads, to determine under what circumstances errors may arise. It has been difficult to assess the probability so far, since we have not yet had a single garbage collection error due to the probabilistic assumption.

CLU, as it is used as an application programming language at MIT, does not have any dynamic linking capability. That is, one compiles all the programs that are to be part of the current run, then one statically links them, and then one brings this static load-image into execution. Swift requires a dynamic linking capability, and over the last year this has been designed and implemented. We are now testing a dynamic loader which will bring subsystems into a running CLU environment and dynamically resolve all of the cross-module references.

A more difficult problem is unlinking the module and removing it from the environment after it is needed. It could be argued that this step is unnecessary. However, this will cause the size of the link-image to grow continuously, which

means that the memory management algorithm must be more sophisticated than if modules can be thrown out when they are no longer needed. Since we do not wish to make sophisticated assumptions about our virtual memory (since we want the system to be portable onto a wide variety of hardware) and since virtual memory techniques are fairly well understood, we are concentrating our energy on exploring the higher- level question of whether or not programs can be unlinked and removed from the address space of the system, either to suspend them while they are running or to remove them when they are no longer needed. Preliminary design for this approach is now completed.

Most operating systems provide a file system for their users and Swift is no exception. However, we were uninterested in writing the necessary device drivers to permit Swift to support disks. Instead, as part of Swift, we have developed a remote file system protocol, which permits Swift to utilize file systems stored on other machines. We have implemented a simple server for this protocol, and the implementation of the Swift interface is almost complete.

We have implemented a number of test applications on Swift, in order to determine system performance and to stress the garbage collector. Perhaps the largest is the text editor Ted, which is a very popular CLU program on both TOPS 20 and Unix. As soon as the file system is working, we will be able to run a fully usable version of Ted on Swift, which will give us a considerable test of the system's performance.

One of the most interesting applications has always been network protocols themselves, and we are currently rewriting our earlier implementation of the protocol package for Swift, in order to improve its performance, and to get more experience with the proper structure of such programs inside Swift.

2.3. Future Directions

The major implementation effort on Swift will probably finish within the next six months. At that point there will be a major project review to determine what future direction the project will take. There are two important questions which we wish to answer before the project terminates. The first of these is whether we can understand how to structure programs built around the concepts of upcalls and multiprocess modules. These concepts give the programmer great freedom, and it has been noticed in the past that great freedom sometimes leads to very poor programming style. We thus identify a conflict between our desire for the efficiency which comes from the these new tools, and the desire to have constraints that lead to good style. We feel that Swift has taught us some fresh insights about the construction of multiprocess programs, and we are anxious to understand this in as general a manner as possible. Second, we are anxious to learn how effectively we can achieve our goal of supporting reliably a single address space operating system.

In order to do this we must experiment further with garbage collection and with linking and unlinking.

In order to achieve these goals, we feel that it may be necessary to implement additional application programs which utilize the features of Swift. However, the hardware base on which we run is sufficiently unstable that large-scale software development is not tractable. We must, therefore, make a decision as to whether we will move Swift onto yet a third hardware base.

2.4. Related Activities

There are a number of small activities which, although not strictly a part of the Swift Project, are closely associated with it within our group. These include our support of Unix, which is used as the program development environment for Swift, development of a tasking package for the IBM PC which supports many of the same programming conventions of Swift (most particularly upcalls), and the support of the Remote Virtual Disk Protocol.

Remote Virtual Disk (RVD) is a protocol which lets one machine attach to a file on a server machine and to use that file as if it were a disk. Our group initially developed RVD to expand the disk space on our Vaxes, and the tool has now become a widely used facility within the Vax-Unix community of LCS. However, the server which we initially implemented, which was not really intended to provide serious operational support, needs rewriting in order to be sufficiently robust, and the Computational Resources Group has undertaken the task of producing and running a better server for RVD. User programs for RVD exist for Unix 4.2, and for the IBM PC.

3. DISTRIBUTED ARCHITECTURES FOR MAIL

For many users, computer mail has been the most important application of computer networks. The software for distribution and forwarding of mail is very well developed at this point, but the software for displaying, generating and archiving mail is still based on the assumption of a centralized time-sharing machine. The goal of this project is to develop a user interface to a mail system which is suitable for a personal computer model of distributing computing.

In the centralized view of mail, each user is served by a particular machine on which is located the *user mailbox*, as well as the necessary software to manipulate this mailbox. The assumption is that the user directly logs in to the machine on which the mailbox is located. The machine containing the mailbox uses some standard mail forwarding protocol to communicate with other mailbox machines located around the network, in order that mail is properly forwarded. For this reason, the mailbox machine must be available as a server on the network essentially all the time.

Since the mailbox machine must operate like a server, it is not appropriate for a personal computer to be the mailbox machine. A personal computer may be powered-off much of the time, and even if it is physically powered-on and connected to a network, may not be able to run server code as a background job.

We have developed a new architecture for mail software, to cope with the characteristics of a personal computer. Our design divides the traditional functions of a mail processing node into two parts, the management of the mailbox and the execution of the user interface software. These functions are implemented on different machines; the mailbox is stored on a centralized server called a *repository*, and the user interface software runs on a personal computer.

The goal of this research is twofold. First, this project is attempting to produce software that can be used in practice. Several of our existing mail nodes are heavily overloaded, so the ability to read mail on a personal computer would simultaneously remove a burden from our centralized time-sharing systems, as well as providing a mail processing environment which is more responsive and interactive. More importantly however, this research has a goal of exploring how traditional applications should be restructured in the context of a distributed computing environment. Several interesting problems arise, which we cannot yet solve in general, but which we can explore using mail as a particular example.

The first problem is that the personal computer may not have continuous access to the data stored in the repository. In fact, the personal computer may be disconnected from the network much of the time. In particular, we would like to permit the user to send and receive mail at a time when a network connection is not open. This will permit a portable computer to be used as an interface to the mail system. This means that a temporary copy of the user's mailbox must be created in the personal computer, matching as closely as possible the master copy which is stored in the repository. We thus have multiple copies of the database describing the mailbox, which are almost always partitioned, but only occasionally able to talk to each other. Updates can occur to either copy, and the software must do its best to keep all of the versions consistent.

The second problem is that the user may wish to interact with the mail system from a number of personal computers, and he would like to see a consistent view of his mailbox, no matter which personal computer is acting as a front end. This means that in addition to the master copy stored in the repository, there may be several rather than one auxiliary copies, each of which has a slightly different version of the mailbox in it. This, plus an additional requirement that the system must recover whenever a personal computer crashes and loses its copy of the mailbox, makes very difficult the problem of keeping the user's view of the mailbox consistent.

During the last year, we have designed the mailbox repository, the user interface,

and the protocol which hooks them together. The protocol contains some rather sophisticated error recovery mechanisms, so that the connection can be disrupted at an arbitrary point without causing the various copies of the mailbox to diverge in an irreconcilable manner. We have a prototype implementation of the repository, and over the next six months we intend to demonstrate a running repository and a user interface program running on an IBM Personal Computer.

4. NETWORK ROUTING AND RESOURCE CONTROL

Elizabeth Martin continued to study dynamic routing problems in an internet. We have been participating in the development of the Exterior Gateway Protocol, which will be used to pass routing information between gateways in the DARPA Internet, and have proposed some schemes for dynamic routing within the MIT Internet, a subsection of the DARPA Internet.

4.1. Exterior Gateway Protocol

The Exterior Gateway Protocol (EGP) partitions groups of gateways in the DARPA Internet into Autonomous Systems of gateways. The gateways in an Autonomous System (AS) are programmed, maintained, and administered by one organization. Gateways in different Autonomous Systems exchange routing information using EGP. EGP consolidates the amount of routing information that is exchanged and provides a more controlled method of passing information between gateways who may or may not trust each other. Gateways within an AS use their own conventions for passing routing and up/down information among themselves; these gateways are free to experiment with different routing algorithms.

Defining rules for the internet routing topology and for the dispersal of routing information that prevent routing loops has proved to be very difficult. Consequently, the early version of EGP which is expected to become a DARPA standard this summer is very restrictive.

We have participated in specifying EGP and in studying the routing issues this protocol is attempting to address. We have implemented EGP for our C gateway.

4.2. Interior Gateway Protocol

The MITnet consists of about 45 interconnected LANs (called subnets), and is expected to grow to about 200 LANs by 1990. It is becoming increasingly inconvenient to manually change the subnet routing tables in all the MIT gateways every time a new subnet and gateway is added to the MITnet. Therefore, we have specified a subnet routing protocol (our Interior Gateway Protocol) to be used to tell gateways within the MITnet: first, which gateway to use to get to which exterior nets

the study in that, when they achieve a level of mastery of the computer, students will be able to take machines to their own homes. The project is scheduled to run until the end of the 1984-85 school year.

The project is emphasizing use of the computer as a tool in all aspects of the students' education. The children are learning to program in Logo and will also use software packages written in Logo. A key research interest are the different learning styles exhibited by the students and the effects of using the computers on those learning styles.

The day-to-day supervision of the project is being carried out by the project director, David Scrimshaw, working under Dr. Papert. Four undergraduates have been programming Logo software for use by the elementary school students.

There are two teachers at each school involved in the project. The first is the prime teacher, the teacher of the class we will be studying. The second is a backup teacher, a teacher of a neighboring class who will have several computers in her class and will attend all meetings and sessions that the prime teacher attends. The purpose of the backup teacher is to stand in for the prime teacher if needed, to provide support for the prime teacher and to be an additional consultant to the research group on classroom teaching matters.

The teachers, other project members, and a group of university students, educators and non-MIT researchers (totalling 25) have been meeting each week to plan and conduct the research for the project. Most of the classroom observation will be carried out by members of this group.

3.2. The Carroll School

Sylvia Weir has been working in the Carroll School with 11-13 year old learning disabled children and Logo. The fundamental postulate of this work is that a certain segment of the population consists of children with learning deficits that may be characterized as linguistic/analytic, but who may be very talented in terms of spatial reasoning and "perceptual mathematics." These children may well be classified as hyperactive and show learning styles which are very far from "planful" or "top down." In view of these characteristics, standard school settings and curricula discriminate strongly against these children.

This year a set of measures of spatial and perceptual abilities have been developed. They show a clear bimodal distribution among children at the school, indicating a very particular subpopulation of children (those in the high mode) with strengths that are not being capitalized on. Work has begun on developing Logo-based materials and techniques that build on their strengths, rather than keeping

search the journal and return a box containing ports to entries which have specified key words -- constructing another view *at need*, rather than *incrementally* as the BY-TOPIC view is constructed. One could even write a procedure to completely reorganize the journal.

2.3. Future Work

Our planned future work on Boxer includes:

- developing a message passing semantics for Boxer;
- defining the basic graphical objects and operations of the system; and
- making a microcomputer implementation.

3. THE EDUCATIONAL CONTEXT

During the last year, there were three principal sites for working with teachers and students: The Quincy and Ohrenberger Schools in Boston (High Density Project), the Carroll School, and MIT itself. In addition to this work, we hosted the first National Logo Conference with over 450 participants from more than 20 countries (June 27 - 30). The conference highlights the major impact our work has had on education at the national and international scales.

3.1. The Quincy-Ohrenberger High Density Project

Computers are becoming a common sight in today's classroom. There are two reasons for this. The first is that as they become more common in society, there is a greater perceived and actual need to teach people how to use them. The second and more powerful reason is that computers can be used to dramatically enhance the learning of essentially all other subjects. There will come a day when every school child has a computer. Before this happens it is important to look ahead to explore the possibilities and anticipate the problems posed by such a high density of computer power. The Quincy-Ohrenberger High Density Project will explore these questions.

This project, under the direction of Seymour Papert, is studying the use of computers and Logo by elementary school students in two Boston elementary schools. A class of 22 third graders at the Ohrenberger School in West Roxbury will use 11 computers, and a class of 20 fourth graders at the Quincy School in Chinatown will use 20 computers. This will produce a higher density of computers per child than any that has been tried in an elementary classroom before. The computer to be used is the Coleco Adam. There will also be a home component of

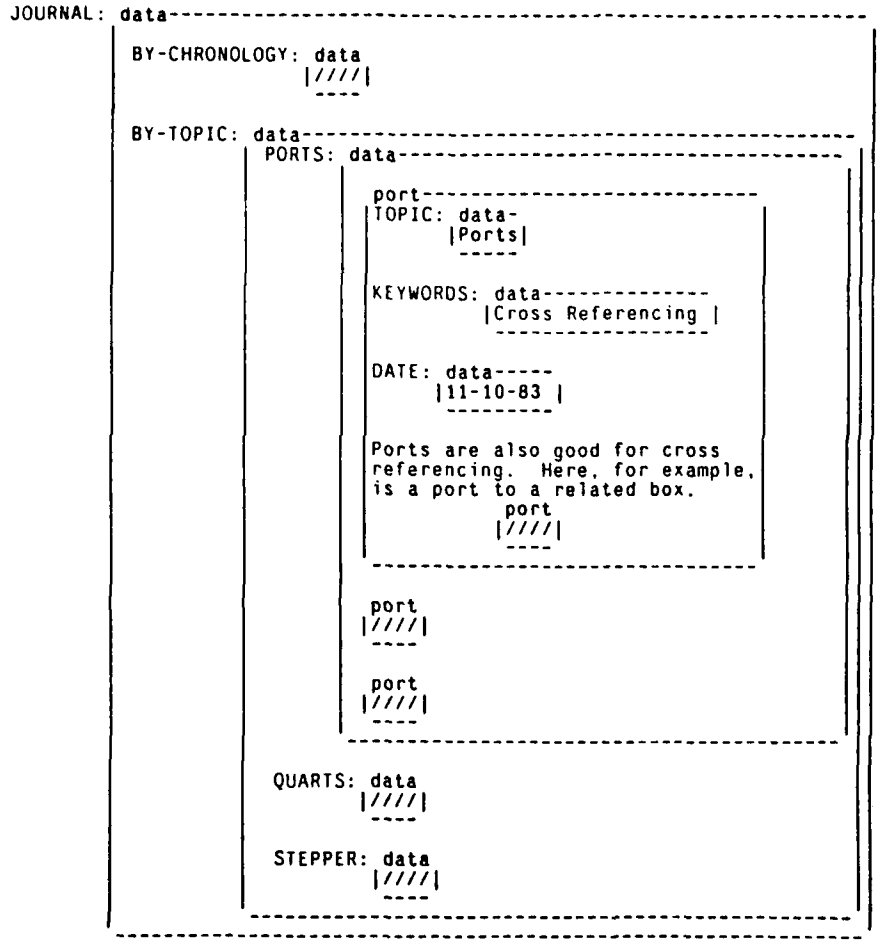


Figure 6-3: The first box in the PORTS section of BY-TOPIC is a port to the first entry shown in the previous figure. The other ports in that section are to other entries in BY-CHRONOLOGY.

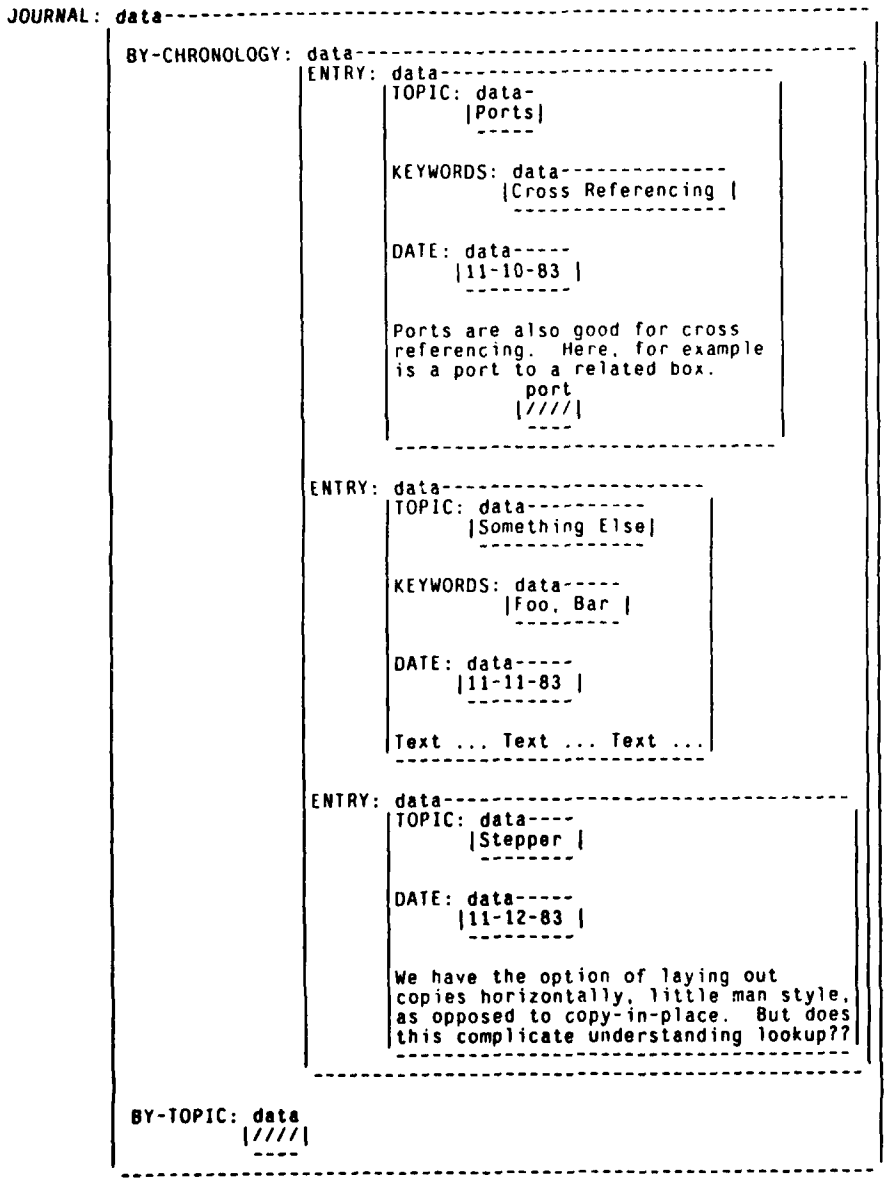


Figure 6-2: A Journal contains two views of its entries, BY-CHRONOLOGY and BY-TOPIC.

organization of the system. The primary difference between a port and a window is that the port itself is spatially located in the system hierarchy, not attached to the screen. A port appearing in a data structure indicates that the contents of the port is shared in basically the Lisp sense. A difference between Boxer sharing and Lisp sharing is that any object really belongs to (is contained in) a unique other object and can only be "used" in other places.

In the context of sharing, one can see a subtle but important shift in the meaning of variables from Lisp and Smalltalk to Boxer. The meaning of setting a variable in Boxer is to change the contents of a box, so that any port to that box sees the change. In Lisp, one cannot share in this way. A second object can indeed point to the value of a variable, but changing the setting of that variable creates a new pointer from the name to the new value, leaving the object which shared the old value still pointing to it. This all means an extra layer of indirection in implementing Boxer variables. But that layer corresponds to a key idea -- it represents place: If a variable is to have a place, that place must remain invariant in the process of setting the variable, and that fact, in turn, must be represented in the implementation.

The following example shows the usefulness of ports in cross referencing and in producing multiple views of a system. It also happens to be a fine example of Boxer used as a personal information system.

Keeping a personal database in Boxer is a trivial matter. Here, we use ports to make available an alternate organization of entries in a journal. The top box in Figure 6-2, BY-CHRONOLOGY, contains all entries in chronological order. The bottom box, shown expanded in Figure 6-3 contains the same entries reorganized (via ports) according to topic. The intent is to allow the journal keeper to access an entry according to preference or how he remembers it: "I seem to remember writing something about that a week or so ago;" versus "Let me see what I have on the topic of ports." Because ports are views on objects which appear in another place, any change made in a port is instantly reflected in the original entry. If both the port and the target of the port are on the screen, typing into either results in the characters appearing in both.

We imagine the protocol for using such a journal to be something like the following. One can make an entry by copying the form of a previous entry, or using a template as described for mail above. Actually putting the entry in place could be handled with a FILE function, which would make sure to insert a port to the new entry under the appropriate topic in the BY-TOPIC listing. Of course, one could also have an UPDATE function which, when executed, placed a port to any new, unported entries in the BY-TOPIC listing. These utilities, FILE and UPDATE, are actually not very complex Boxer procedures, though one would not expect early novices to write them. They could be augmented with procedures to, for example,

by-one and then at some later time indicate that the typed statements should be incorporated into a program. Consequently, programming in Boxer is often not so much "writing" programs as it is piecing them together concretely from objects already in the system.

2.2. Focus of Our Work This Year

Last year we built a minimal but usable implementation of Boxer on a CADR Lisp Machine. During this year we transported the system to a Symbolics Machine and filled out the basic functionality of the system, including improving and extending the editor. Among the basic issues settled during the year were the set of data operations and how they relate to sharing structure. We illustrate this work here by describing a new structure added to Boxer.

Ports: Boxer maintains a strong identification of "things" with "places," and "organization" with "spatial relationship" (in particular, containment implies inheritance). This provides a firm foundation for easy, incremental learnability of the system through inspection and through a uniform method of interpreting, modifying and expanding what one sees. Nonetheless, these identifications are very strong constraints on system organization and possible interpretations of "running a program." In particular, Boxes are strictly hierarchical, and each box exists in precisely one place. This means it is impossible to share in pure Boxer as one can in Lisp, having an object be part of more than one other object (via multiple pointers to the shared object). It also means one has only a single view of any object in the system -- that provided by the spatial context where the object exists. In contrast, one may sometimes want to see things on the screen that are related in some way other than with respect to their system organization. While running a program in some environment, one might wish to view the changing contents of some distant data box. Or one might want to be looking at some part of the system while constructing another part, say constructing a program in analogy with another from a different context. Window systems were invented partially to serve this kind of function.

Sharing and multiple views are not of first-rank importance to novices, but they are important enough that we would like to incorporate them for more advanced users. To meet this need we have implemented a single structure that provides much of this functionality, but which we consider minimally subversive of the box semantics. It is called a *port* ("view port") and has most of the properties of a box. It appears as a rectangular region that can be named and is constructed and erased in essentially the same way that a box is. But its meaning is a passageway to another part of the system. What one sees in a port is a part of the system located in another place. Thus one can inspect and even change remote objects. In general, one can pretend that another part of the system is in the place of viewing without changing the "real"

2.1. Design Principles of Boxer

One of our major concerns in formulating the design of Boxer is to provide a mechanism that will enable even beginning users to deal with the large-scale structure of the system. The approach we have adopted is to organize Boxer in terms of a pervasive spatial metaphor. Elements of the environment can be thought of as places, and their visible spatial relationships have structural meaning. All compound objects are versions of "boxes" which are two dimensional arrays of words or, recursively, boxes. The containment relation among boxes provides, via the spatial metaphor, a model of all hierarchical structural relations in the system, from variable scoping, through program/sub-program structure to index and file arrangement. In fact, the entire system can be regarded as a single two-dimensional geometric space through which the user moves. This is a crucial aspect of making the system accessible to beginning users, since it links the central organizing image of the system to geometric intuitions.

Figure 6-1 shows a simple example of spatial organization in a procedure called SQUARE, which draws a square on a graphics display screen by repeating four times a sequence of two moves called CORNER. Notice that the definition of the CORNER procedure is geometrically contained within the SQUARE procedure. This shows how boxes can be used to achieve the usual organization of block-structured languages. In Boxer, we extend this principle, using boxes as geometric carriers of many other hierarchical structures as well.

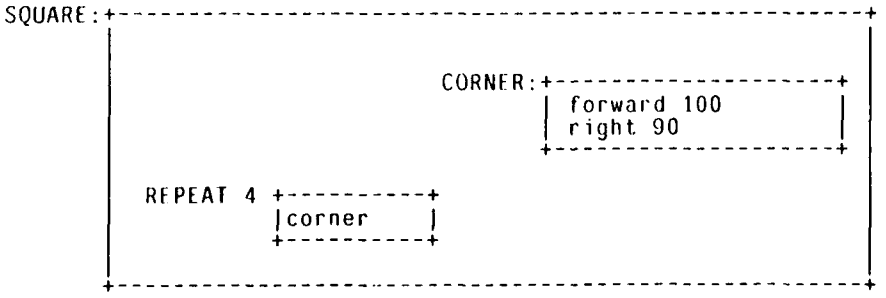


Figure 6-1: *Spatial Organization*

A second design principle in Boxer is the identification of objects with their screen representations. This is, in essence, a consistent commitment to the idea that the system *is* the way it appears on the screen. For example, any text that appears on the screen, whether typed by the system, typed by the user, previously executed or not, is available to be manipulated, edited or executed. This uniformity enables Boxer to support styles of interactive use that are very different from those found in other computational environments. For example, one can execute statements one-

1. INTRODUCTION

Serious work in education today requires an extraordinary breadth of competences and concerns which we can partition into: (1) technology, (2) the educational context (students, teachers and curriculum), and (3) cognition. In the Educational Computing Group we accept the need to work simultaneously in all of these areas -- understanding the basic principles of learning and knowing, fashioning computational tools from those principles, and developing learning materials and teaching techniques which are responsive at once to the best current ideas about learning, the most promising uses of technology and the realities of the classroom.

In the area of technology, our major work is the Boxer project which aims at providing the most general, easy-to-use computational facilities possible for non-expert computer users such as students, trainees, teachers and computer-materials developers. The educational context is represented by our work in a number of settings, including an experiment in very high density of computers in two elementary classrooms in the Boston Public Schools: work in the Carroll School with learning disabled youngsters, and work with MIT undergraduates under the auspices of Project Athena. Our work on cognition has been spread across projects. It has centered on understandability of complex computational systems in the Boxer Project, on spatial reasoning at the Carroll School, and on studying children's notions about physics in connection with the Cambridge Public Schools.

2. BOXER

During the past year, we have continued the design and implementation of Boxer, whose goal is to integrate a wide variety of applications -- text manipulation, programming, graphics, information retrieval and data manipulation -- within an easy-to-learn framework. Boxer is motivated by our conviction that most non-specialist users of computers are best served by providing a computational environment that is integrated and coherent, in which all the basic capabilities can be assimilated to a single, uniform computational scheme. In Boxer, we have been trying to achieve system coherence through (1) a simple spatial model representing all hierarchical organizations of data and programs, and (2) a universal insistence that all objects are equivalent to their screen representation. The latter provides a uniform way of manipulating all system objects, files, programs, databases, records, textual objects, etc. using the same "extended text" editor.

EDUCATIONAL COMPUTING

Academic Staff

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

Research Staff

E. Lay
D. Scrimshaw
D. Smith

Graduate Students

M. Eisenberg
D. Bisaillon
M. Hassamali
L. Morecroft
F. Turbak

Undergraduate Students

R. Harris
T. Kellison
L. Kolodney
D. Luneau
S. Martin
R. Ouellette
R. Steinmetz
C. Hibbert
L. Klotz
J. Lee
J. Marshall
A. Mocl
D. Spitz
E. Twietmeyer

Support Staff

P. Davis
J. Karaslaanian
M. Palmgren

Visitors

T. Globerson
W. McKay

3. Romkey, J.L. "Reliable Datagram Multicast on the Internet," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1984.

4. Sollins, K.R. "Distributed Name Management," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1984.

Talks

1. Clark, D.D. "Remote Virtual Disk Protocol," Internet Research Group, January 1984.

2. Clark, D.D. "A Case Study: The Campus Network Plan for the Massachusetts Institute of Technology," ACIS, IBM, Rockville, MD, January 1984, March 1984.

3. Clark, D.D. "Overview of Research at Massachusetts Institute of Technology, Laboratory for Computer Science," Digital Equipment Corporation, Hudson, MA, March 1984.

4. Clark, D.D. "The Reality of the Network Protocol Jungle," MIT Industrial Liaison Program, Cambridge, MA, April 1984.

5. Greenwald, M.B. "Swift: An Operating System for a Personal Computer," Ninth ACM Symposium on Operating System Principles, Bretton Woods, NH, October 1983.

6. Greenwald, M.B. "Accessing Secondary Storage Across a Data Network," Digital Equipment Corporation, Littleton, MA, June 1984.

7. Martin, E.A. "MIT Gateway Projects: Campus Network/Project Athena and Dynamic Routing," Gateway Special Interest Group Meeting, USC Information Sciences Institute, Marina Del Rey, CA, February 1984.

8. Sollins, K.R. "Distributed Name Management, Ninth ACM Symposium on Operating System Principles, Bretton Woods, NH, October 1983.

Conference Participation

1. Clark, D.D. Panel Session, ACM Sigcomm '84, Communications Architectures and Protocols, Montreal, Quebec, Canada, June 1984.

Publications

1. Saltzer, J.H., Reed, D.P. and Clark, D.D. "End-To-End Arguments in System Design," to appear in *ACM Transactions on Computer Systems*, (November 1984).

2. Greenwald, M.B. "Remote Virtual Disk Protocol Specification," MIT Laboratory for Computer Science Technical Memorandum, Cambridge, MA, to appear 1984.

Theses Completed

1. Gobioff, B. "An Investigation of Development Methodologies for Communications Software," M.S. thesis, MIT Sloan School of Management, Cambridge, MA, May 1984.

2. Kim, T.H. "A Distributed Mail System Repository," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

3. Krajewski, R.P. "Required Capabilities for a File Access Protocol," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

4. Shinsato, H.J. "A CLU Interface for a Bit-Mapped Display," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

5. Skinner, G.D. "An Implementation of an ARPANET FTP Server for UNIX," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

6. Spurlock, J. "A Comparative Study of Distributed File Systems," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

Theses in Progress

1. Siegel, E.H. "Dynamic Linking in a Type Safe Environment," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1984.

2. Gramlich, W.C. "Checkpoint Debugging," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1984.

retained. Checkpoint debugging can also be effectively used to help debug real-time systems. Checkpoint debugging is also useful for debugging high availability programs (such as printer servers) where the system maintainer can not be available for debugging 24-hours a day.

exist yet; it need not exist until execution is required. A second aspect of availability relates to locking or checking out program modules for modification or improvement. These and other aspects of availability are still under study.

A second issue arising from the programming support environment is that the set of participants is distinct from the set of objects named, unlike the mail example. In fact, humans often use the set of participants as part of their means of identifying a particular aggregate. The third and fourth issues are not directly naming issues, although they are closely related. The third is the issue of how and when participants are notified of modifications to the shared context in cases where someone else made the modifications. The fourth is the implementation issue of how the updates are done and to what degree distributed copies of the shared context are synchronized. In the mail system, both of these were achieved through the mail system itself. A message carried both name and address of the sender and all recipients. If a name was not known in the local copy of the shared context, it was added and until it had been used several times the recipient saw both the name and the address. At such time that the name was accepted, the address was stripped off before displaying the message to the recipient. In the programming support environment such facilities for conveying and displaying information are not available. As the work progresses these issues will be addressed in further detail.

The current state of the dissertation is that the implementation is complete, work is progressing on a detailed design for a programming support naming facility, and writing of the dissertation is under way.

6. CHECKPOINT DEBUGGING

Wayne Gramlich continued work on his thesis in the area of debugging distributed systems. The specific debugging technique that is being investigated is called checkpoint debugging.

The basic idea is to regularly take an atomic snap-shot of the process state and then record all subsequent process input until the next atomic snap-shot. When a bug is encountered, the previous snap-shot and all of its process input is retained for subsequent analysis. Debugging is performed by reloading the snap-shot and replaying the process input.

Since computers are deterministic finite state machines, the sequence of events leading up to the occurrence of the bug can be recreated as many times as necessary. A conventional interactive debugger can be used to help find the bug when the checkpoint is being replayed. For a distributed computation, all the processes in the computation must be regularly checkpointed. When a bug is encountered in any of the processes, the checkpoints for all processes must be

outside the MIT network, and second, which neighbor gateway to use to get to the various MIT subnets.

This IGP should serve the MITnet's needs for the next five years or so, at which point the amount of routing information that must be processed may become too cumbersome.

4.3. Internet Congestion Control

Lixia Zhang began a study of network resource allocation techniques suitable for the DARPA Internet. The Internet currently has a simple technique for resource allocation, called "Source Quench."

Simple simulations have shown that this technique is not effective, and this work has produced an alternative which seems considerably more workable. Simulation of this new technique is now being performed.

5. DISTRIBUTED NAME MANAGEMENT

Karen Sollins continued work on her dissertation as described in the previous progress report. The previous report addressed a collection of issues in naming in a distributed environment. The work this year included further work on a paradigm for naming (as described in that earlier progress report). To this end, Sollins has implemented the ideas in an electronic mail system. The exercise of implementing has had both positive and negative effects. First, due to limitations of the programming environment and the resources available, the model was simplified. Contexts and aggregates are available to the users, but not in their full generality. For example, names can only be translated to and from other strings, which are assumed to be network addresses. On the other hand, the implementation did highlight a problem that had not been previously considered in detail: the issue of proposing new names and the stages through which a name progresses until it is accepted by the community of participants in an aggregate. Further work on this issue continues.

Electronic mail provided a restricted set of naming problems. In order to investigate naming further, work is now progressing on examining the naming issues in a programming support environment. A number of issues arise here that did not arise in the mail example. First, there is the issue of availability. There are several aspects of availability. One is whether or not an object that is being named is currently accessible or not and whether this is important for the activity at hand. For example, if the activity is compilation and the interface to the named procedure is known, perhaps the executable version of the procedure residing on another computer need not be accessible. Perhaps the executable version does not even

them back because of their weaknesses. Preliminary results indicate positive results. It is expected that these materials will be of benefit even to students who are not classified as learning disabled, but who have preferred learning styles and special reasoning abilities that are not tapped by standard curriculum.

3.3. Project Athena

Two Athena Projects have been begun by members of the Educational Computing Group. Hal Abelson has initiated the development of a course on linear algebra. Andy diSessa has started developing a course in computer-aided research and design. The latter uses material in the standard freshman physics curriculum, but is intended to bring students as quickly as possible to the position where they can engage in original research and design problems which take a significant fraction of a term to complete. The course is being developed in modules consisting of text computer simulations and microworlds that not only serve as a basis for student homework problems, but as a starting point for student research projects as well.

4. COGNITIVE STUDIES

In addition to Weir's work on spatial reasoning described briefly above, another piece of research is in progress by diSessa and a postdoctoral assistant, Tamar Globerson, visiting from Israel. This project is aimed at determining how systematic and theory-like the common-sense physical knowledge of children is. Students from pre-school, third and sixth grades have been presented with a number of situations, including computer simulations, that all present basically the same situation: An object is moving in a circular path (e.g., a ball on a string) and suddenly the circular constraint is broken (e.g., the string is cut). What trajectory does the ball follow?

In contrast to other studies, this work is showing a very complex context dependence in the answers the children give, indicating the non-theory-like nature of their knowledge. Children will change their minds on the basis of subtle changes in the problem, can be convinced to accept possibilities they do not suggest themselves, and even change their minds on viewing a simulation of the predictions they themselves make, as if thinking in the abstract about motion, and judging plausibility by watching involve two different knowledge pools.

Follow-up work will chart in detail the development of particular intuitive "laws" of physics, examine the systematicity of the apparently patchwork collection of ideas children have, and eventually develop computer activities to engage and develop children's physical intuitions.

Publications

1. Abelson, H. *TI Logo*, Byte Books, Peterborough, NH, 1984.

2. Abelson, H. *Structure and Interpretation of Computer Programs*, (with G.J. Sussman and J. Sussman), MIT Press and McGraw-Hill, Cambridge, MA, 1984.

3. diSessa, A. "The Computer as an Epistemological Catalyst," in *Children and Computers*, L. Klein (ed.), Jossey-Bass, Inc. (in press).

4. diSessa, A. "A Principled Design for an Integrated Computational Environment," *Human-Computer Interaction* (in press).

Theses Completed

1. Harris, R. D. "FloWorld: A Graphical Modeling System for Momentum Flow," S.B. thesis, MIT Department of Electrical Engineering, Cambridge, MA, June, 1984.

2. Hibbert, C. T. "Integrated Computing Environments: The Control of Complexity in Powerful Systems," S.B. thesis, MIT Department of Electrical Engineering, Cambridge, MA, June 1984.

3. Kellison, T. "Two Kinds of Measurement: The Way Kids Think about Angles and Lines," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1984.

4. Strassmann, S. H. "Learning Lisp: The Barriers to Novice Programmers at MIT," S.B. thesis, MIT Department of Electrical Engineering, Cambridge, MA, June 1984.

Talks

1. Abelson, H. "A Lisp-Programmer's View of Software Engineering," *General Telephone and Electronics Conference on High-Level Languages*, Norwalk, CN, September 1983.

2. Abelson, H. "Turtle Geometry," *Annual Conference of Computer Education Group of Victoria*, Melbourne, Australia, May 1984.

3. Abelson, H. "Advanced Programming," *National Logo Conference*, MIT, Cambridge, MA, June 1984.

4. diSessa, A. "Introducing Computers in Schools - Logo across K-12," all principles and faculty development staff of R-1 school district, Jefferson County, CO, August 2, 1983.

5. diSessa, A. "Intuitive Physics," Colloquium in Developmental Psychology, Columbia University, New York, NY, September 25, 1983.

6. diSessa, A. "Learning Physics from Dynamic Turtles," and "Boxer: Goals and Means for Producing an Integrated Computational Environment," *IEEE Educational Computer Conference*, San Jose, CA, October 1983.

7. diSessa, A. "Computers and Learning: New Means to New Ends," keynote address at the President's Forum on Educational Issues: Computers and Education, Rutgers University, New Brunswick, NJ, November 4, 1983.

8. diSessa, A. "Perspectives on the Future of Educational Software," WNET Professional Forum on Educational Applications of Interactive Technologies, New York, NY, January 12, 1984.

9. diSessa, A. "Future Computer Languages for Education," Swedish National Board of Colleges and Universities, Symposium on Computers in Research and Education, Cambridge, MA, January 25, 1984.

10. diSessa, A. "Learning Physics with Logo," Boston Computer Society, Cambridge, MA, March 1984.

11. diSessa, A. "Boxer: Designing an Integrated Computational Environment," Annual Meeting of the American Educational Research Association, New Orleans, LA, April 27, 1984.

12. diSessa, A. "The Future of Programming," *National Logo Conference*, MIT, Cambridge, MA, June 30, 1984.

13. Papert, S. Approximately 30 talks from July 1983 to July 1984.

14. Weir, S. "The Logo Learning Environment: A Catalyst for the Education of Severely Handicapped Children," *Annual Statewide Conference on Microcomputers for Special Education*, Worcester Marriott Hotel, Worcester, MA, September 19, 1983.

15. Weir, S. "Incorporating Microcomputers for Special Needs Children into

an Urban School System." The Lowell Model for Educational Excellence Study Commission, Lowell, MA, October 12, 1983.

16. Weir, S. "Computers and Special Needs Children," St Paul Day Teacher Workshop. Evening: Lecture to St. Paul Medical Group; St. Paul, MN, April 1984.

17. Weir, S. "What do Children Learn," *National Logo Conference*, MIT, Cambridge, MA, June 28, 1984.

18. Weir, S. and Ary, T. "Bridge-building between Logo and Classroom Math" poster session *Logo 84*, MIT, Cambridge, MA, June 29, 1984.

19. Weir, S. segment in "The Talking Turtle," P.B.S. NOVA, October 25, 1983.

FUNCTIONAL LANGUAGES AND ARCHITECTURES

Academic Staff

Arvind, Group Leader

Research Staff

R. Iannucci
J. Pinkerton

Graduate Students

M. Beckerle
S. Brobst
C. Chiang
D. Culler
P. Fuqua
S. Heller
R. Iannucci
V. Kathail
P. Lim
G. Papadopoulos
K. Pingali
M. St. Pierre
R. Soley
B. Vafa

Undergraduate Students

R. Adamjee
S. Viqar Ali
M. Bucci
J. Buonora
J. Cernada
T. Chambers
E. Desai
S. Douglass
S. Gahni
W. Hamdy
T. Im
C. Li
D. Morais
G. Ng
J. Ngai
C. Ozveren
J. Picciotto
K. Rahmat
R. Sathyanandan
K. Traub
W. Tsang
C.-S. Wei
A. Wang
M. Wong
C. Wu
J. Ying
S. Younis

Support Staff

P. Sedell

Visitors

E. Hagersten

I. Jacobson

1. INTRODUCTION

The primary direction of the Functional Languages and Architectures Group continues to be the study of new computer structures to exploit parallelism in application programs. Our approach in studying parallelism is based on functional languages and dynamic dataflow machines. We believe the success of a general-purpose multiprocessor computer depends on its effective programmability and efficient utilization of resources. Thus, in addition to the hardware architecture, we are concerned with high level language support, communications requirements, and efficient distribution of workload over the machine. We feel the development of novel parallel architectures will require several iterations. Hence, the group is pursuing a variety of interrelated projects, all aimed at developing our understanding of the problems and moving us closer to a final implementation. We have organized this report in two major sections: The first describes the Tagged Token Dataflow Project and the other describes the Multiprocessor Emulation Facility (MEF) for experimenting with parallel machines. The Tagged Token Dataflow architecture is going to be the first large scale emulation experiment on the MEF.

The Tagged Token Dataflow Project is a major effort to realize the model of dataflow computation embodied in the U-interpreter [3] The architecture continues to evolve as our understanding of the issues related to realizing this model deepens. The development of a detailed simulation of the architecture on the IBM 4341 in cooperation with IBM Yorktown Heights over the past year has provided invaluable experience. In order to build the simulation it was necessary to give detailed specifications of every major component of the architecture. This process uncovered a number of oversights in the early design. Moreover, in order to run programs on the simulated machine, it was necessary to develop a run-time resource management system to control the allocation of resources and distribute work over the collection of processors. This has considerably sharpened our attention on resource management issues in the past one year.

The simulator is too slow to be used as a vehicle for experimenting with large dataflow applications. Thus, a large scale multiprocessor emulation of the Tagged Token Dataflow Machine is being developed to meet this need. We expect the emulation to bring about a depth of understanding of dataflow applications far beyond the current state of the art. Currently, the emulated dataflow machine runs on five high-performance Lisp Machines which are connected by an Ethernet. programming for both the simulated and the emulated machines is done in the high-level dataflow language Id. We have an operational Id-to-graph compiler which is used to drive both prototype dataflow machines. The Id definition has been revised to permit a more elegant use of higher-order functions. We plan to implement a new version of Id in the next two years.

The goal of the Multiprocessor Emulation Facility is to develop it as a useful tool for research on new parallel architectures and associated languages. The facility will consist of 64 Lisp Machines and a high bandwidth interconnection network. Generic software for inter-processor communication and for emulating other architectures is also to be provided. In the past year, significant progress toward these goals was made. Two parallel efforts to design the communication network have been put in place. One design uses circuit switching and communication on four bit parallel data links while the other uses packet switching and bit-serial communication. The former is conservative in the use of technology and thus represents lower risk than the latter. A detailed design for the circuit switch card, excluding the Lisp Machine interface, has been completed.

We have been able to attract significant industrial support in the form of three circuit designers from IBM Endicott and one from Ericsson, to do the hardware development for the emulation facility. However, the infrastructure in the Laboratory for Computer Science for hardware development is not adequate to support the MEF. Thus, a detailed plan for a hardware laboratory was prepared and partially executed. Further construction in the hardware laboratory is contingent upon receiving more research funds which are believed to be available during the coming year.

2. TAGGED TOKEN DATAFLOW PROJECT

2.1. Architectural Background

The Tagged Token Dataflow Architecture is composed of a number of Processing Elements (PEs), connected by a packet switched communications network. Each PE is a complete dataflow computer. The basic organization is shown in Figure 7-1. The PE consists of a number of asynchronous pipeline stages, connected by FIFO buffers. The various stages form three subsystems. The subsystem shown toward the right in Figure 7-1, performs the basic instruction processing. The stages of this pipeline reflect the essential steps in the processing of a dataflow instruction: detect when data has arrived to enable an instruction, fetch the instruction, compute the result, generate result tags, and finally dispense the result tokens. The subsystem to the left provides storage for data structures. This structure store incorporates a number of innovative ideas to allow for sharing of information without constraining parallelism. The benefits of this approach are presented in [4] A detailed design of the controller for the structure store is presented in [9]. The center subsystem includes a PE controller, which provides a variety of support operations, including input/output, block transfers, and access to the resource management system.

Tokens and Tags: In the Tagged Token Dataflow Architecture, values are carried

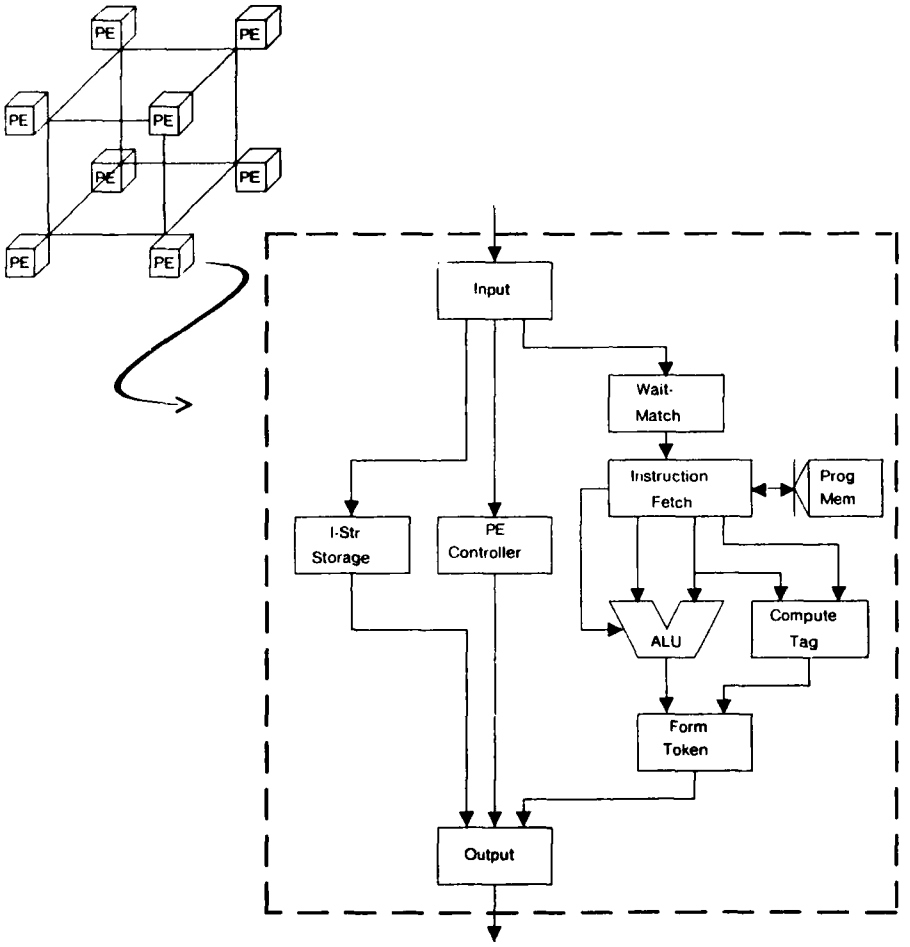


Figure 7-1: A Block Diagram of the Abstract Machine

on *tokens*, which are passed from one instruction to the next. The arrival of data causes the corresponding instruction to be fetched, unlike a conventional computer in which the execution of an instruction causes data to be fetched. There is no program counter in this machine. Each token carries a *tag*, in addition to a data value, which specifies the instruction to be executed. The tag contains essentially three items of information: the address of the PE which is responsible for executing the instruction, the address of the instruction to execute within that PE, and the *context* in which the instruction is to be executed. The PE address is required because a code-block may be spread over many PEs, and tokens must be freely transferred between PEs. The contextual information is required because many logically distinct activations of a given code-block may be in execution simultaneously. There must be a way to distinguish the various activations so that tokens belonging to different activations do not interact. All tokens belonging to a given activation carry the same context identifier or *color*. Thus, two tokens are destined for the same instance of an instruction if and only if their tags match.

Instruction Processing: Upon arriving at a processing element, a token enters the waiting-matching section. The tag it carries is compared against the tags of all the tokens resident in the waiting-matching store. Instructions are limited to two operands, so if a match is found, the corresponding instruction is enabled for execution. The two matching tokens are purged from the waiting-matching store and forwarded to the *instruction fetch* section. The instruction specified in the tag is fetched from program memory, along with any required constants. The data values are aligned, and an operation packet is sent to the *ALU* for processing. In parallel with the ALU, the *compute-tag* section forms tags for result tokens, based on the destination list of the instruction and contextual information on the input tag. The result values and tags are merged to form tokens and passed on to the communication network, whereupon each is delivered to the PE specified in its tag.

Tolerance to Communication Latency: In many respects, a multiprocessor setting presents a fundamental architectural challenge. Communication latency between processors is generally large and unpredictable. Thus, for a multiprocessor architecture to be successful, the individual processing elements must be extremely tolerant to communication latency [4]. The PEs which comprise the Tagged Token Dataflow Architecture meet this challenge. Note that once an instruction is enabled, it may be processed to completion without further communication with other PEs. The pipeline is never held up by communication latency. Waiting only takes place in the waiting-matching section. Completion detection for external communication is provided naturally by the basic instruction scheduling mechanism; when data arrives an instruction is scheduled. This dynamic scheduling, coupled with the ability to interlace many independent threads of computation, allows for overlapped requests to the communication system, tolerates long latencies, and tolerates unordered responses.

2.2. Resource Management

The description of the architecture presented above assumes that the program graph is in place, distributed appropriately over the machine. This section looks at the process of executing programs on the Tagged Token Dataflow Architecture at a somewhat higher level. It focuses on how resources are managed and how work is distributed over the machine.

Static vs. Dynamic Resource Allocation: There are basically two approaches to resource allocation, each offering advantages and disadvantages. On one extreme is the static approach where the compiler assigns processors and storage to a program. This is a common strategy for scientific programs written in Fortran. If the decomposition is just right, the performance can be extremely good, with little overhead for the distribution of work at run-time. However, developing a good static decomposition appears to be a very difficult problem. In any case, functional languages require dynamic resource allocation and thus preclude a purely static approach. A purely dynamic approach, on the other hand, implies allocating each computational activity on the least busy processor. Dynamic allocation on the Tagged Token machine is undertaken only at the level of procedure calls, (i.e., code-block invocation), each of which is assigned to a group of PEs at run-time. David Culler has developed a dynamic resource management system for the simulation along these lines. This resource management system has also been adopted for the emulated dataflow machine. We are currently experimenting with various allocation algorithms and load balancing techniques. The present architecture provides an efficient and flexible way to map loops and recursive procedures on a group of processors. As static analysis techniques develop, we will consider more aggressive optimizations for mapping special program structures.

Hierarchy of Resources: A variety of resources are required to support a code-block activation. These include PEs, program memory, code-block registers, colors, etc. Coordinating resource allocations between a variety of PEs can be extremely cumbersome, if PEs are allowed to cooperate in arbitrary ways. In order to keep the complexity of resource management tractable, a hierarchy is imposed on the set of system resources. In effect, the resource manager deals with blocks of resources and allows the hardware to distribute them at a finer grain automatically. This also allows the number of requests to the manager to be reduced.

The notions of *physical domain* and *physical subdomain* are introduced to facilitate selecting a set of PEs for an invocation. The collection of PEs in the system is divided into disjoint *physical domains* (PD), each being a set of consecutively addressed PEs. This partition is not allowed to change while programs are running. Each PD may be further partitioned into a set of disjoint *physical subdomains* (PSD), again each is a set of contiguously address PEs. The size of the PSD is dynamic and will vary for different code blocks in a given PD. A code block activation is assigned to a domain, and a complete copy of the code is placed in each subdomain.

To simplify memory management, we require that memory allocation be identical in every PE in a given domain. The code-block is split into as many sub-blocks as there are PEs in a subdomain. Each PE receives one sub-block, all starting at the same base address. This allocation is replicated for each subdomain in the domain. For computational efficiency we require that all PD and PSD be a power of two in size.

Each copy of the code-block may be used by many concurrent activations and each of these may require mapping information to be stored locally to the PEs, so that result tags can be generated. Again, by grouping activations together this mapping information can be kept within reason. Each PE contains a set of 256 *code-block registers* (CBR). Each code-block register contains a code-base address and information pertaining to the mapping of the code-block onto the domain. Each code-block register can support 16 *colors*; additional information can be associated with each color. Code-block registers, like program memory, are allocated uniformly throughout each domain. With this partitioning, the manager views the system resources in terms of domains, code-block registers, and colors.

To understand the role of the manager, consider the process of code-block activation. A request is sent to the resource manager. It first selects a domain with sufficient resources to support the activation. If the code-block is not present in this domain, it will have to be loaded. In this case, the subdomain size may be set as suggested by the compiler, or as determined by the resource manager. A code-block register must be allocated, and a set of colors, relative to this CBR, is assigned to this activation. Finally, program memory is allocated throughout the domain and the code is loaded. Mapping information is established in each PE to describe the disposition of the code-block, as part of the CBR. Further information pertaining to each color can also be provided. The code-block is now ready to execute. The manager allocates argument and result structures and sends descriptors to both the caller and the newly activated code-block. The code-block may now execute on its own except for invocation and resource allocation requests. Subsequent invocations of the same code-block may share the CBR, if colors are available. If not, a new CBR must be allocated, but it may share the copy of the code. In either case the mapping parameters will have to be the same as the original, since the code has the same disposition. New color information may be provided.

These resource management concerns essentially dictate the structure of the tag carried on tokens. It consists of five fixed size fields <PE #, CBR, Color, Initiation #, Relative Instruction Address>. The code-block register gives the base address of the code and the base address of the color information. This allows instructions and constants to be fetched from program memory. The CBR also describes the disposition of the code-block across the domain, so the hardware can generate tags for result tokens.

Efficient Internal Invocation: The resource manager provides a mechanism for initiating activity in a possibly distant region of the machine. However, in many cases this generality is not required and is overly expensive. In order to allow highly repetitive program structures to execute without involving the resource manager, 256 initiation numbers are associated with each color. Loops and recursive procedures make use of the initiation numbers to distinguish tokens belonging to different iterations, or different recursive invocations. A code-block activation is allotted a color and has the range of initiation numbers at its disposal. Loops utilize the D operator, which increments the initiation number. Recursive procedures make use of the R operator, which computes a new initiation number of the form A * I + B. Many loops may exceed 256 iterations, hence facilities are provided for allocating blocks of colors and for using the next color in the case of initiation number overflow. The compiler generates code to test whether all colors and initiation numbers have been exhausted and to issue a manager request for more colors if necessary.

The hardware provides a mechanism to distribute internal activations over processors efficiently. This is the rationale for partitioning domains into subdomains. Recall, each subdomain contains a complete copy of the code. The hardware distributes activations within a domain based on the initiation number. An additional parameter is provided to support block distribution (i.e., k iterations on this PSD, k on the next, and so on). This follows along the lines discussed in Arvind, et al. [2].

The results of static analysis are expected to guide the resource manager in making allocations. In particular, it enters into the process of choosing among domains, choosing subdomain size, choosing k (the number of iterations to keep together), and for deciding where to allocate data structures. Conveying this information is somewhat subtle, however. Completely static information, such as nesting relationships, can be included in the compiler output along with the code. Execution dependent information is supplied by augmenting the graph slightly and sending information along with the request. Currently, only simple information is conveyed in this way, such as the expected number of iterations. But we are investigating more powerful aids of this form.

Compiler Responsibilities: Introducing a resource management system into a dataflow model raised a number of interesting compilation issues. The U-interpreter is entirely independent of resources; it assumes unbounded computational resources. In order to execute dataflow programs on a practical machine, resources must be explicitly requested. They must also be released in some manner. The basic dataflow graphs must be augmented so that the resource manager request is generated when resources are required. Also, the compiler must generate code to make proper use of blocks of resources. For example, to use initiation numbers

properly, it is necessary to handle overflows. The compiler must generate code to determine when resources can be released. This involves detecting completion of code-block activations and proper handling of reference counts for structures.

Detecting completion is non-trivial in a dataflow system because a code-block may continue to execute after all interesting results have been produced. Many iterations of a loop may be active simultaneously, so the compiler must detect that all have completed. For acyclic graphs there is little trouble. A signal token is generated for any operators which have no destinations. These and all the output arcs of the graph form the inputs to a binary reduction tree. The token that falls out the bottom of the tree is the last token of the activation. This technique can be extended to loops, but care must be taken to avoid serializing loops that would otherwise provide parallelism. The special signal token of one iteration is part of the completion tree for the next. In effect completion detection is serial, even though the loop may be parallel. The detection simply lags behind the loop execution.

One important implication of this incremental completion detection is that it makes it possible to run loops of an arbitrary number of iterations using only a few colors. The loop is given multiple colors. As it exhausts one color it goes on to the next or requests more. The completion follows behind detecting and releasing colors that are no longer needed and, hence may be recycled.

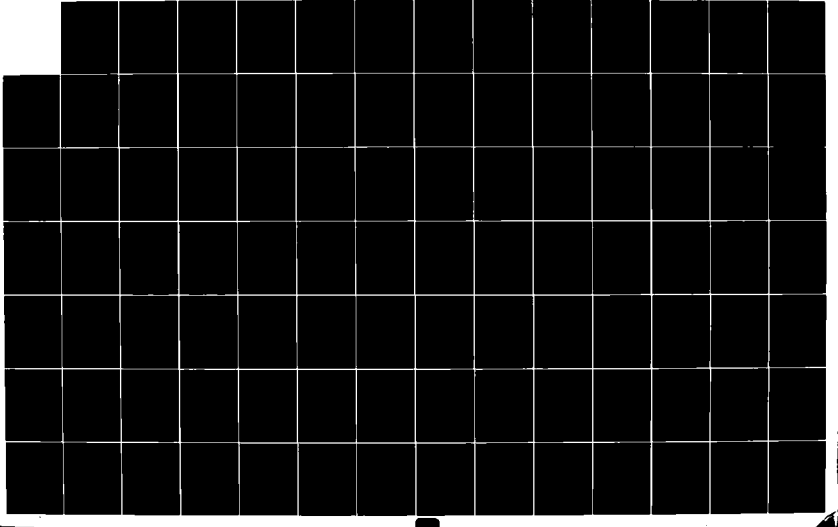
The compiler must also assist in the detection of parallelism in the construction of data structures. When certain restrictions apply, arrays can be modeled as I-structures which allows for a more parallel implementation than for ordinary structures. Detecting those arrays which can be implemented as I-structures is difficult in general; it involves analysis similar to that done by vectorizing compilers during code vectorization. There are, however, several important special cases where I-structures can be detected easily. We have also found several situations in which the techniques of vectorizing compilers can be borrowed and applied. The compiler generates instructions for the run-time system to allocate I-structure storage. For storage reclamation, reference counting can be implemented because I-structures are acyclic. The overhead of maintaining reference counts can be substantially reduced if, with compiler assistance, they are incremented and decremented at procedure call and exit only.

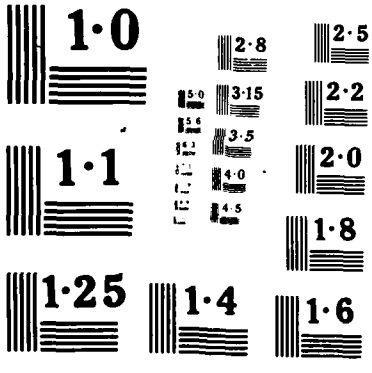
2.3. Simulation Experiments

The primary purpose of the simulation facility is to provide a prototype and test bed for the Tagged Token Dataflow Architecture. The basic requirement in designing the simulator was that all aspects of the architecture be modeled in a manner that could be cast directly into hardware. Deficiencies in the architecture were to be exposed and remedied, rather than disguised by software tricks. Working through the

AD-A158 299 LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 21 JULY 2/4
83-JUNE 84(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB
FOR COMPUTER SCIENCE M DERTOUZOS JUN 84

UNCLASSIFIED F/G 9/2 NL




1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
5·0
5·6
2·5
2·2
2·0
1·8
1·4
1·6

implementation of the machine in this manner did indeed uncover shortcomings and caused the specification of the architecture to become much more precise. The architecture is basically an asynchronous pipeline, with stages connected via finite sized buffers. This modularity is reflected in the design of the simulator; the functional behavior of each stage is modeled by a Pascal procedure which receives an input packet and the state of a station and produces a list of result packets with a new state. Thus, the specifications of the major hardware components are easily discernible from the simulator software. Also, the modeling of buffer interactions is divorced from the modeling of the functional behavior of the components. This design has greatly simplified the check-out task and allows the architectural specifications to be easily modified.

The first requirement of this 'soft' prototype is that it support all operations defined in the instruction set [1] and provide enough resource management support to run substantial Id programs. This requirement has been met. Most of the code has been written by Culler, Brobst, Vafa and Wei. In the coming months, the simulator will provide a vehicle for analyzing the architecture. The initial set of experiments are directed at identifying potential architectural failures: buffer deadlock, insufficient waiting-matching facilities, and inordinate overhead. A larger body of experiments will focus on determining the proper balance of various components (e.g., buffer sizes and processing speeds) and how various factors affect the performance of the machine.

The simulation facility has provided valuable experience in programming, debugging, and diagnosis in a multiprocessor setting. The simulator pursues many computations in parallel. Thus, debugging a program being run on the simulator involves all the subtleties of debugging on a true multiprocessor: what does a breakpoint mean? what would be a meaningful trace? etc. We have developed a debugging tool which will allow the user to examine the invocation tree (i.e., the parallel counterpart to a conventional procedure trace) and to observe the execution of a specific code-block at the instruction level. Debugging the simulator itself has proved much like diagnosing hardware: bugs have to be isolated by running diagnostic programs on the simulator and inferring from their misbehavior. We are developing a significant diagnostic package which will carry over to hardware implementations of the Tagged Token Dataflow Architecture.

2.4. Emulation Experiments

As a further proving ground for the Tagged Token Dataflow Architecture on large applications, we are developing a high speed emulation of the architecture on the MEF. The first version of the software for this emulator has been completed, allowing us to run graphs generated by the Id compiler. The emulated dataflow machine can be configured to run several dataflow PEs on each Lisp Machine which

communicates with other Lisp Machines over a ten megabit Ethernet. Higher-speed networks are currently under development and the software is structured so as to allow machines to easily integrate with such networks when they become available.

Currently, enhancements to support debugging in an asynchronous environment and a user interface consistent with the simulator are being developed. The performance of the Lisp version of the emulator is roughly 200 dataflow instructions per second on a single Lisp Machine. If scaled up to 64 processors, this would be approximately 13,000 dataflow instructions per second. However, we have not yet begun optimizing performance by finding bottlenecks in the Lisp code or moving critical pieces of the Lisp code into micro-code, so we are confident that we can obtain our ultimate performance goal of 64,000 to 640,000 dataflow instructions per second after some fine tuning. These issues are discussed further in Section 3.

2.5. Compiler Development

Two compilers for the dataflow language Id have been developed. One of them translates Id into Maclisp, and provides a complete run-time system to execute the object code; the other one translates Id into dataflow graphs where each node in the graph is a Tagged Token Dataflow Machine instruction. The Id-to-graph compiler is also responsible for generating instructions that ask for allocations and deallocation of resources. Both compilers are written in Maclisp, and run on DEC-20 as well as on Lisp Machines. The first compiler was written by Kathail and Pingali, while the second one was written by Kathail alone. Work, in cooperation with IBM Yorktown Heights, is underway to run these compilers on IBM machines using YKTLISP.

Further work on compilers can be classified into the following three categories:

1) Develop and implement algorithms for the Id-to-graph compiler for doing static analysis of the programs to acquire information that may be useful to the system manager. This includes finding producer-consumer relationship among code-blocks and finding sizes of various data structures to name a few.

2) Develop and implement optimization techniques to improve the efficiency of the code generated by the Id-to-graph compiler. This includes optimizations like constant subexpression elimination, code-block merging to eliminate the setup time associated with a procedure call, as well as reduce the number of calls to the manager and constant propagation.

3) Incorporate streams, managers, and a declarative or deductive typing system in both compilers.

2.6. Higher-order Functions and Reduction

It has often been remarked that much of the power and elegance of functional languages stems from higher-order functions i.e., functions that can take functions as arguments or return functions as the result of application. The use of higher-order functions has been stressed by researchers interested in *reduction* and reduction languages; in particular, by Turner [17], Backus [6] and Burge [8] As part of our design effort, Arvind, Kathail and Pingali [5] re-examined the primitives provided in Id for programming with higher-order functions and found them deficient in several respects. In particular, we were dissatisfied with the *compose* operator of Id since we found that it was clumsy to use and led to contorted programs. Our research into languages based on the reduction (in particular, SASL [16] model of implementation has led us to revise Id in order to permit efficient implementation and ease of use of higher-order functions. In addition, this research has lead one of our group members, Kenneth Traub, to design a novel architecture for performing parallel reduction [15] The proposed architecture has several advantages over current proposals in the literature for performing parallel reduction.

Given an expression in a functional language, reduction interpreters attempt to simplify the expression by applying a "rewrite rule" to generate another expression which can be simplified in turn. By doing this repeatedly, the interpreter generates a sequence of expressions; if, at some point, an expression is generated which cannot be simplified further, the interpreter returns that expression as the answer. Expressions that cannot be simplified further are called *normal forms* and the number of reduction steps taken by an interpreter to produce the normal form of an expression is usually taken be be a measure of the work performed by the interpreter in reducing the expression. Examples of reduction-based interpreters are the normal-order and applicative-order interpreters for the λ-calculus.

Since most functional languages can be considered syntactic sugar for the λ-calculus, our research concentrated on interpreters for the λ-calculus. Our goal was to determine the factors which affected the number of times identical sub-expressions were evaluated. We believed intuitively that if an interpreter shared larger sub-expressions, it would take fewer steps to find the normal form. To our surprise, we found that this was not always true. We have shown in [5] that Wadsworth's *fully lazy* interpreter [18] shares more sub-expressions than Henderson and Morris' *lazy* interpreter [10] at each reduction step. Furthermore, we have given a λ-calculus expression for which the *fully lazy* interpreter takes more steps than the lazy interpreter. This led us to define the notion of *weak normal forms* for which it is in fact true that an interpreter that does more sharing takes fewer steps to find the answer. The importance of weak normal forms is that practical interpreters return weak normal forms as answers. We then showed that it is possible to transform any expression so that a lazy interpreter reducing the transformed expression would

share the same sub computations as a fully lazy interpreter reducing the original expression. Finally, the effect of representing λ-expressions as combinatory forms was explored. It was found that sharing was affected not by the alternative representation itself but by the abstraction algorithm used to transform the λ-expression into the combinatory form.

3. THE MULTIPROCESSOR EMULATION FACILITY

3.1. The Emulator as a Prerequisite to the Dataflow Machine

The Tagged Token Dataflow Machine and associated programming environment represent a radical departure from both the hardware and software for parallel processing based on von Neumann processors communicating via shared memory. A real challenge in building the first Tagged Token Dataflow Machine is to solve two highly interrelated problems. Firstly, there are few large dataflow programs whose dynamic behavior is well understood; estimates of dynamic behavior must be based exclusively on the analysis of the source code expressed in dataflow languages because of a lack of facilities for executing these programs. This method of gaining insights into the behavior of dataflow programs can be applied to a very limited number of application programs because it requires close cooperation of dataflow and application experts. Application experts have little motivation to spend time to understand the behavior of their applications on "hypothetical computers". Secondly, the architecture of the Tagged Token machine is so different from conventional processors that it is difficult to estimate some architectural parameters (e.g., size of buffers) whose effect on the overall performance of the machine may be critical, without some concrete assumptions about the dynamic behavior of programs. One way we are trying to resolve this paradoxical situation is by implementing the Tagged Token Dataflow Machine on a multiprocessor emulation facility.

3.2. The Emulator as an Interesting Parallel Processor

In addition to providing a highly productive vehicle for the direct emulation of large scale dataflow programs, the Emulator is an interesting and innovative multiprocessor in its own right. It is the first MIMD machine which can support general, distributed, asynchronous processes that require relatively large amounts of interprocess communication *and* can do so much more rapidly and be more cost-effective than the equivalent simulation on a single very large SISD machine. This unique characteristic is a result of two fundamental design decisions.

First, the Lisp workstations being used to build the facility are themselves powerful machines with sophisticated programming environments. The power lies in the

supermini class internal architecture of a high-performance micro-engine, wide data paths, and a high-speed memory subsystem. The machine also has an I/O structure capable of supporting a high-performance interprocessor communication mechanism. The associated Lisp system provides a good environment for the development of cooperating asynchronous processes, and thus also a good vehicle for architectural exploration, assuming that each processing element (or each of its subsystems) is viewed as a process.

Second, the interprocessor communication networks provide an instrumented, flexible, and inherently fault-tolerant data transport mechanism that is well-matched to the processor's throughput. The network is easily reconfigured and allows evaluation of a variety of communication strategies including perfect n-cube, planar, token ring(s), and shuffle exchanges. This permits empirical exploration of the effects of interconnect topology and bandwidth on the overall machine performance. The communication controller design is matched to the high-performance of the Lisp Machine. It provides its own horizontal microcontrollers to permit transmission and reception of messages concurrently with the normal program execution. Each switch node supports an aggressive instrumentation, test, and maintenance subsystem as well as managing the routing tables for the currently supported network topologies.

In a sense, these two features are not new - at least when taken independently. It is precisely the fact that mature, high-performance von Neumann machines like the Lisp Machine and the theory of n-dimensional packet switching *exist* that reduces the risk, and thus the time to completion, of the Emulator. What makes this interesting is that this level of power, a necessary level to run meaningful programs, has never been assembled in such a flexible and balanced fashion. Previous implementations have either lacked processor performance, productive programming environments or, the common problem, a well-matched communication system that does not impose excessive processor overhead. Compromises in design either due to cost or narrow intent haven taken their toll. This really marks the first time that a parallel machine can directly execute and instrument a reasonably broad class of distributed programs *significantly faster than an equivalent simulation on the largest uniprocessor systems.*

3.3. Hardware Development

We view the Emulator as being an evolutionary step toward our goal of building a VLSI Tagged Token Dataflow processor. It will allow us to properly study this radically different architecture without the usual constraints and commitment of resources that are required for a typical hardware project. Also, the component of the Emulator which is being *invented*, namely the communications network, is crucial to the realization of a Tagged Token Dataflow Machine. The network is being

designed in three phases, all oriented toward developing VLSI for packet communication over high-speed serial point to point circuits. The *circuit switch* is the first phase, while the *packet switch* and its VLSI version are the final two phases.

The Circuit Switch: In an effort to get a network operational, we decided to adapt the network used by Bolt, Beranek, and Newman in their Butterfly multiprocessor [14]. The switching node in this network is a 4x4 crossbar with 4 bit wide data paths. These nodes are interconnected and centrally clocked to form a synchronous network of arbitrary topology. Messages traversing the network carry an encoding of the network path they are to take. The adaptation of the BBN design required re-engineering the switch so that the network will be built solely out of cards plugged directly into the backplanes of our Lisp Machines. As a consequence, the design had to allow for longer inter-switch cables. More importantly, we are designing the necessary buffering and control logic so that the network presents as little processing load on the associated processors as possible. Greg Papadopoulos and Eric Hagersten have been doing this design.

We believe there is a high probability that a system based on the circuit switch will be operational within a year, as opposed to a system based on the packet switch which may not be available for two years. In a move to reduce duplication of effort, we have constrained both switch designs to share major subsystems. Our re-implementation of the BBN circuit switch has proceeded smoothly since September. We now have three of the four major subsystems designed and entered into our Computer Aided Engineering station. An extensive discussion of the development plan for the circuit switch can be found in [13].

Packet Switch: We think that a good network structure for the Emulator is a hypercube of high speed, bidirectional, serial interconnections with full *store and forward* logic at each switching node [12]. The exploratory design work by Robert Iannucci has shown the approach to be within the capability of modern engineering techniques [11]. Nevertheless, to achieve the target speed and reliability, the serial link drivers and receivers within the switch will be implemented with a mixture of analog and digital circuits. Further, the internal logic of the switch places strong demands on the circuit technology for speed and, more critically, predictability. Off the shelf logic typically has a very wide spread of "acceptable" performance specified by the manufacturer. This large spread, when combined with the desire to produce a robust design, forces the use of worst-case performance figures. This typically translates directly into a bloating of the design in terms of chip count. It should be noted that we are interested in medium sized volumes of assembled and *tested* switch cards because of our own needs and anticipated need for those who wish to replicate the Emulator.

To solve these problems, we have reached an agreement with IBM under which

they have installed a department of four full-time engineers at MIT for three years. These engineers will help in the design, prototyping construction, and testing of this packet network. We are studying the possibility of using an IBM serial data communications circuit of their own design which more than meets our performance goals. In addition, IBM will give us access (indirectly through their engineers) to a medium capacity bipolar gate array technology and to their Engineering Design System. This should solve our circuit testing problem because IBM will provide us with tested gate array chips.

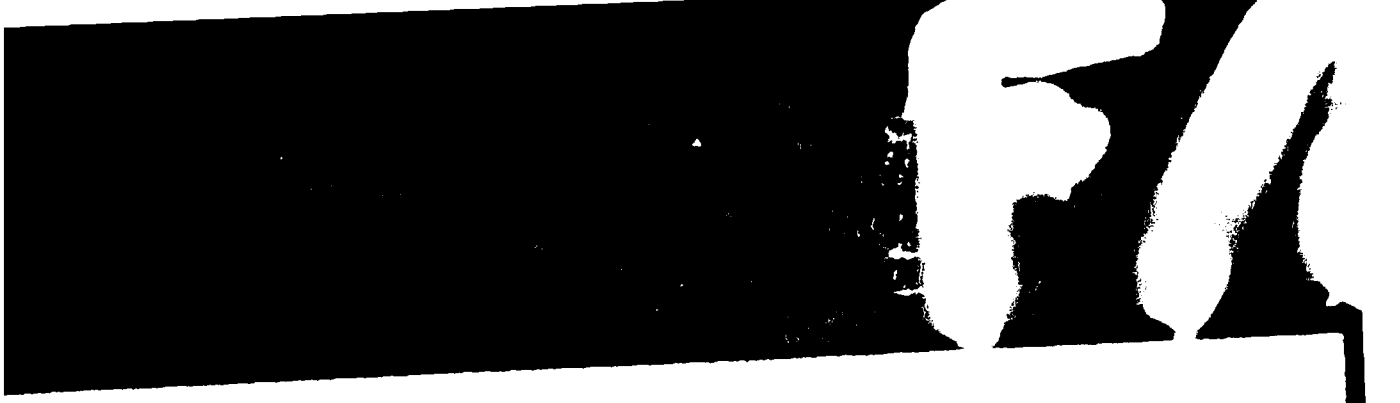
During the spring of 1983 a project to study the feasibility of a VLSI version of the bit-serial transciever was undertaken in the advanced VLSI subject at MIT. It resulted in the preliminary design of a 100 MHz serial data communications circuit based on the MOSIS 1.2 μm two layer metal cMOS process [7]. The transmitter section develops a Manchester coded data stream using asynchronous finite state techniques. The receiver re-extracts data and clock from the Manchester waveform using an on-chip phaselock loop.

We are continuing this VLSI effort because the bit-serial transreciever will be a retrofitable part of this switch and has applications far beyond the Packet Switch. In VLSI chips, the transistor to I/O pin ratio will continue to increase and thus, as the use of pins for inter-chip communication will rely more heavily on time multiplexing techniques in the future. Our VLSI version of the bit serial transreciever set is very robust and implements full flow control. It will allow a multichip system to be designed according to the *synchronous on-chip, a synchronous across-chip* methodology. Thus, traditional chip design techniques can be employed and the need for centralized clocking eliminated.

3.4. Emulation Software

The Multiprocessor Emulation Facility, developed by Richard Soley, is a large body of Lisp software which allows its user to quickly and easily prototype a single- or multiprocessor computer architecture, and to then execute the proposed machine in emulation. In order to achieve high emulation speed without excessive cost, and to force systems architects to begin thinking of computers in a distributed, multiprocessing way, the Emulation Facility is designed to execute on many parallel Lisp processors, linked via a variety of communications media.

The general case of a multiprocessor is a group of various asynchronous independent *processing elements* with no shared state, connected by a packet communications network of arbitrary connectivity. MEF supports this type of architecture directly, by providing a set of software abstractions which facilitate the modeling of such an architecture in the Lisp language. The abstractions include programs to define *experiments* and *logical processors*. An *experiment* consists of



the number and type of processors being emulated, their interconnection topology, and other information pertinent to running the architecture in emulation. A *logical processor* is a Lisp program which exhibits the behavior of a physical processor in the multiprocessor architecture by modifying local data representing the state of the processor, and communicating with other processors by calling MEF's message sending primitives. The MEF will also provide support for dealing with the lower-level issues of handling networks and protocols used on them, configuring emulated topologies on top of different physical interconnections of the Lisp Machines, etc.

In addition to these tools, the system includes more mundane subsystems, such as the *Control Panel*, which acts as the bootstrap-load processor of the system, configuring the Lisp Machines taking part in a particular emulation experiment, broadcasting the experiment to be executed, and setting up the communications media for the desired interconnection topology. The control panel subsystem also provides primitives for collection and display of various statistics, either during execution, or afterward.

The MEF system automatically distributes logical processors in the experiment over the physical Lisp Machines currently configured into the facility. In this way, an experiment that requires sixteen logical processors can be run on the facility using one, two, or more physical Lisp Machines. In addition, the Lisp Machines may be partitioned into separate sub-facilities by the control panel, allowing multiple emulation experiments to run at the same time.

Although the abstract structure outlined above was chosen for its generality in emulating multiprocessor configurations, the skeptic will immediately note that it does not directly support such multiprocessor designs as synchronous processors (like the Connection Machine, or Illiac IV) or the shared memory model (e.g., C.mmp). In fact, this model is abstract enough to allow prototypical implementation of even these models of parallel computation; generally, another virtual processing element is added to the emulation experiment definition to model this shared resource (e.g., clock or memory) of the system. This is actually quite close to the real hardware implementation of such a system, in which a central clock for synchronous operation or a central shared memory will actually be a separate subsystem of the architecture. In addition, interconnection schemes other than packet-switching networks can be simulated on top of our packet switch by using the packets to simulate individual items in a continuous stream communication mechanism.

The implementation of the Multiprocessor Emulation Facility is proceeding smoothly. We currently have an emulation system, written entirely in ZetaLisp, running on five Symbolics 3670 processors. We continue to utilize the ten megabit Ether network, executing Chaos protocols, for our communications medium. As

work progresses on the circuit and packet switch hardware, we are in the midst of preparation for use of those new and faster media.

We currently have two architectures in emulation. The first, written by Richard Soley, is a simple von Neumann style machine, emulated as two logical processors (one *CPU* and one *MEMORY* box) connected via a *bus* (all communications actually pass through the Emulation Facility software, as described above). This simple emulation provided a vehicle for debugging parts of the system.

We also have an emulated version of the Tagged Token Dataflow Machine running on the Emulation Facility. This emulation, developed by Richard Soley, Paul Fuqua, and Poh Lim, fully supports our current definition of the Tagged Token Dataflow Architecture. We are already using the *dataflow emulation experiment* to tune the design of the Tagged Token Dataflow Architecture. We are executing Id programs, compiled into machine code, at the rate of one hundred dataflow machine instructions per second.

We have an immediate pressing need for more speed in the dataflow emulation; therefore, selected parts of the emulation facility and the *dataflow experiment itself* are being hand-tuned to reach the goal of one thousand emulated dataflow machine instructions per second.

4. RELATED TOPICS

4.1. Logic Programming

A great deal of attention has recently been given to the Fifth Generation Computing project in Japan, mostly because of its focus on Logic Programming as the programming method of choice. Logic Programming shares the "single assignment" characteristic of Functional Programming, as well as the declarative style; hence, it is of great relevance to our group. In order to learn more about Logic Programming, during the IAP in January '84, an informal, week long workshop was held. Talks were presented by Jan Komorowski, who was invited from Harvard, Michael Beckerle, Gary Lindstrom (a visiting scientist from the University of Utah), Arvind, and Gordon Robinson (from the AI Laboratory). The talks covered topics ranging from the fundamentals of logic programming to current efforts in the parallel processing of logic programs.

4.2. New Equipment

Five Symbolics 3600s were acquired as the first of 64 machines for the Multiprocessor Emulation Facility. These machines are in the process of being

upgraded to 3670s. Each machine has 6 megabytes of main memory and one machine is equipped with 500 megabytes of disk storage for file service. An additional 1.6 gigabytes of disk memory was installed on the group's VAX-750, also for file server use by the 3670s. A total of sixteen 3670s are expected by the end of this year. Several IBM PCs, networked onto TCP/IP-speaking Ethernet, were also obtained. The PCs are used for local editing, software development, and laboratory instrumentation control.

4.3. Support Tools

As his first UROP project, Dinarte R. Morais, an undergraduate member of the group, implemented an interactive illustrator program, which is now being used by members of both the LCS and AI labs. The program, called ILLUSTRATE, is an interactive illustrator for creating pictures composed of lines, curves and text captions, and was modeled after the DRAW program available on Alto computers, as described in the Alto User's Handbook.

The problems with using the existing DRAW program on the Alto computers include the fact that pictures could not be very complicated due to the relatively small amount of memory available. In addition, DRAW came *as is* and consequently could not be customized. Finally, the few Alto computers left are getting older and less reliable, and it was hoped that ILLUSTRATE could allow our group to finally do away with the Altos.

ILLUSTRATE was implemented in ZetaLisp on a Symbolics 3600 Lisp Machine. The advantages of ILLUSTRATE over DRAW are many. One advantage is that there is no practical limit to the complexity of a picture created with ILLUSTRATE. More importantly, we now have the ability to customize the program to suit our needs.

References

1. Arvind and Iannucci, R. A. "Instruction Set Definition for a Tagged-Token Dataflow Machine," Massachusetts Institute Technology Laboratory for Computer Science, Computation Structures Group Memo 212-3, Cambridge, MA, February 1983.

2. Arvind, Culler, D.E., Iannucci, R.A., Kathail, V., and Pingali, K. "The Tagged Token Dataflow Architecture," to appear as an MIT/LCS/TM.

3. Arvind, Gostelow, K.P. and Plouffe, W. "An Asynchronous Programming Language and Computing Machine," University of California Technical Report 114a, Department of Information and Computer Science, Irvine, CA, December 1978.

4. Arvind, and Iannucci, R.A. "A Critique of Multiprocessing von Neumann Style," *Proceedings of the Tenth International Symposium on Computer Architecture*, Stockholm, Sweden, June 1983.

5. Arvind, Kathail, V. and Pingali, K. "Sharing of Computation in Functional Language Architectures," *Proceedings of the Workshop on High-level Language Architectures*, Los Angeles, CA, May 1984.

6. Backus, J. "Can Programming be Liberated from the von Neumann Style?" *Communications of the ACM* 21, 8 (August 1978) 613-641.

7. Bassett, P.D, Geldens, P.M., Goodhue, J.T. and Iannucci, R. "Design of a 100 MHz cMOS Phaselocked Manchester Encoder/Decoder Circuit," Term Paper in VLSI Subject 6.372, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, 1982.

8. Burge, W.H. *Recursive Programming Techniques*, Reading, MA, Addison-Wesley Publishing Company, 1975.

9. Heller, S. and Arvind "Design of a Memory Controller for the the Massachusetts Institute of Technology Tagged Token Dataflow Machine," Computation Structures Group Memo 230, MIT Laboratory for Computer Science, Cambridge, MA, October 1983.

10. Henderson, P. and Morris, J.H. "A Lazy Evaluator," *Conference Record of the Third ACM Symposium on Principles of Programming Languages*, 95-103, Atlanta, GA, January 1976.

11. Iannucci, R. "Packet Communication Switch for a Multiprocessor Computer Architecture Emulation Facility," Computation Structures Group Memo-220, MIT Laboratory for Computer Science, Cambridge, MA, October 1982.

12. Ng, G.W. "Design of a Packet Communication Switch for a Multiprocessor Computer Architecture Emulation Facility Part 1: Clock Subsystem, Functional Languages and Architectures Group Design Note 2, MIT Laboratory for Computer Science, Cambridge, MA, November 1983.

13. Papadopoulos, G.M., Iannucci, R.A. and Chiang, C.J. "Preliminary Design for MEF Near-Term Communication Switch," Functional Languages and Architectures Group Design Note 4, MIT Laboratory for Computer Science, Cambridge, MA, January 1984.

14. Rettberg, R., Wyman, C., Hunt, D., Hoffman, M., Carvey, P., Hyde, B., Clark, W. and Kraley, M. "Development of a Voice Funnel System: Design Report," Bolt Beranek and Newman, Inc. Technical Report 4098, Cambridge, MA, August 1979.

15. Traub, K.R. "An Abstract Architecture for Parallel Graph Reduction," MIT/LCS/TR-317, MIT Laboratory for Computer Science, Cambridge, MA, September 1984.

16. Turner, D.A. "A New Implementation Technique for Applicative Languages," *Software -- Practice and Experience* 8 (1979) 31-49.

17. Turner, D.A. "The Semantic Elegance of Applicative Languages," *Functional Programming Languages and Computer Architecture* 85-92, October 1981.

18. Wadsworth, C.P. *Semantics and Pragmatics of the Lambda-Calculus*, University of Oxford Press, Oxford England, 1971.

Publications

1. Arvind, Dertouzos, M.L. and Iannucci, R.A. "Multiprocessor Emulation Facility," MIT/LCS/TR-302, MIT Laboratory for Computer Science, Cambridge, MA, September 1983.

2. Arvind and Iannucci, R.A. "Two Fundamental Issues in Multiprocessing: The Dataflow Solution," MIT/LCS/TM-241 MIT Laboratory for Computer Science, Cambridge, MA, September 1983.

INFORMATION MECHANICS

Academic Staff

E. Fredkin, Group Leader

Research Staff

T. Toffoli

G. Vichniac

Graduate Student

N. Margolus

Undergraduate Students

S-W. Chen

C. Ferreira

Support Staff

R. Hegg

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

Talks

1. Gifford, D. "The Technology Behind the Boston Community Information System,"

   University of Washington, May 1984.
   University of California, May 1984.
   Stanford University, May 1984.
   IBM T. J. Watson Research Center, April 1984.
   Carnegie-Mellon University, April 1984.

2. Gifford, D. "Data Communication Aspects of the Boston Community Information System," AT&T Bell Laboratories, February 1984.

3. Gifford, D. "The Boston Community Information System," Xerox Palo Alto Research Center, November 1983.

Publications

1. Gifford, D. K. and Spector, A. Z. (ed.) "The TWA Reservation System," *Communications of the ACM*, (July 1984).

2. Lucassen, J. "The Personal Information System," Imaginative Systems Group Memo No. 1, MIT Laboratory for Computer Science, Cambridge, MA, February 1984.

3. Stamos, J. W. "Static Grouping of Small Objects to Enhance Performance of a Paged Virtual Memory," *ACM Transactions on Computer Systems*, 2, 2 (May 1984).

Thesis Completed

1. Lamb, C. "A Screen Oriented Data Base Editor," S.B. and S.M. theses, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1983.

Theses in Progress

1. Carnese, D. "An Analysis of the Type Systems of Five Contemporary Programming Languages which Support Synthetic Type Definition," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1984.

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

3. Gunther, B. "A Personal Database System for the IBM XT," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1985.

4. Hyre, R. "Personal Map Systems," S.B. thesis MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1985.

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

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

4. PARTIAL EVALUATION AND PROGRAMMING LANGUAGE DESIGN

This is a new activity in our group, and is work that has been done in conjunction with two of my graduate students, John Lucassen and Richard Schooler. The basic thesis of the work is that if built-on types are roughly as efficient as built-in types then programming languages will be more flexible and able to meet the needs of their users than is possible to achieve with current techniques. The optimization technology that we have been exploring is called partial evaluation because it evaluates a program as best as possible given partial information. The language that we are using for our work on partial evaluation is called IBL, and it strongly resembles Scheme. In the past year we have completed a partial evaluator that has successfully optimized programs that used a message passing system, and the partial evaluator has also converted an interpreter for a simple declarative rule-based language into a compiler for the same language.

Our plan for next year is to complete a front-end for our language work that will transform Modula-2 into IBL. All type computations will be made explicit in the transformation. Because Modula-2 is strongly typed all type computations and checks will be performed by partially evaluating the IBL result of the front-end.

2.4. Plans for the Next Year

In the next year, we plan to improve the system we are operating in a number of ways. First, we plan to add new information sources, including the Associated Press. We have also started a joint effort with the Boston Convention and Tourist Bureau to add visitor and event information to our system. Second, we plan to add personal computer access to our central site database system. This will allow information that was missed when it was broadcast or historical information to be retrieved in the same framework that is used in our present personal database system. Finally, we plan to evaluate the experiences of our initial test audience.

3. ADVANCED GRAPHICS SUPPORT FOR USER INTERFACES

In support of our work on community information systems we have been working on new types of user interfaces. One application that we have built is a browser for a street map of the Boston area. It permits one to look at a digital map of the entire area, zoom in on locations, and find a place given a street address. The original implementation of the Boston map system was on a Symbolics 3600, a fairly expensive personal machine. The map system work is continuing on low-cost MacIntosh personal computers, and we are exploring ways of integrating it with new databases that we will be receiving from the Boston Convention and Tourist Bureau and the Massachusetts Bay Transportation Authority.

In a related project we are trying to discover what user interface metaphors will be useful beyond the window system and mouse ideas that were made popular by bit-map displays. To do this, we are using an advanced display system provided by the combination of a Symbolics 3600 and the Silicon Graphics IRIS system. The Silicon Graphics IRIS is a polygon display engine that will display an animated color scene of 300 polygons in real time.

In the past year, we have completed a 3D graphics library for the Symbolics 3600 that sends graphics commands to the IRIS over a serial line. This link has been used to show portions of 3D databases that we have received from NASA, including the space shuttle. Our goals for next year for the 3600/IRIS system are to have the 3600 communicating with the IRIS via an IP/TCP byte stream on an Ethernet, to fully specify and implement a graphics library that permits direct specification of kinematics, and to bring up a simple application that demonstrates the capabilities of the foundation that we are building.

character buffer is chosen to be large enough to buffer characters for the maximum latency between service times from the receive task. Once the receive task starts running it begins to empty the character buffer. The character stream is reassembled into packets, the checksum on the packets is checked, and the packets are reassembled into a complete segment in a buffer area. If the segment is encrypted then the proper key is computed and the segment is decrypted in place.

Once a segment has arrived and has been decrypted it is checked against the user's filter. The algorithm that is used to check a text segment against a user's keyword filter is very efficient, and runs in time linear in the size of the segment. Thus users are not penalized for having complex filters. If an incoming segment does not match a user's filter, it is discarded. However, if the segment does match the filter it is written to a new file on the local disk and the file is indexed in the local database directory.

As mentioned earlier, while all of this is going on the user task services requests from the keyboard. The user interface allows users to type in queries that are the same as filter predicates. Queries are boolean combinations of words, phrases, and specifiers of subject, author, source, importance, and so forth. Once a query is entered it is decomposed and the local database directory is searched for matching entries. The entries that result are displayed in a menu format on the screen. At this point a user can select a specific entry for display by adding its number to the query that is in the input line, or the user can enter an entirely new query. The user interface includes a simple window manager that responds to page up and page down keys in the event output will not fit on a single screen. There is also a built in help facility that will provide instruction on how to use the system.

2.3. Digital Radio Receiver Design

To support an experimental user population we have designed and built 20 receivers for our broadcast digital signal (D. Gifford). The receiver is in a small box that has input connections for an antenna and power and it has a single RS-232 compatible output connection that plugs into the serial port of a home computer. The parts cost of the receiver is approximately $130. This includes a standard FM receiver front-end card that we buy from an outside supplier, a power supply, and our own PC card that implements the subcarrier demodulator.

The subcarrier demodulator is comprised of seven integrated circuits and an assortment of discrete components for the analog part of the board. The subcarrier demodulator works by taking the output of a standard FM receiver and first removing the normal programming on the channel with a 4-pole active filter. The output of the filter is sent to an FSK receiver chip that uses a phased-locked-loop to recover the data and provide an RS-232 signal on the output terminals of the receiver.

The flow of information in our system is as follows. Information is received at our central database system at MIT from a number of sources, including the New York Times. The information is put into standard form and placed in a database. This database can be searched on-line and users can also keep profiles of their interests and the system will automatically send them a message when information arrives that matches their profile. In addition, a program called the scheduler (S. Berlin) monitors arrivals to the database and schedules them for transmission out over our broadcast channel to remote personal computer users. A typical information item is transmitted a number of times, but the frequency of transmissions can depend on the type and priority of the information item.

The communication channel that we use to communicate with remote users is a 4.8 KBit/second broadcast channel that is piggybacked on a normal FM broadcast station. The channel uses asynchronous signalling to keep the cost of receiver hardware low; the receiver that we have designed plugs directly into the RS-232 port of personal computers.

To use the communication channel effectively we have developed a broadcast packet protocol. This protocol allows multiple data streams to be multiplexed on a single channel, and provides for packet error detection and error recovery. A theoretical model of the communication channel that we developed allowed us to compute the optimum packet size for our system. The model requires the byte-error rate as an input parameter. Empirical tests were done at various locations in the Boston area to compute the byte error rate of our system.

The next layer of protocol up from packet transport provides for the transport of arrays of bytes called segments. Software above the segment layer is insensitive to the packet size that we use and the number of packet retransmissions that are done to enhance reliability. The segment layer also provides for encryption. A master/sub-master scheme is used so that the compromise of a single segment key will not disclose a master key.

2.2. Personal Database Software

The personal database software that we have completed (J. Lucassen) serves two functions. First, the receive task listens to the packet radio broadcasts and keeps the local database up-to-date. Incoming database updates are incorporated if they pass a filter that the user has specified. At the same time that the receive task is keeping the database up-to-date, the user task processes queries from users. These two tasks run as separate processes under MS-DOS using a package written by the Computer Systems and Communications Group of our Laboratory.

The receive task services a character buffer that is filled at interrupt level. The

1. INTRODUCTION

The Imaginative Systems Group is involved in a project to build and experiment with new types of man-machine communication systems. The first system that we have built is a large-scale community information system that is now operating in the Boston area. As described below, information is transmitted via a broadcast packet-radio system to home computers where it is received by a customizable personal database system. The next section describes this work in detail. A second project that we have been involved in is the optimization of programming languages by a technique called partial evaluation. The results of the first partial evaluator that we built were encouraging, and we are proceeding to build a front-end for a contemporary programming language to test our ideas further. The second section of our group's progress report gives a description of what we have accomplished in this area in the past year.

2. THE BOSTON COMMUNITY INFORMATION SYSTEM

As described in our original DARPA proposal, we are engaged in building a new type of community information system. The system has a number of novel aspects that differentiates it from other such systems. First, it uses a synthesis of uni-directional (broadcast) packet radio communication and personal computers to provide a customized information service. Second, because uni-directional communication is used the system can be used by an arbitrary number of users. Third, an extensive protection mechanism has been specified and implemented that provides fine-grained access control. Access can be controlled on the basis of information source, or access can be limited to certain time periods. Finally, we provide a sophisticated query interface based on free-text searching as opposed to simpler menu-based approaches used in other community information systems.

In the past year we have begun regular transmissions of information over our broadcast packet-radio system, completed the design and implementation of our personal database software for the IBM PC, and finished the first build and checkout of 20 data receivers for our test of the system this summer. The following paragraphs describes each of these accomplishments in more detail.

2.1. Broadcast Packet Radio System

Our regular transmissions to home computers began on April 15, 1984. The FM radio station that we use, MIT's WMBR, is on the air for approximately 14 hours a day. During this time we transmit our packet protocol on their subcarrier. Our tests in the field show that on WMBR's 200 watt transmitter with 10% subcarrier injection we achieve a primary service area that lies within five miles of the transmitting tower in Kendall Square, Cambridge.

IMAGINATIVE SYSTEMS

Academic Staff

D. Gifford, Group Leader

Research Staff

S. Berlin

Graduate Students

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

Undergraduate Students

K. Buttner
B. Gunther
F. Huettig
R. Hyre
D. Lane
E. Olson
R. Rabines
J. Yoon
R. Zee

Support Staff

R. Bisbee

21. Pingali, K.K. "Demand Driven Evaluation on Dataflow Machines," University of California, Irvine, CA, May 27, 1984.

22. Pingali, K.K. "Demand Driven Evaluation on Dataflow Machines," IFIP Working Group 2.2 Meeting, MIT Endicott House, Needham, MA, June 12, 1984.

23. Soley, R. M. "A General Multiprocessor Emulation Facility," First Annual ACM Northeast Regional Conference, University of Lowell, Lowell, MA, March 20, 1984.

8. Arvind "The Tagged-Token Dataflow Machine," University of Delaware, DE, March 19, 1984.

9. Arvind "Fundamental Issues in the Design of Multiprocessor Computers," University of Texas, Austin, TX, March 26, 1984.

10. Arvind "The Dataflow Solution," University of Texas, Austin, TX, March 27, 1984.

11. Arvind "Sharing of Computation in Functional Language Implementations," International Workshop on High-Level Computer Architectures, Los Angeles, CA, May 24, 1984.

12. Arvind "Sharing of Computation in Functional Language Implementations," IFIP Working Group 2.2 Meeting, MIT Endicott House, Needham, MA, June 12, 1984.

13. Brobst, S. A. "Simulation Techniques in the Design of a Data Flow Supercomputer," 1984 Summer Computer Simulation Conference, Boston, MA, July 24, 1984.

14. Culler, D. E. "Why Dataflow Architectures?," Fourth Jerusalem Conference on Information Technology, Jerusalem, Israel, May 1984.

15. Heller, S. K. "Design of a Memory Controller for the MIT Tagged Token Dataflow Machine," IEEE International Conference on Computer Design, Portchester, NY, October 31, 1983.

16. Iannucci, R.A. "VLSI: The Next Cottage Industry?," IBM Glendale Laboratory, Endicott, NY, September 14, 1983.

17. Iannucci, R.A. "Phaselocking for Fun and Profit," MIT VLSI Design Review, Cambridge, MA, December 1983.

18. Iannucci, R.A. "Dataflow Architecture: an Introduction," IBM Glendale Laboratory, Endicott, NY, December 1983.

19. Iannucci, R.A. "Dataflow Architecture: an Exercise in Top-Down Design," IBM Glendale Laboratory, Endicott, NY, December 1983.

20. Papadopoulos, G.M. "Dataflow Models for Fault-Tolerant Control Systems," American Control Conference, San Diego, CA, June 1984.

Theses in Progress

1. Beckerle, M. J. "The Graph Resolution Method for Logic Programming," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected April 1985.

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

3. Culler, D.E. "Resource Management for the Tagged-Token Dataflow Architecture," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1984.

4. Soley, R. M. "Generic Software for the Emulation of Multiprocessor Architectures," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1984.

5. Vafa, B. "A Resource Management Policy for the Tagged-Token Data Flow Machine," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected December 1984.

Talks

1. Arvind, "The Tagged-Token Dataflow Machine," Honeywell Avionics Division, Minneapolis, MN, August 19, 1983.

2. Arvind "The Tagged-Token Dataflow Machine," The Sixteenth IEEE EASCON, Washington DC, September 19, 1983.

3. Arvind "Two Fundamental Issues in Multiprocessing," The 1983 IFIP's Congress, Paris, France, September 23, 1983.

4. Arvind "The Tagged-Token Dataflow Machine," IBM-Research, Zurich, Switzerland, September 26, 1983.

5. Arvind "The Tagged-Token Dataflow Machine," General Electric, Schenectady, NY October 18, 1983.

6. Arvind "The Tagged-Token Dataflow Machine," ICOT, The Institute for New Generation Computers, Tokyo, Japan, January 18, 1984.

7. Arvind "The Tagged-Token Dataflow Machine," Yale, New Haven, CN March 7, 1984.

13. Pingali, K. and Arvind, "Efficient Demand-Driven Evaluation (II)," MIT/LCS/TM-243, Laboratory for Computer Science, Cambridge, MA September 1983.

14. Pinkerton, J. T., Iannucci, R.A. and Papadopoulos, G. M. "A Comprehensive Hardware Laboratory for the Multiprocessor Emulation Facility," MEF Design Note #4, MIT Laboratory for Computer Science, November 1983.

15. Soley, R. M. "A Third Opinion on Dataflow Machines and Languages," to appear in *IEEE Computer.*

Theses Completed

1. Fuqua, P.C. "Emulating the I-Structure Memory for the Tagged-Token Dataflow Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

2. Muriph, S. W. "Optimized Execution of the APL Structured Functions," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

3. Chambers, T. B. "A Database System for Parameters of Data Flow Machine Simulation," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

4. Desai, E. D. "Specification for a High Speed Point to Point Serial Data Communication Circuit," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

5. Douglass, S. A. "Demand-driven Efficiency on Dataflow Machines," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

6. Traub, K. R. "An Abstract Architecture for Parallel Graph Reduction," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

7. Ying, J. J. "Instrumentation to Collect Statistics for a Multiprocessor Emulation Facility," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

3. Arvind, Kathail, V. and Pingali, K.K. "Sharing of Computation in Functional Language Implementations," *Proceedings of the Workshop on High-Level Language Architectures*, Los Angeles, CA, May 1984.

4. Arvind and Culler, D.E. "Why Dataflow Architectures?" *Proceedings of the Fourth Jerusalem Conference on Information Technology*, Jerusalem, Israel, May 1984. (Also CSG Memo 229-1)

5. Bassett, P. D., Geldens, P., Goodhue, J.T. and Iannucci, R.A. "A 100 MHz. cMOS Manchester Encoder/Decoder Circuit" MEF Design Note #6, MIT Laboratory for Computer Science, Cambridge, MA, May 1983.

6. Brobst, S.A. "Simulation Techniques in the Design of a Data Flow Supercomputer," *Proceedings of the 1984 Summer Computer Simulation Conference*, Boston, MA, July 1984.

7. Desai, E. D. and Pinkerton, J.T. "Design of a Packet Communication Switch for a Multiprocessor Computer Architecture Emulation Facility *Part 2: Input FIFO, Output Buffer, Sequencer, and Scheduler*, MEF Design Note #3, MIT Laboratory for Computer Science, Cambridge MA, September 1983.

8. Desai, E. D. "Specification for a High Speed Point to Point Serial Data Communication Circuit," MEF Design Note #8, MIT Laboratory for Computer Science, Cambridge, MA, May 1984.

9. Heller, S.K. and Arvind, "Design of a Memory Controller for the MIT Tagged-Token Dataflow Machine," *Proceedings of IEEE ICCD 83*, Portchester, NY, October 1983. (Also CSG Memo 230)

10. Ng, G., "Design Of A Packet Communication Switch For A Multiprocessor Computer Architecture Emulation Facility *Part 1: Clock Subsystem*" MEF Design Note #2, MIT Laboratory for Computer Science, Cambridge, MA, September 1983.

11. Papadopoulos, G.M. and Arvind "Dataflow Models for Fault-Tolerant Control Systems," *American Control Conference Proceedings*, San Diego, CA, June 1984.

12. Pingali, K. and Arvind, "Efficient Demand-Driven Evaluation (I)," MIT/LCS/TM-242, Laboratory for Computer Science, Cambridge, MA, September 1983.

1. INTRODUCTION

One of the goals of information mechanics is to explore the possibility of virtually nondissipative computing. To this end, it is necessary to establish a very close match between the logical structure of the desired computation and the physical structure of the computer that should carry it. This approach has given significant insights into the information-processing aspects of reversible dynamical systems. It has also provided novel, conceptually attractive tools in the mathematical modeling of physics.

The theoretical aspects of our work were complemented and stimulated by work concerned with the efficient implementation of such abstract tools by means of special-purpose machines.

2. A MATURE VERSION OF THE CELLULAR AUTOMATON MACHINE

After three years of design and development, CAM (Tom Toffoli's high-performance machine dedicated to the simulation of cellular automata and other distributed discrete dynamical systems) this spring has reached a mature form under which it will get into the world. System Concepts (San Francisco, CA) is now building a version that fits a single slot of the IBM PC. This version should be commercially available this fall. Norm Margolus has provided a FORTH package to drive the new CAM.

3. A NEW CLASS OF CELLULAR AUTOMATA

Our research this past year has received a big boost following the discover (by Norm Margolus) of a new family of cellular automata based on local functions with equal numbers of inputs and outputs [3]. In these "partitioning" cellular automata (PCA), many properties of the overall evolution carry over in a straightforward manner from those of the local evolution. For example, if the local mapping is invertible, so is the global mapping.

The Billiard-Ball-Model-Cellular-Automaton (BBMCA), an instance of PCA provides a basis for universal computation since it simulates a classical mechanical model of computation [3]. It thus constitutes a laboratory in which to study, among other things, the relevance of energy and entropy analogues in computer models. The BBMCA also yields a description of deterministic computations which could, in principle, operate within the constraints of a local quantum-mechanical Hamiltonian [4]. We are studying 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.

4. PARALLEL COMPUTATION OF THE DYNAMICS OF DISTRIBUTED SYSTEMS

Cellular automata provide a stimulus to designing algorithms which effectively exploit the potential of massively parallel hardware. Gerard Vichniac [5] has studied the effects of synchronous updating in a regular network of locally interconnected Boolean variables (i.e., Ising spins). Surprisingly, even in this very simple instance of distributed system, one must strictly follow somewhat counter-intuitive rules in order to make a full use parallel computational resources [1].

5. DISCRETE REPLACEMENTS

While digital computers are discrete objects, the language of most mathematical physics employs continuum structures (viz., differential equations). 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 -- we have investigated finitary models based on the combinatorics of a large number of discrete variables. We have found, using PCAs, that the 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.

We have continued this year our program of reducing concepts from physics to computational primitives such as counting, labeling, and comparing [5][2]. We recently found that any iterated deterministic manipulation on a random configuration of symbols is analogous to the *quenching* of a disordered system to low temperatures. This sheds some new light on the processes of ordering via domain growth and interface motion.

6. THE QCD MACHINE PROJECT

We have been involved since January 1984 in the design and construction of a special-purpose machine for calculating properties of elementary particles on the basis of Quantum Chromodynamical (QCD) theoretical formulations. This machine will be based on VLSI processing elements connected to parallel data streams coming from a very large RAM. It will actually calculate inverses of very large sparser matrices, and should apply to a wide class of different problems, in particular to the numerical solution of partial differential equations.

This project is carried on in collaboration with R. Giles (MIT Center for Theoretical Physics), R. Brower (visitor from University of Santa Cruz Department of Physics), and R. Suaya (Fairchild Corporation). The particular input of our group in this collaboration consists in applying the experience we have gained with the

construction of CAM in special-purpose computer architecture, in particular in the problems concerning memory management and interfacing the special machine with a host general-purpose computer.

7. A WORKSHOP ON PHYSICS AND COMPUTATION

In order to promote a rapid communication of recent results and an exchange of information among leading physicists and computer scientists, we organized the Second International Workshop on "Physics and Computational Processes," that was held in January 1984.

References

1. Brower, R., Giles, R. and Vichniac, G. "Cellular Automata and the Parallel Computations of Ising Spins," MIT Laboratory for Computer Science, Cambridge, MA, to appear.

2. Fredkin, E. "Digital Information Mechanics," MIT Laboratory for Computer Science, Cambridge, MA, to appear.

3. Margolus, N. "Physics-like Models of Computation," *Physica* 10D (1984) 81-95.

4. Margolus, N. "Quantum Computation," MIT Laboratory for Computer Science, Cambridge, MA, to appear.

5. Vichniac, G.Y. "Simulating Physics with Cellular Automata," *Physica* 10D (1984) 96-116.

Publications

1. Farmer, D. and Toffoli, T. *Proceedings of the Cellular Automata Workshop*, S. Wolfram (ed.), North Holland Publishing, 1984. Also in *Physica D* 10D (1984) 1 and 2.

2. Ritzenberg, A.L. and Vichniac, G.Y. "Exact Results in a Periodically Driven Ising Model," MIT Laboratory for Computer Science, Cambridge, MA, to appear.

3. Toffoli, T. "CAM: A High Performance Cellular-Automaton Machine," *Physica* 10D (1984) 194-205.

4. Toffoli, T. "Cellular Automata as an Alternative to (rather than an Approximation of) Differential Equations in Modeling Physics," *Physica* 10D (1984) 117-127.

5. Toffoli, T. "A Comment on 'Dissipation and Computation'," to appear in *Phys. Rev. Letter*, 1984.

6. Vichniac, G.Y. "Instability in Discrete Algorithms and Exact Reversibility," *SIAM Journal Alg. Disc. Methods*, 5, 4 (December 1984), to appear.

Thesis in Progress

1. Margolus, N. "Physics and Computation," Ph.D dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected 1985.

Talks

1. Toffoli, T. "CAM: A High-Performance Cellular Automaton Machine," IBM T.J. Watson Research Center, Yorktown Heights, NY, March 7, 1984.

2. Toffoli, T. "Dedicated Hardwares for the Physical Problems: Ising Spins in the Microcanonical Ensemble, and QCD," Brookhaven National Laboratory, Brookhaven, NY, May 2, 1984.

3. Vichniac, G.Y. "Simulating Physics with Cellular Automata,"
   Northeastern University, Boston, MA, October 1983
   Schlumberger-Doll Research, NJ, November 1983
   MIT Chemical Physics Colloquium, Cambridge, November 1983
   Nuclear Theory Seminar, University of Maryland, February 1984
   IBM T.J. Watson Research Center, Yorktown Heights, NY, March 1984
   Joint Theoretical Physics Seminar, Cambridge, MA, March 1984
   Argonne National Laboratory, May 1984

4. Vichniac, G.Y. "Faster than 1000 Vaxes," Physics Department, University of Chicago, Chicago, IL, May 16, 1984.

5. Vichniac, G.Y. "Demonstration of the CAM Machine,"
   Rutgers University Statistical Mechanics Meeting, December 1983.
   Disorderly Growth and Scaling, Exxon Meeting, Princeton, NJ, August 1983.

MESSAGE PASSING SEMANTICS

Academic Staff

C.E. Hewitt, Group Leader

Research Staff

Gerald Barber

Support Staff

C. Smith

1. OPEN SYSTEMS

Through this report, we describe some problems and opportunities associated with describing and taking action in the kind of "open systems" we foresee must and will be increasingly recognized as a central line of computer system development. Computer applications will be based on communication between systems which will have been developed separately and independently. Some of the reasons for independent development are the following: competition, different goals and responsibilities, economics, and geographical distribution. We must deal with all the problems that arise from this *conceptual* disparity of systems which have been independently developed. Systems will be open-ended and incremental -- undergoing continual evolution. There are no global objects. The only thing that all the various systems hold in common is the ability to communicate with each other. In this paper we study description and actions in Open Systems from the viewpoint of Message Passing Semantics, a research program to explore issues in the semantics of communication in parallel systems such as: negotiation, transaction management, problem solving, change, and self-knowledge.

The kind of systems we envisage are be open-ended and incremental -- undergoing continual evolution. In an open system it becomes very *difficult* to determine what objects exist at any point in time. For example a query might never finish looking for possible answers. If a system is asked to find all the current mail address of people who have graduated from MIT since 1930, it might have a hard time answering. It can give all the telephone numbers it has found so far, but there is no guarantee that another one can't be found by more diligent search. These examples illustrate how the "closed world assumption" is intrinsically contrary to the nature of Open Systems. We understand the "closed world assumption" to be that the information about the world being modeled is complete in the sense that *all and only* the relationships that can possibly hold among objects are those implied by the given information at hand [33]. Systems based on the "closed world assumption" typically assume that they can find all the instances of a concept that exist by searching their local storage. In contrast we desire that systems be accountable for having evidence for their beliefs and be explicitly aware of the limits of their knowledge. At first glance it might seem that the closed world assumption, almost universal in the A.I. and database literature, is smart because it provides a ready default answer for any query. Unfortunately the default answers provided become less realistic as the Open System increases in size.

2. DESCRIPTIONS OF BEHAVIOR

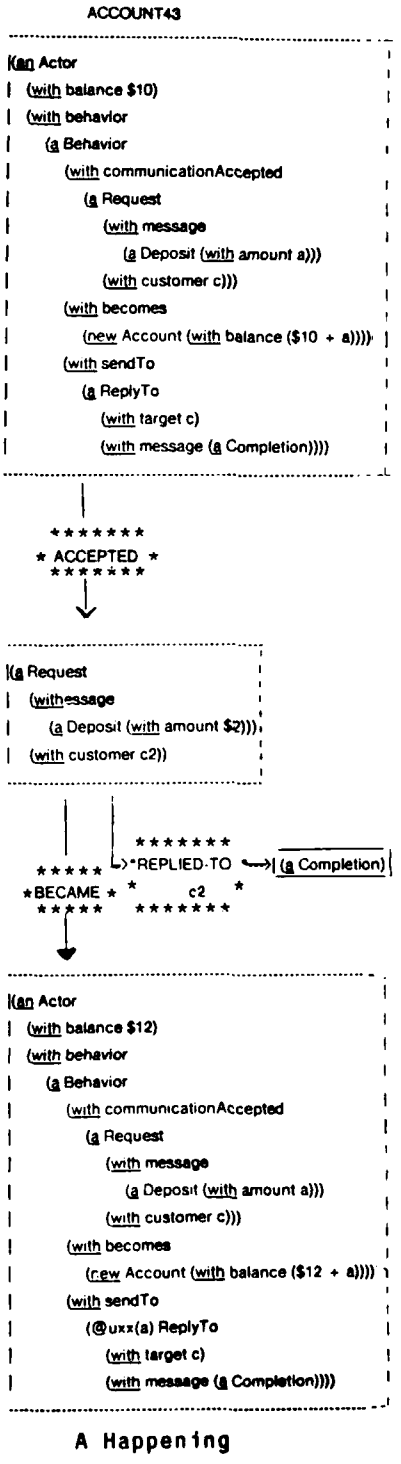
To build a conceptual modeling system for Open Systems, we need to develop a coherent understanding of the semantics of concurrent message passing systems. Message Passing Semantics is a research program to explore issues in the

semantics of communication in parallel systems. It builds on Actor Theory as a foundation for the conceptual modeling of Open Systems. This involves important aspects of parallelism and serialization beyond the sequential coroutine message passing developed in systems like Simula and SmallTalk.

An *actor system* is composed of abstract objects called *actors*. *Actors are defined by their behavior when they accept communications.* When a communication is accepted an actor can perform the following kinds of actions concurrently: make simple decisions (such as whether some actor it received in the communication is the same as one of its acquaintances), *create* new actors, *transmit* more communications to its own acquaintances as well as the acquaintances of the communication accepted, and change its behavior for the next message accepted (i.e., change its local state) subject to the constraints of certain laws [17].

3. AN EXAMPLE

In this section we describe the behavior of a simple actor which is a shared checking account which we will call **ACCOUNT43**. One kind of description might be a *partial* description of what happened when the actor **ACCOUNT43** with a balance of $10 accepted a request to make a deposit with amount $2 for customer **c2**, and as a result created the actor $12, sent a **Completion** report to **c2**, and became an account with balance $12:


ACCOUNT43
(an Actor
(with balance $10)
(with behavior
(a Behavior
(with communicationAccepted
(a Request
(with message
(a Deposit (with amount a)))
(with customer c)))
(with becomes
(new Account (with balance ($10 + a))))
(with sendTo
(a ReplyTo
(with target c)
(with message (a Completion))))
ACCEPTED
(a Request
(withessage
(a Deposit (with amount $2)))
(with customer c2))
BECAME
REPLIED-TO
(a Completion)
c2
(an Actor
(with balance $12)
(with behavior
(a Behavior
(with communicationAccepted
(a Request
(with message
(a Deposit (with amount a)))
(with customer c)))
(with becomes
(new Account (with balance ($12 + a))))
(with sendTo
(@uxx(a) ReplyTo
(with target c)
(with message (a Completion))))

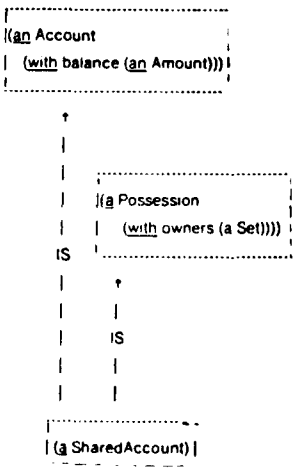
A Happening

Complete descriptions of the history of what happened can be constructed by keeping careful records of ongoing computations. The Time Warp System [21] is a particularly elegant way to collect and keep the records in a useful format. Such records can be used by an actor in an Open System to keep track of the effects of its actions. However these records do not in general make the effects of a given action deterministic because of the effects of other actors in the Open System. *Taking the same action in the future in what appears to a single actor as being identically the same circumstances as when the action was previously taken can have different effects in an Open System.*

The modeling of shared resources is fundamental to Open Systems. Actor systems aim to provide a clean way to implement *effects* (not "side-effects" a term that has been used as a kind of curse by proponents of purely applicative programming in the lambda calculus). By an *effect* we mean a local state change in a shared actor which causes a change in behavior that is visible to other actors. For example sending a deposit to an account shared by multiple users should have the effect of increasing the balance in the account.

ACCOUNT43 is an example of the kind of shared resource which is important in the conceptual modeling of Open Systems. Such shared resources should be suitable for use by a growing collection of users. To a given user, a shared **SharedAccount** will exhibit indeterminacy in the balance depending on the deposits and withdrawals made by other users. The indeterminacy arises from the indeterminacy in the arrival order of messages at the shared account.

A **SharedAccount** inherits attributes and behavior from *both* the description of an **Account** *and* the description of a **Possession**.



Multiple Inheritance

Dealing with the issues raised by the possibility of being a specialization of more than one description has become known as the "Multiple Inheritance Problem". A number of approaches have been developed in the last few years including the following: [40][11][8][[6] and [7]. Our approach differs in that it builds on the theory of an underlying description system [1] and in the fact that it is designed for a parallel message passing environment in contrast to the sequential coroutine object-oriented programming languages derived from Simula. Traditional properties of transactions (e.g., the all or nothing property) can be implemented by actors following the appropriate message protocols. These actors are called transaction managers. **SharedAccount** actors could be implemented as specializations of transaction managers and thereby acquire the proper message protocol.

If an actor can change its local state, it is called a *serialized* actor. A serialized actor accepts only one message at a time for processing; it will not accept another message for processing until it has dealt with the one which it is processing in at least a preliminary fashion. A communication received by a serialized actor is processed in order of arrival

4. MESSAGE PASSING SEMANTICS

Message Passing Semantics takes a different perspective on the meaning of a sentence from that of truth-theoretic semantics. In truth-theoretic semantics, the meaning of a sentence is determined by the models which make it true. For example the conjunction of two sentences is true exactly when both of its conjuncts are true. *In contrast Message Passing Semantics takes the meaning of a message to be the effect it has on the subsequent behavior of the system.* In other words the meaning of a message is determined by how it affects the recipients. Each partial meaning of a message is constructed by a recipient in terms of how it is processed [32]. The meaning of a message is open ended and unfolds indefinitely far into the future as other recipients process the message.

At a deep level, understanding always involves categorization, which is a function of interactional (rather than inherent) properties and the perspective of individual viewpoints. Message Passing Semantics differs radically from truth-theoretic semantics which assumes that it is possible to give an account of truth in itself, free of interactional issues, and that the theory of meaning will be based on such a theory of truth [24].

5. LIMITATIONS OF DESCRIPTIONS

One of the most challenging problems in the conceptual modeling of Open Systems is sorting out the relationship between *doing* and *describing*.

The distinction between *doing* and *describing* is different from the usual distinction made in the Artificial Intelligence literature [29][13] between the *epistemological adequacy* of a system (its *accuracy* with respect to truth-theoretic semantics [37] and the *heuristic adequacy* (the *efficiency* of its inferential procedures in proving theorems). Thus the distinction between epistemological adequacy and heuristic adequacy is founded on the basis of truth-theoretic semantics. Our simple example can help clarify the distinction. An epistemologically adequate theory of financial accounts gives an accurate description of the rules that govern them. An heuristically adequate system can derive theorems in the theory of financial accounts fast enough to answer queries. Both kinds of adequacy are concerned with the *description* of financial accounts: the former with its accuracy; the later with the efficiency with which the description can be used to answer questions.

Neither kind of adequacy actually *accomplishes* creating a shared account with a $10 in it so a growing set of geographically dispersed users can make deposits and withdrawals. We could *describe* a certain kind of account with an axiom such as the following:

```
((d > 0) implies
   (replies-after-acceptance
        (a SharedAccount (with balance b))
        (a Deposit (with amount d))
        (a CompletionReport
            (with ResultingBalance (b + d)))))
```

where the variables **b** and **d** are universally quantified and the predicate **replies-after-acceptance** takes three arguments which are a description of the local state of an actor when it accepts a request message, a description of the request accepted, and a description of the reply to the request, respectively. Now it should be clear that hypothesizing that **ACCOUNT43** satisfies the following description:

```
(a SharedAccount
    (with InitialBalance $20)
    (with PlaceBankOfLondon)
    (with Time "Midnight December 31 1987"))
```

does not by itself actually create such an account. Indeed the above assertion is ambiguous between the following interpretations:

- *Hypothesis Interpretation*: We have just discovered that **ACCOUNT43** is already such an account and want to explicitly record this in the data base.

- *Goal Interpretation*: We want to declare our goal of having **ACCOUNT43** be such an account and we will devote some effort to establishing the goal.

To actually create the account requires actually providing the money for the initial deposit in the account. Making assertions by itself will not suffice. Similarly asserting the ACCOUNT43 is not a **SharedAccount** on the other side of the country does not in itself destroy the account.

A common bug in attempting to model computation in open systems is to assume that an agent sending messages can determine the order in which they will be seen by the recipient. It is important to realize that the above assumption is contrary to the nature of Open Systems. Implementing shared resources inherently involves being open to outside communications, even those put forth by parties which joined the Open System after the request being considered was received.

6. TAKING ACTION

Knowing that something is true and taking action to make it come true are two different things. Both are important and they should not be confused. In this section we discuss how actors can take action to supplement the previous sections discussion of how they can manipulate descriptions.

Actions such as creating a new shared account with balance $12 and owner Ken can be performed by evaluating an expression such as the one below in the programming language Act2:

```
(new SharedAccount
     (with (Balance of account) $12)
     (with (Owners of Possession) (Ken)))
```

Below we present *part* of the implementation of **Sharedaccount**. This implementation is written out in "parenthesized English" [22][14][16][26][38] which is gradually developing into a technical language. The underlined words have special technical meanings.

```
(Define
(new SharedAccount (with (balance of Account) b)
        (with (owners of Possession) s))
to have the following implementation:
(create a new actor with the behavior that
 after acceptance, select one of the following handlers
 for the communication accepted:
  (If it is (query (balance of Account)) request,
    (reply b))
  (If it is (a Withdrawal (with Amount w)) request,
    (select one of the following cases for w:
     (If it is (less than or equal to b),
      (reply (a CompletionReport)) as well as
      (become (new SharedAccount
              (with (balance of Account) (b - w)))))
     (If it is (greater than b),
      (complain (an Overdraft Complaint)))))
  (If it is (a Deposit (with Amount d)) request,
    (reply (a CompletionReport
             (with ResultingBalance (b + d))))
    (become (new SharedAccount
             (with (balance of Account) (b + d)))))
  (If it is (query (owners of Possession)) request,
    (reply s))
  ...
  .)
```

At this point we would like to take note of several unusual aspects of the above implementation. By default, commands in the body of a procedure are executed concurrently. For example, in the communication handler for **Withdrawal** requests above, the following two commands are executed concurrently:

```
(reply(aCompletionReport))
(become(newSharedAccount(with(balance ofAccount)(b - w))))
```

The principle of maximizing concurrency is fundamental to the design of actor programming languages. It accounts for many of the differences with conventional languages based on communicating sequential processes. Another example of maximizing concurrency is that the processing of messages by a serialized actor can be pipelined. Serialized actors can be pipelined since processing on a subsequent message can commence once the become command has been executed since it designates the actor which will process the subsequent message. For example after accepting a **Withdrawal** message then executing the following become command

```
(become(newSharedAccount(with(balanceofAccount)(b - w))))
```

action of the call has aborted (this would happen, for example, if the top action ran at the crashed guardian), or that some ancestor of the handler call will contain an appropriate aid, i.e., an ancestor of the call, in its *aborts_list*.

For example, consider the case of the system aborting the call. Recall that in executing a handler call, the system actually creates two subactions, one, the *call action*, at the calling guardian and another, the *handler action*, at the called guardian. The call action is a child of the action making the call, and the handler action is a child of the call action. Two actions are needed so that the call can be aborted independently at the calling guardian even if the handler action committed at the called guardian. This detail was suppressed in the action tree shown in Figure 11-1. When the system aborts the call action at the calling guardian, it inserts the aid of this subaction in the *aborts_list* of its parent. Thus, the *aborts_list* of an ancestor of the handler action contains an appropriate aid.

2.3. Two-phase Commit

Distributed commitment in Argus is carried out by a standard two-phase commit protocol [7], [12]. In this protocol, the guardian of the committing top action acts as the *coordinator*. The other guardians, namely those in the *plist*, act as the *participants*.

The coordinator sends the aborts_list to each participant in the prepare message. When a participant receives a prepare message for some top action A, it compares every descendant of A in its *committed* with the actions listed in the aborts_list of the prepare message. If a descendant of A is not also a descendant of some action in the aborts_list, then it is known to have committed to A. In this case, its *olist* is used to cause all new versions created for it to be written to stable storage. If D is a descendant of some action in the aborts_list, its effects are undone: its *olist* is used to cause its locks to be released and its versions discarded. Thus the aborts_list is used at the participant to reconstruct enough of the action tree to determine what to do with every local descendant of the committing top action.

Actually, sending just the aborts_list in the prepare message is not *quite enough*, as the following example illustrates. If a guardian crashes after running some handler calls that are subactions of top action A, and then runs more handler calls that are subactions of A after it recovers, only the latter calls will be listed in *committed*. If a handler call that ran before the crash committed to the top, it versions should be written to stable storage. Since the versions were lost in the crash, the guardian should refuse to prepare. However, given the information discussed so far, there is no way that it could know this. For example, consider the action tree of Figure 11-1, and suppose that A.2.1 committed at G3, then G3 crashed, and after G3 recovered A.1.1 ran and committed there. The algorithm

olist — The list of local atomic objects used by this action and its local committed descendants

These lists are empty when an action starts. Whenever an action uses an atomic object, this information is added to its *olist*. The other data are modified as descendants of the action commit or abort. If a local subaction commits, its *olist*, *plist* and *aborts_list* are merged with those of its parent. Also its locks and versions are propagated to its parent. If the local subaction aborts, its locks and versions are discarded (using the information in the *olist*). Since the action is aborting, it is contributing no participants to its parent, and therefore its *plist* is discarded. Finally, the aborting subaction's aid is added to its parent's *aborts_list* if it may have committed descendants at other guardians. This will be true if either its *plist* or its *aborts_list* is non-empty, or if it is waiting on a handler call when it aborts.

When a handler call is made, a call message is constructed and sent to the handler's guardian. Later a reply message is sent from the handler's guardian to the caller's guardian. This reply message contains a *plist* and an *aborts_list*, which are merged with those of the caller. The *olist* is not sent back in the reply; information about used objects is kept locally at the guardian that contains the objects.

When the call message is received at the handler's guardian, a subaction is created for it, with its associated *plist*, *aborts_list* and *olist*, all empty. As the handler action runs, these data are modified as described above. Now let us consider what happens when the handler action completes. First, suppose it commits. In this case its aid and *olist* are stored in *committed* at its guardian. Its gid is added to its *plist* and then its *plist* and *aborts_list* are sent back to the calling guardian as part of the reply message. If the handler call aborts, its locks are released. The *plist* in the reply message is empty. If the aborting subaction made no remote calls that may have committed at other guardians (i.e., its *plist* and *aborts_list* are empty), the *aborts_list* in the reply message is empty; otherwise it contains the aid of the aborting handler action.

For example, when A.2 in Figure 11-1 commits, its *plist* with its gid added is {G2, G3, G5} and its *aborts_list* is empty; this information is sent to G in the reply message. When A.1 aborts, its *plist* is non-empty, so the reply message contains *aborts_list* {A.1}. When this information is merged at G we end up with the *plist* and *aborts_list* discussed earlier.

In the above discussion, we assumed that when the reply message arrived at the calling guardian, the caller was waiting for the reply. However, this is not necessarily so. The calling guardian may have crashed before the reply arrived, the calling action may have aborted because of the termination of a coenter, or the call may have been timed out by the system because of a network partition. In all these cases the reply message is discarded. Furthermore, we can be certain that either the top

@G1	No information
@G2	A.2 committed
@G3	A.1.1 committed A.2.1 committed
@G4	A.1.2 committed
@G5	A.2.2 committed
@G6	No information

Figure 11-2: Information Stored at Subactions' Guardians.

At the top action's guardian, we remember (in volatile storage) the parts of the tree that are not stored at the subactions' guardians. First, there is a data structure called the *plist* that lists the participants. Second, there is the *aborts_list*, which lists those subactions that aborted but that might have committed descendants at other guardians. For the action tree in Figure 11-1, we have

plist = {G2, G3, G5}
aborts_list = { A.1 }

Notice that the *aborts_list* need not contain aborted subactions that have no remote descendants (e.g., A.2.3), because this information is (effectively) stored at the aborted subaction's guardian. Notice also that the *aborts_list* need not contain two subactions where one is an ancestor of the other; in such a case only the older of the two subactions need be remembered.

2.2. Constructing the Action Tree

While an action is running, the implementation maintains the following volatile information for it at it's guardian:

plist	The list of guardians visited by committed descendants of this action, where all ancestors of the descendant up to this action have committed (i.e., the descendant committed to this action).
aborts_list	The list of aborted descendants of this action that may have committed descendant at other guardians

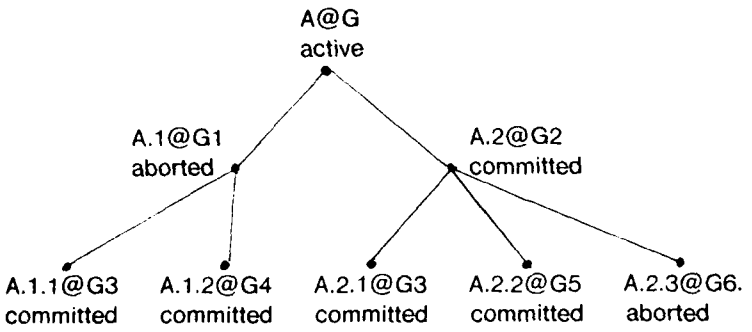


Figure 11-1: An Action Tree Just Before Committing of Top Action A.

tree were sent to all participants, then they could determine which of the subactions that ran locally should have volatile versions written to stable storage, and which should have their volatile versions discarded. For example, at G3 it would be known that A.2.1's volatile versions should be written to stable storage, but A.1.1's versions should be discarded. However, the tree may be large, so we have chosen a different approach. Rather than building the tree at the top action's guardian as the top action and its descendants run, we keep parts of the tree at the descendants' guardians. Some information must still be collected at the top action's guardian, but the amount of information is reduced.

Before discussing the stored information, it is necessary to say a few words about action identifiers (*aids*). The action identifier of a subaction, e.g., A.2.2, contains within it the guardian identifier (*gid*) of the guardian at which the subaction ran, e.g., G5, and also the aids of all the ancestors of the subaction, e.g., A.2 and A. Furthermore, given two aids, it is possible to tell whether one is an ancestor of the other. Thus some of the information in the action tree is stored in the aids.

Figure 8-2 shows the information kept at the guardians where descendants ran for the action tree shown in Figure 8-1. These guardians remember, in a local, volatile data structure called *committed*, those handler subactions that ran at the guardian and then committed. In addition, for each of these subactions, the guardian remembers all the atomic objects on which the subaction holds locks. This information is used to determine what needs to be written to stable storage during two-phase commit and to release locks. Handler subactions that ran at the guardian and then aborted are forgotten; these subactions hold no locks (the locks were released at the time of the abort) and no information need be written to stable storage for them.

subactions. Each individual action runs at a single guardian; we will refer to this guardian as the *action's guardian*. An action can affect other guardians by means of handler calls. A distributed computation starts as a top action at some guardian and spreads to other guardians by means of handler calls, which are performed as subactions of the calling action. A handler call subaction may make further handler calls to other guardians; it may also make use of the objects at its own guardian and thus acquire locks on them and also (if it modifies the objects) cause new volatile versions to be created. Since these versions are in volatile storage, they will be lost if their containing guardian crashes before the top action commits. Therefore, when a top action commits, a distributed commitment procedure must be carried out to guarantee that the new versions of objects modified by descendants of that action are copied to stable storage.

In this section, we describe how the distributed transaction system is implemented. We begin by describing a model of actions that can be used both to discuss how actions execute, and what happens when actions complete. We use this model to describe the information that is collected at guardians as the action and its subactions run, how this information is used during distributed commitment, and how locks are propagated from one guardian to another.

2.1. Action Trees

A top action and its descendants can be modeled by means of a tree structure called an *action tree*. The root of the tree is labeled by the top action; the interior nodes are labeled by descendant subactions. Only subactions appear below the root of the tree; a nested top action will be represented by its own tree. Each node of the tree contains information about the state of its action (active, committed or aborted) and the guardian at which the action is running or ran. Figure 11-1 shows a tree that might exist just before the top action, A, commits.

A subaction is said to have *committed to the top* if it committed, and so have all its ancestors up to, but not including, the top action. For example, A.2.1 committed to the top, but A.1.1 and A.1.2 did not. When a top action commits, it is necessary to communicate with the guardians of all actions that committed to the top. The guardians are called the *participants*. We need not communicate with guardians of actions that did not commit to the top (for example, guardian G4 in the figure), which is fortunate since such communication may be impossible. For example, the reason A.1 aborted may have been because a network partition made it impossible to receive the results from G4.

If a top action's tree were known at the top action's guardian when the top action is about to commit, the information in it could be used to control distributed commitment. The participants can be computed from the tree, and if, in addition, the

1. INTRODUCTION

This year, the Programming Methodology Group has continued to study the structure and execution of distributed programs. As before, our work has focused on the design and implementation of the Argus language and system, which is being developed to support the construction and execution of distributed programs. During the past year, we have completed the design of a preliminary version of Argus, and have published a reference manual describing this initial language [16].

The implementation of Argus has been a major effort this year. This implementation is being carried out on a number of Vaxes running Berkeley Unix 4.2 and connected by a local-area net. This year we succeeded in bringing up guardians in isolation: an individual guardian can now run, including the execution of top actions and subactions, but it is not yet possible to run a computation that takes place at many different guardians. In addition, as a first step in supporting distributed computations, we implemented a communications subsystem that runs on top of IP [20], and we implemented remote procedure calls on top of this substrate. Finally, as we implement Argus itself, we are implementing a system for debugging Argus programs so that as each new feature of Argus is executable, debugging support for programs using that feature is also available.

In addition to our work on Argus, we are conducting research on a number of topics in both programming methodology and distributed computing. Julie Lancaster [13] designed a naming structure for a program development system that supports version control. Our work in distributed computing includes orphan detection algorithms, data replication techniques and action debugging [4].

In the following sections we assume some familiarity with Argus; an overview can be found in [14]. Section 2 provides an overview of the Argus implementation, focusing on the way that atomic actions are implemented. Section 3 discusses our orphan detection algorithm, and then describes a method for reducing the cost of the algorithm. Section 4 includes some issues concerning atomic data types, and Section 5 describes a new replication method for constructing highly available distributed objects.

2. IMPLEMENTATION

The Argus implementation includes an operating system kernel that supports execution and scheduling of guardians and their processes, and also some form of message communication. In addition, it contains a distributed transaction system, and a recovery system.

A distributed computation in Argus consists of a top action and all of its

PROGRAMMING METHODOLOGY

Academic Staff

B. H. Liskov, Group Leader

Research Staff

P. R. Johnson
R. W. Scheifler

Graduate Students

S.-Y. Chiu
M. P. Herlihy
R. Ladin
J. N. Lancaster
G. T. Leavens
B. M. Oki
J. P. Restivo
E. F. Walker
W. E. Weihl
L. J. Yedwab

Undergraduate Students

N. A. Beardsley
M. W. Chan
A. Della Fera
C. A. Gosling

Support Staff

A. Rubin

Visitor

Y. Miyashita

36. Steele, G.L., Jr. and Sussman, G.J. "The Revised Report on SCHEME: A Dialect of LISP," MIT/AI/TM-452, MIT Artificial Intelligence Laboratory, Cambridge, MA, January 1978.

37. Tarski, A. "The Semantic Conception of Truth," *Philosophy and Phenomenological Research 4*, (1944), 341-375.

38. Theriault, D. "A Primer for the Act-1 Language," MIT/AI/TM-672, MIT Artificial Intelligence Laboratory, Cambridge, MA, April 1982.

39. Turing, A.M. "Computability and @G(1)-Definability," *Journal of Symbolic Logic*, 2, (1937), 153-163.

40. Weinreb, D. and Moon, D. "LISP Machine Manual," MIT Artificial Intelligence Laboratory, Cambridge, MA, March 1981.

23. Kahn, K.M. "Intermission - Actors in Prolog," Logic Programming, Academic Press, 1982.

24. Lakoff, G. and Johnson, M. Metaphors We Live By, The University of Chicago Press, Chicago, IL, 1980.

25. Landin, P. "A Correspondence Between ALGOL 60 and Church's Lambda Notation," *Communications of the ACM*, 8, 2, (February 1965).

26. Lieberman, H. "A Preview of Act-1", MIT/AI/TR-625, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1981.

27. Lieberman, H. "Thinking About Lots of Things at Once Without Getting Confused: Parallelism in Act-1," MIT/AI/TR-626, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1981.

28. Liskov, B., Snyder, A., Atkinson, R and Schaffert, C. "Abstraction Mechanism in CLU," *Communications of the ACM*, 20, 8, (August 1977).

29. McCarthy, J. and Hayes, P.J. "*Some Philosophical Problems from the* Standpoint of Artificial Intelligence," Machine Intelligence 4, Edinburgh University Press, Edinburgh, Scotland, 1969, pp. 463-502.

30. McCarthy, J. LISP 1.5 Programmer's Manual, The MIT Press, Cambridge, MA, 1962.

31. Milne, R. and Strachey, C. A Theory of Programming Languages, John Wiley & Sons, New York, NY, 1976.

32. Reddy, M. "The Conduit Metaphor," Metaphor and Thought, Ortony, A. (ed.), Cambridge University Press, 1979.

33. Reiter, R. "Towards a Logical Reconstruction of Relational Database Theory," *Perspectives on Conceptual Modeling*, Springer-Verlag, Brodie, M.L., Mylopoulos, J.L, and Schmidt, J.W. (eds.). 1982.

34. Shapiro, E. "A Subset of Concurrent Prolog and Its Interpreter," ICOT/TR-003, Tokyo, Japan, January 1983.

35. Shaw, M., Wulf, W.A. and London, R.L. "Abstraction and Verification in Alphard: Defining and Specifying Iteration and Generators," *Communications of the ACM*, 20, 8, (August 1977).

12. Friedman, D.P. and Wise, D.S. "The Impact of Applicative Programming on Multiprocessing," *Proceedings of the International Conference on Parallel Processing*, 1976, 263-272.

13. Hayes, P.J. "In Defense of Logic," *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, MA, 1977, 559-565.

14. Hayes-Roth, F., Gorlin, D., Rosenschein, S., Sowizral, H. and Waterman, D. "Rationale and Motivation for ROSIE," Rand Corporation TR-N-1648-ARPA, Santa Monica, CA, November 1981.

15. Hewitt, C., Attardi, G. and Lieberman, H. "Specifying and Proving Properties of Guardians for Distributed Systems," *Proceedings of the Conference on Semantics of Concurrent Computation*, Evian, France, July 1979.

16. Hewitt, C., Attardi, G. and Simi, M. "Knowledge Embedding with a Description System," *Proceedings of the First National Annual Conference on Artificial Intelligence*, August 1980.

17. Hewitt, C. and Baker, H. "Laws for Communicating Parallel Processes," *1977 IFIP Congress Proceedings*, 1977.

18. Hewitt, C.E. "The Apiary Network Architecture for Knowledgeable Systems," *Conference Record of the 1980 LISP Conference*, Stanford, CA, August 1980.

19. Ichbiah, J.D. Reference Manual for the Ada Programming Language, United States Department of Defense, Arlington, VA, November 1980.

20. Ingalls, D. "The Small TAlk-76 Programming System, Design and Implementation," *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, Tucson, AZ, January 1978.

21. Jefferson, D. and Sowizral, H. "Fast Concurrent Simulation Using the Time Warp Mechanism, Part 1: Local Control," Rand Corporation TR-N-1906-AF, Santa Monica, CA, December 1982.

22. Kahn, K.M. "Creation of Computer Animation from Story Descriptions," MIT, Cambridge, MA, 1979.

References

1. Attardi, G. and Simi. M. "Semantics of Inheritance and Attributions in the Description System Omega," *Proceedings of IJCAI 81*, Vancouver, B.C., Canada, August 1981.

2. Backus, J. "Can Programming be Liberated from the von Neumann Style? A Functional Style and its Algebra of Programs," *Communications of the ACM*, 21, 8, (August 1978), 613-641.

3. Baker, H. and Hewitt, C. "The Incremental Garbage Collection of Processes," *Conference Record of the Conference on AI and Programming Languages*, Rochester, NY, August 1977.

4. Barber, G.R., deJong, S.P. and Hewitt, C. "Semantic Support for Work in Organizations," *Proceedings of IFIP-83*, September 1983.

5. Birtwistle, G.M., Dahl, O-J., Myhrhaug, B. and Nygaard, K. Simula Begin, Van Nostrand Reinhold, New York, NY, 1973.

6. Bobrow, D.G. and Stefik, M.J. "Loops: An Object Oriented Programming System for Interlisp," Xerox, Palo Alto Research Center, Palo Alto, CA, 1982.

7. Borgida, A., Mylopoulos, J.L. and Wong, H.K.T. "Generalization as a Basis for Software Specification," Perspectives on Conceptual Modeling, Brodie, M.L., Mylopoulos, J.L. and Schmidt, J.W (eds.), Springer-Verlag, 1982.

8. Borning, A.H. and Ingalls, D.H. ""Multiple Inheritance in Smalltalk-80," *Proceedings of the National Conference on Artificial Intelligence*, August 1982.

9. Church, A. "The Calculi of Lambda-Conversion," Annals of Mathematics Studies Number 6, Princeton University Press, Princeton, NJ, 1941.

10. Clinger, W.D. "Foundations of Actor Semantics," MIT/AI/TR-633, MIT Artificial Intelligence Laboratory, Cambridge, MA, May 1981.

11. Curry, G., Baer, L., Lipkie, D. and Lee, B. "Traits: An Approach to Multiple-Inheritance Subclassing," *Conference on Office Information Systems*, June 1982.

of the above possibilities inherently involves taking any action. The capability to take action as well as to describe the world is very important. The relationships between description and action are quite subtle.

We claim that actors (unlike lambda expressions, logical implications, etc.) are the universal objects of concurrent systems and that they can serve as a efficient interface between the hardware and software. Actors provide an absolute conceptual interface between the software and hardware of parallel computer systems. The function of the hardware is to efficiently implement the primitive actors and the ability to communicate in parallel. Software systems in turn can be implemented in terms of actors completely independently of the hardware configuration. The actor concept itself is defined mathematically and is thus logically independent of all programming languages and hardware architectures. A system consisting of multiple processors -- called the APIARY -- is being developed to use the inherent parallelism of actor systems to increase the efficiency of computation [18].

Message Passing Semantics deals coherently with both doing and describing whereas truth-theoretic semantics only addresses some of the issues of describing. We claim that it is impossible to implement shared resources for Open Systems in description systems such as first order logic and the lambda calculus because they lack the necessary communication capabilities.

Description languages based on first order logic and/or the lambda calculus have been designed to express properties but are incapable of taking action. On the other hand procedural languages (such as current dialects of Lisp and Ada) have been designed to efficiently take action but they suffer from a lack of descriptive capabilities. We need good ways to integrate the roles of descriptions and actions in our systems. Some of the ideas in this report have been applied to the analysis of the relationship between the roles of descriptions and actions in organizational work [4].

Systems independent agents can incrementally spawn ongoing computations so that the environment of a computation is not fixed when a computation is begun.

7. RELATED WORK

The object-oriented programming languages (e.g., [5][28][35] and [19]) are built out of objects (sometimes called "data abstractions") which are completely separate from the procedures in the language. Similarly the lambda calculus programming languages (e.g., [30][25][12][2] and [36]) are built on functions and data structures (viz. lists, arrays, etc.) which are separate. SmallTalk [20] is somewhat a special case since it simplified Simula by leaving out the procedures entirely, i.e. it has only classes. The Simula-like languages provide effective support for coroutines but not for concurrency. In contrast the Actor Model is designed to aid in conceptual modeling of shared objects in a highly parallel open systems. Actors serve to provide a unified conceptual basis for functions, data structures, classes, suspensions, futures, procedure invocations, exception handlers, objects, procedures, processes, etc. in all of the above programming languages [3][27][15]. For example sending a request communication generalizes the traditional procedure invocation mechanism which requires that control return to the point of invocation. A request communication contains the mail address of a customer to which the response to the request should be sent as well as the message specifying the task to be performed. In this way, exception handlers [28][19] and co-routines [5] are conveniently unified with other more general control structures. The Actor Model unifies the conceptual basis of the lambda calculus and the object-oriented schools of programming languages -- being mathematically defined, it is independent of all programming languages.

The results in this paper are intended to contrast with and further develop the ideas in [23] and [34]. Both of the above systems are pragmatically useful and interesting experiments in their own right. Unfortunately, neither as yet has a well developed mathematical semantics which would make it possible to directly analyze them as we have done for the lambda calculus and first order logic. *To the extent that Intermission and Concurrent Prolog are based on first order logic and the lambda calculus, they inherit limitations discussed in this paper.*

8. CONCLUSION

The fundamental thesis of this report is that knowing that something is the case and taking action to make it the case are two different things which should not be confused. Asserting a proposition **P** can mean that we have established that **P** is the case and want to explicitly record this in the data base. Or it can mean that we want to declare that **P** is our goal and we will devote some effort to establishing it. Neither

a **SharedAccount** can then *concurrently* accept an **Account Balance** query and reply with the Completion Report for the **Withdrawal** request.

An important special case occurs when an actor can never change its local state. Such an actor is called *unserialized* and is treated specially so that conceptually it is able to process arbitrarily many messages at the same time. Actors such as the square root function and the text of Lincoln's Gettysburg Address are unserialized.

Another unusual aspect is that there are *no* assignment commands. Effects are implemented by an actor changing its *own local state* using a become command [15]. We model change by actors which change their own local state. Our conceptual model of change contrasts with the usual computer science notion in which change is modeled by updating the state components of a universal global state [31]. The absence of the existence of a well defined global state is a fundamental difference between the actor model and classical sequential models of computation [39][9]. *Actor systems can perform nondeterministic computations for which there is no equivalent nondeterministic Turing Machine* [10]. The nonequivalence points up the limitations of attempting to model parallel systems as nondeterministic sequential machines [31]. This is not to say that actor systems can implement functions which are not recursively computable (like solving the Halting Problem). Instead the point is that recursive functions do not provide an adequate model of parallelism, i.e., an Open System cannot be adequately modeled as a recursive function which maps global states to global states because at any given point in time an Open System does not in general have a well defined global state. Will Clinger has developed an elegant mathematical theory (called Actor Theory) which accounts for capabilities of actor systems which go beyond those of nondeterministic Turing Machines. *We claim that nondeterministic Turing machines are an unsatisfactory model of the computational capabilities of large-scale open systems.*

Concurrency in Open Systems stems from the parallel operation of an incrementally growing number of multiple, independent, communicating agents. Sites can join an Open system in the course of its normal operation -- sometimes even affecting the results of computations initiated *before* they joined. *Actor Theory has been designed to accurately model the computational properties of Open Systems.* It is a consequence of the Actor Model that purely functional programming languages based on the lambda calculus *cannot* implement shared accounts in Open Systems. The technique promoted by Strachey and Milne [31] for simulating some kinds of parallelism in the lambda calculus using continuations does not apply to Open Systems. The lambda calculus simulation is sequential whereas Open Systems are inherently parallel. Concurrency in the lambda calculus stems from the concurrent reduction/evaluation of various parts of a *single* lambda expression with an environment which is fixed when the lambda expression is created. In Open

discussed above will (erroneously) prepare in this case. Such a situation is not a problem for us because of orphan detection, which is described in Section 3.

2.4. Lock Propagation

Our lock propagation rule specifies that when a subaction commits, its locks and versions are propagated to its parent; when it aborts, its locks and versions are discarded. Propagation of an object's locks and versions is always performed at the object's guardian. In addition, we carry out this propagation immediately only for local actions. For example, when a handler action commits, we do not propagate its locks and versions to its parent, nor do we communicate with the guardians of its non-local descendants to cause the appropriate propagation of locks for their local objects. In this way we avoid the delays that such communication would cause; only when top actions commit is this communication necessary.

Since communication only happens when top actions commit, certain problems can arise. For example, consider the action tree in Figure 11-1. How does guardian G4 discover that the locks held by A.1.2 should be released? A related problem may occur at G3. Suppose that A.1.1 and A.2.1 modify the same object, X. Suppose further that A.1 and A.2 are actually running concurrently, so that it is possible that either A.1.1 or A.2.1 may get to G3 first. If A.1.1 modified X first, then before A.2.1 can use X, we must discard A.1.1's lock and version of X. To do this, we must learn at G3 about the abort of A.1. Alternatively, if A.2.1 modified X first, then before A.1.1 can use it we must learn about the commit of A.2, and propagate the lock and version of X to A. Only then can A.1.1 use X.

We solve problems like these by means of *lock propagation queries*. As mentioned earlier, when a handler call commits, we continue to keep all its used objects locked on its behalf. If later some other action wants to use the locked object, the object's guardian will send a query to determine the fate of the handler action that holds the lock. Such a query will be sent to a guardian at which an ancestor of the handler action ran. Recall that the aid of a subaction can be used to determine the aids of the ancestors of the subaction and thus the guardians of the ancestors.

In the following, we discuss two actions, H and R. H is a committed handler action that holds a lock on some object. R is an active action that wants to acquire a lock on that object. H and R's guardian will send a query to some other guardian. That other guardian will respond with a *query response* telling what it knows, and H and R's guardian will then act on that information.

There are two situations to consider, depending on whether H is related to R or not. If H and R are not related, then H's lock can be broken only if H did not commit to the top or H's top action, T, aborted. A good place to send the query is to T's guardian, GT. There are the following possibilities at GT:

1) T is not known at GT, and the query response so indicates. There are two possibilities here: T aborted, or T committed but two-phase commit is finished. In the latter case, if H committed to the top, its guardian will have already discarded its locks and made its versions the current versions, and the query response will be ignored. If H still holds locks when the query response is received, its locks will be released and its versions discarded.

2) T is in the second phase of two phase commit. Again, there are two possibilities: either H's guardian is not a participant, or it is. If H's guardian is a participant, it will either be in phase 2 or finished with two phase commit for T when the query response arrives, and will ignore the response; otherwise, it will discard H's locks and versions.

3) If T is active or in phase 1, but H is a descendant of some action in T's *aborts_list*, then the query response can indicate that some ancestor of H aborted. In this case H's locks and versions can be discarded.

4) If T is in phase 1, but H is not a descendant of some action in the *aborts_list*, then we can discard the query, since H's guardian is about to receive appropriate information anyway as part of two-phase commit.

5) If T is active and H is not a descendant of some action in the *aborts_list*, then the query cannot be resolved yet. In this case, H's guardian cannot release H's locks; it can query to GT again later, or it could query to guardians of other ancestors of H to try to discover if some ancestor of H aborted. This additional querying is useful only if an ancestor of H is active at some other guardian; such information can be included in the query response.

For example, suppose that A.1.1 of Figure 11-1 is the holder, and the requester, R, is a non-relative. The query would be sent to G. If G knows nothing about A, or is in phase two for A, or A is active or in phase 1 but A.1 appears in the *aborts_list*, then G3 can be told to release A.1.1's lock. (In fact, all of A.1.1's locks will be released and its versions discarded at this point.) The only other possibility for this action tree is that A.1 is still active; in this case G3 might try a query to G1.

The second situation is when H and R are related. In this case we are interested in an ancestor of H and R called the *least common ancestor* (the LCA). The LCA is the action that is an ancestor of both H and R, and that is younger than all other ancestors of H and R. For example, for the action tree of Figure 11-1, A.2 is the LCA of A.2.1 and A.2.2, while A is the LCA of A.1.2 and A.2.2. To satisfy R's request for the object, we must learn whether or not H committed to the LCA. If H did commit to

the LCA, its locks and versions can be propagated to the LCA and then R can obtain its needed lock. Notice that in this case R will observe any modifications made by H. The other cases are described below.

So, when H and R are related, a query is sent to the LCA's guardian. There are the following possibilities:

1) If the LCA is active and H is a descendant of some action in its *aborts_list*, then the query response can indicate that some ancestor of H aborted. In this case, H's locks and versions can be discarded, and then R can obtain its needed lock.

2) If the LCA is active, H is not a descendant of an action in the *aborts_list*, and the handler call that gave rise to H has completed, then the query response can indicate that H did commit to the LCA. In this case, H's locks and versions can be propagated to the LCA and then R can acquire the needed lock.

3) If the LCA is active and the handler call that gave rise to H is still active, then the query cannot be resolved at the LCA's guardian. In this case H's guardian may query to other guardians where ancestors of H are running.

4) Otherwise, the LCA is no longer active at its guardian. In this case, the fate of H is not known at the LCA's guardian, but R is an orphan. We will discuss orphans in the next section. When H's guardian receives this response, it will leave H's locks and versions intact, but destroy R as discussed below.

It is worth noting that queries are the glue that holds the system together. For example, when a handler action aborts, we may choose to notify guardians where descendants committed about the abort, but this is strictly an optimization to avoid queries later. We do not need to communicate before the reply message for the aborting handler is sent to its caller, so we incur no delay in the processing of actions. Instead we can communicate when it is convenient, for example when the handler's guardian is not doing anything. Furthermore, we need not try very hard to communicate, since if a message is lost, the information can always be obtained by a query.

3. ORPHANS

An *orphan* is any active action whose results are no longer wanted (see also [19]). Orphans arise from two different sources: explicit aborts and crashes. Let us consider the case of explicit aborts first. One way aborts happen is when the system determines that a handler call cannot be completed right now. As mentioned above, a handler call causes two subactions to be created, and the call action will commit only after it has received the handler's reply and extracted the results from that reply message. This can occur only if the handler action has actually finished and either committed or aborted. On the other hand, the call action can abort without receiving a reply from the handler, and in this case the handler action (or some descendant of it) may still be running as an orphan.

Aborts also happen when an arm of a coenter exits the coenter and causes other arms to abort. The local subactions that correspond to these other arms will be terminated when this occurs, but if any of those arms is waiting for a handler call to complete, we abort the call action immediately and may leave an orphan at some other guardian.

The second way orphans arise is due to crashes. For example, the guardian making a call may crash, leaving the handler action as an orphan. Another possibility is the following: Suppose subaction S running at guardian GS made a call to guardian GC. Suppose this call committed, but subsequently GC crashed. In this case S must ultimately abort, because it depends on information at GC that has now been lost. Therefore, S is an orphan.

Since S is an orphan, it cannot commit and therefore none of its changes will become visible to other actions. Nevertheless, orphans are bad for two reasons. Since their results are not wanted, they are wasting resources. For example, some other action may be delayed because an orphan holds a needed lock. In addition, because they depend on locks that have been broken (for example, S depends on locks acquired by the handler call to GC), there is a danger that they may observe inconsistent data. Such inconsistent data can cause a program that behaves reasonably when the data is consistent to behave strangely; for example, a program that would ordinarily terminate may loop forever. In addition, if the program is interacting with a user, it may display inconsistent data to the user.

To illustrate this problem of inconsistent data, consider the following example. Suppose guardian Y replicates the data stored at another guardian X. The consistency constraint between each object x at X and its replica y at Y is that $x = y$. Now consider the scenario shown in Figure 11-3: First suppose that top action S makes a call, S.1, to guardian X. S.1 reads x, thus obtaining a read lock on x, displays the value of x to a user at a console, and then commits. Next X crashes and subsequently recovers; however, S.1's lock on x has been lost. Then another top

action T makes a call to X; this call, T.1, changes the value of x to $x + 1$ and commits. Next T makes a call T.2 to Y, which changes y to $y + 1$ (thus preserving the invariant) and commits. Finally T commits, so the locks on x and y held by its descendants are released, and x and y take on their new values. Now S makes a call S.2 to Y, which reads the new value of y and displays this value to the user at the console. The new value of y is different from the value of x displayed previously, so the user has seen inconsistent data.

```
S @ G_S:                    T @ G_T:

  S.1 @ X:
    display x to user
    commit

X crashes and recovers - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                              T.1 @ X:
                                x := x + 1
                                commit

                              T.2 @ Y:
                                y := y + 1
                                commit

                              T commits

  S.2 @ Y:
    display y to a user
```

Figure 11-3: An Orphan Scenario.

In Argus, we guarantee to eliminate orphans quickly enough that an orphan can never observe data in a state that could not be observed by a non-orphan. As a result of this guarantee, it is not necessary to worry about orphans when writing Argus programs. For example, the programmer need not be concerned that a program might expose inconsistencies of the sort discussed above to the user.

Our method of detecting orphans is to send extra information in some of the messages that guardians use to communicate, namely, call and reply messages, prepare messages, and queries and query responses. We also keep extra information at each guardian. First each guardian has a *crash count*; this is a stable counter that is incremented after each crash. In addition, the guardian maintains

two stable data structures: *done* and *map*. *Done* lists all actions known to this guardian that may have orphans somewhere. Like the *aborts_list*, it is not necessary to keep two actions in *done* where one is an ancestor of the other; instead only the older of the two actions need be kept. *Map* lists all guardians known to this guardian and their latest known crash counts.

In each message we include *done* and *map* of the sending guardian, and, in addition, the *dlist*, which lists the guardians *depended on* by the action on whose behalf the message is being sent. Intuitively, an action depends on a guardian if a crash of that guardian would cause it to become an orphan. The guardians an action depends on can be computed from information in plists: An action depends on all the guardians in its plist, but it also depends on all guardians listed in plists of all its ancestors at the time it was created. The *dlist* is maintained for each running action (along with the *plist*, etc.).

The information in a message is used at the receiving guardian to detect and destroy any local orphans and possibly to reject the incoming message; it is also merged with local information to bring that information more up to date. For example, a call message, C, is processed as follows:

1) We check for any action running at the receiving guardian that is a descendant of an action in C's *done*, or that depends on a guardian whose crash count in C's *map* is higher than what is known locally. Such an action is an orphan. These orphans are destroyed before the call message is acted upon.

2) We check that the call itself is not being performed on behalf of an orphan. The call is an orphan if its aid is a descendant of some action in the local *done*, or if any guardian in C's *dlist* has a lower crash count in C's *map* than in the local *map*. If the call action is an orphan, we send back a special reply rejecting the call.

3) Otherwise, we merge C's *map* and *done* with the local *map* and *done* and then run the handler action. In merging the maps, if the two maps disagree about the crash count of a guardian, the higher crash count is retained.

Orphan detection will prevent the problem illustrated by the scenario in Figure 11-3. When subaction T.1 of T runs at X, X has a higher crash count than it did when subaction S.1 ran there. This information is propagated to GT in the reply message of T.1, and then to Y in the call of T.2. The new information about X's crash count will then be recorded in Y's *map*. Later, when the call of S.2 arrives at Y, it will be rejected because the *map* sent in this call will contain the old crash count for X.

Notice that the information in the map is just what is needed to correct the problem in two-phase commit mentioned earlier. For example, consider the situation discussed earlier, in which A.2.1 committed, then G3 crashed and recovered, and later A.1.1 arrived. There are two possibilities. If A.1 and A.2 are sequential (in which case A.2 actually ran before A.1), then A.1 depends on G3; the *map* in the call message for A.1.1 will list G3's old crash count, so the call will be rejected by G3 as an orphan. Another possibility is that A.1 and A.2 are concurrent. In this case, whichever one commits to A first indicates one crash count for G3, while the second one to commit to A indicates a different crash count for G3. In either case, when the second one commits to A, A will be recognized as an orphan and aborted; two-phase commit will not be carried out.

In the above, we discussed how orphans are detected, but simply assumed that it was a simple matter to destroy an orphan. In fact, orphan destruction is not too difficult in Argus. Any process within a guardian can be destroyed without impact on the guardian's data provided that it is not in a critical section. Since the action of the process aborts, this ensures that any modifications to atomic objects are undone. (Note that we do rely on actions sharing only atomic objects; this restriction is needed if the actions are to be atomic.)

Argus processes enter critical sections in two ways: explicitly, by gaining possession of special built-in objects called *mutex* objects, which are similar to semaphores, and implicitly, by executing some system code that runs in a critical section. For example, the operations on the built-in atomic objects run in a critical section while examining the current status of the object (whether it is locked on behalf of some action). We keep track for each process of whether or not it is in a critical section. If it is not, we can destroy it immediately and abort its action. If it is, we let it run until it exits its critical sections. If it does not exit its critical sections, we can always crash the guardian as a last resort.

3.1. Optimization of the Orphan Detection Algorithm

The practicality of orphan detection depends primarily on the amount of information that need be sent in messages. Both *map* and *done* are potentially very large, but it is possible to limit the information that need be sent. Below we describe a technique developed by Walker [22] for reducing the sizes of both *map* and *done*. We describe how the scheme works for *done*, and then discuss how it can be used for *map*.

In this scheme, every action is assigned a *done-deadline*. This deadline is created whenever a top action is created; a subaction inherits the done-deadline of its parent. The deadline is some time in the future, e.g., several hours after the top action was created according to the clock of the creating guardian.

All guardians guarantee that the the top action and all its descendants will be destroyed as soon as the done-deadline is reached. Since no descendants of an action continue to run once the deadline is passed, it is not necessary to keep an action identifier in *done* once that action's deadline has passed. Thus, the length of time an action identifier remains in *done* is limited.

Of course, it is not possible for all guardians to recognize that descendants of an action have passed their deadlines at exactly the same instant, since the clocks at different nodes cannot be exactly the same. However, we can accommodate clock skew provided that some upper bound on the maximum deviation of different guardians' clocks can be defined. Thus we require that clocks be loosely synchronized. It is not difficult with today's technology to synchronize clocks to within a few minutes of one another, even in a large system [17].

The actual algorithm takes clock skew into account. When a guardian discovers an orphan based on comparing the orphan's done-deadline with its local clock, it destroys the orphan. However, it does not remove action identifiers from *done* the instant their deadline is passed. Instead, action identifiers are removed from *done* only after

D + epsilon

where D is the action done-deadline and epsilon is the maximum clock skew.

One problem with the above algorithm is that it is difficult to define a reasonable deadline when an action is created. If the deadline chosen is very large, then action identifiers of descendants of the action must remain in *done* a very long time. On the other hand, if the deadline is small, some action might be aborted just because its deadline passed, even though it is not an orphan. To provide a more flexible scheme, Walker describes a method of extending deadlines. (A similar method is described in [11].) Shortly before an action is due to expire because of its deadline, its guardian can query to determine whether the action's deadline can be extended. The response to this query indicates either that the action is an orphan or contains a new deadline. In the latter case, the new deadline replaces the old deadline and the action continues to run.

The deadline scheme can also be applied to the *map*. In this scheme, there is a system-wide map *deadline-interval*. Whenever a guardian recovers from a crash, it is entered in the local *map* with its new crash count, and a *map-deadline* equal to its current time plus the deadline-interval. In addition, when a top action is created it is assigned a map-deadline in addition to the done-deadline. The map-deadline is equal to its guardian's current time plus the deadline-interval. The map-deadline, like the done-deadline, is sent in all messages concerning the action, so all guardians at which descendants of the action run know the action's map-deadline.

Whenever a guardian detects that an action running locally is past its map-deadline, it destroys the action. Therefore, the system guarantees that no descendants of an action run past the map-deadline, modulo the clock skew.

Since no actions run past their map-deadlines, it is necessary to retain an entry in *map* only until

D + epsilon

where D is the entry's map-deadline and epsilon is the maximum clock skew. This scheme works for the following reason. Suppose some guardian G creates a top action A with map-deadline D_A and then crashes. Note that all descendants of A become orphans because of the crash. Subsequently G recovers and is entered in the *map* with some map-deadline D. No matter how quickly G recovers, it recovers after the creation of A, so

$$D_A \leq D$$

Therefore, the entry for the crashed guardian will remain in the *map* long enough to permit detection of descendants of A as orphans. Notice that the *map* will be propagated just as in the unoptimized algorithm. If the *map* containing the new crash count for D happens to reach a guardian at which a descendant of A is running, the entry for D will still be stored explicitly in the *map* because $D_A \leq D$.

Just as was the case with the done-deadline, it is possible to extend an action's map-deadline. However, if crashes are rare the map-interval can be very large without having *map* be large. Therefore, it is probably sufficient to have a large map-interval and not extend the map-deadline.

Done-deadlines for actions can be chosen independently at each guardian. However, such flexibility does not exist for the map-interval. For example, consider the situation shown in Figure 11-4. Here top action A was created at G. Its descendant A.1 was running at H, and A.1's descendant A.1.1 was running at I at the time H crashed. When H recovers, it must be entered in the *map* with a map-deadline greater than the map-deadline of A.1.1. The only way to guarantee this is to have H's map-interval be at least as big as G's. This requirement poses a problem if there is ever a need to change the map-interval. It is possible to increase the map-interval and have the change propagated incrementally throughout the system, but decreasing the map-interval is much more difficult.

It is too early to predict how well the deadline scheme will work. Walker performed an analysis (see [22]) that indicated the scheme will work well under "typical" system load. This analysis is based on assumptions about what a typical load is, for example, how often guardians create top actions and how often aborts happen. If these assumptions are correct, then the scheme will be successful. However, since

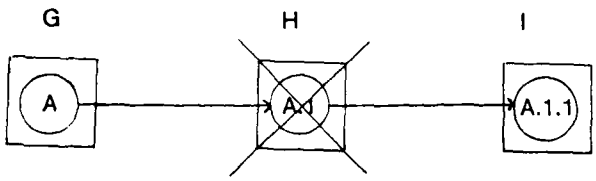


Figure 11-4: An Orphan Example.

the Argus implementation is not yet finished, it is too early for us to validate the assumptions. This will be possible only when statistics can be collected about the actual use of Argus.

4. SPECIFICATION AND IMPLEMENTATION OF ATOMIC TYPES

An atomic data type, like a regular abstract data type [15], provides a set of objects and a set of operations. An atomic type is an abstraction, and hence is described by a *specification*; it may be implemented by a *program*. As with regular abstract types, the operations provided by an atomic type are the only way to access or manipulate the objects of the type. Unlike regular types, however, an atomic type provides serializability and recoverability for activities that use objects of the type. Argus provides a number of built-in atomic types, and in addition provides facilities that permit users to implement new atomic types (see [24]).

Weihl [23] studied the problems of specifying and implementing atomic data types. In particular, he addressed three fundamental questions:

- *What is an atomic type?*
  We need a precise characterization of the behavior of atomic types. For example, we need to know how much concurrency can be allowed by an atomic type.

- *How do we specify an atomic type?*
  What aspects of the type's behavior must appear in the specification of an atomic type, and how should the specification be structured?

- *How do we implement an atomic type?*
  What problems must be solved in implementing an atomic type, and what kinds of programming language constructs make this task simpler?

To answer the first two questions existing work on concurrency control (or serializability) was generalized in three ways:

- The definition of atomicity is data dependent: It is based on an explicit specification of the desired behavior for the data types shared by

activities. This is crucial in achieving the concurrency required by applications.

- The definition of atomicity is integrated: Both serializability and recoverability are treated. This facilitates the description and verification of implementations of atomic types, which necessarily must cope with both.

- The focus is on modularity issues: Local properties of individual objects that ensure atomicity of activities are identified, and conditions under which different kinds of objects can be combined in a single system while preserving atomicity of activities are described.

Weihl explored three local properties, each of which is optimal: No strictly weaker local property suffices to ensure atomicity. The three properties characterize respectively the behavior of three classes of protocols: two-phase locking protocols (e.g., see [6], [18]), in which the serialization order of activities is determined by the order in which they access objects; *multi-version timestamp-based protocols* (e.g., see [21]), in which the serialization order of activities is determined by a predetermined total order; and hybrid protocols (e.g., see [1], [3], [5]), which use a combination of these techniques.

Weihl also presented a novel locking protocol and verified its correctness. His protocol generalizes previously existing protocols in two ways: First, it permits the results of operations, as well as their arguments, to be used in determining the appropriate lock mode. This extra information can be used to allow greater concurrency. Second, the protocol handles partial and non-deterministic operations. It thus has a wider range of application than previously existing protocols, which require operations to be both total and deterministic. In addition, the implementation of both synchronization and recovery are described and verified; descriptions of previously existing protocols are limited to synchronization alone.

Weihl's approach to specifying atomic types permits the programmer of an individual activity to ignore *how* atomicity is achieved. To reason about whether an individual activity preserves consistency, one needs only the serial specification of each type used by the activity, and the knowledge that activities are atomic; one need not know how types cooperate to ensure atomicity.

In addition, Weihl's specification framework supports an approach that permits the concurrent specification of a type to be derived systematically from a specification of its sequential behavior. Thus, the problem of specifying a type is reduced to the simpler problem of specifying how it should behave in the absence of concurrency.

Finally, several example implementations of atomic types are presented in [23],

illustrating how existing techniques for synchronization and recovery can be extended to use information about the specifications of objects to increase concurrency. Linguistic support for atomic types is also explored through an analysis of the advantages and disadvantages of several alternative approaches.

In the remainder of this section we describe informally one of the local atomicity properties explored by Weihl, and discuss how it relates to two-phase locking protocols.

4.1. Description of Dynamic Atomicity

Atomic activities are characterized by two properties: serializability and recoverability. *Serializability* means that the concurrent execution of a group of activities is equivalent to some serial execution of the same activities. *Recoverability* means that each activity appears to be all-or-nothing: Either it executes successfully to completion (in which case we say that it *commits*), or it has no effect on data shared with other activities (in which case we say that it *aborts*).

The problem in defining a local property of objects that ensures global atomicity of activities is illustrated by the following example. Consider a system containing two objects, *x* and *y*, each with read and write operations, and each with initial value 0. Suppose that *x* is implemented using two-phase locking, and *y* is implemented using multi-version timestamping. Now suppose there are two activities *a* and *b*, with timestamps 1 and 2, respectively. Consider the following execution:

b reads x, receiving 0.
b writes 1 into y.
b commits.
a reads y, receiving 0.
a writes 1 into x.
a commits.

This execution is not serializable: In a serial execution, the second activity should see the value written by the first, but both *a* and *b* read the initial values of one of the objects. However, the execution is atomic at each object: At *x*, *b* is serializable before *a*, and at *y*, *a* is serializable before *b*. In a sense we have an agreement problem: For activities to be atomic, the objects in the system must agree on at least one serialization order for the committed activities. In this case *x* and *y* do not agree on a serialization order for *a* and *b*, and atomicity is violated. Achieving agreement can be difficult because each object is aware of only the events in which it participates. In other words, each object has purely *local* information; no object has complete information about the *global* computation of the system. One can imagine building a system in which objects have more global information; however, such an organization suffers from lack of modularity.

The reason that *x* and *y* do not agree on a serialization order in this example is that they use incompatible protocols to ensure atomicity. If both objects used two-phase locking, or both used multi-version timestamping, the above execution could not occur and atomicity would be guaranteed. Our goal in defining a local atomicity property is to ensure that activities are atomic whenever all objects shared by the activities satisfy the local atomicity property. Note that any such local atomicity property must be satisfied by at most one of the objects *x* and *y*. The property *dynamic atomicity*, described below, ensures global atomicity by ruling out objects like *y* while allowing objects like *x*.

Dynamic atomicity characterizes the behavior of objects implemented with protocols like two-phase locking, in which the serialization order of activities is determined dynamically based on the order in which activities access objects. The essence of such protocols is the notion of *delay*: If the steps executed by one activity conflict with the steps executed by another activity, then one of the activities must be delayed until the other has committed. As discussed in [23], an implementation of dynamic atomicity need not actually delay activities to achieve this effect. All that is necessary is that the overall effect for committed activities be as if conflicts were resolved by delays. Indeed, most optimistic protocols (e.g., see [10]) resolve conflicts by aborting some activities when they try to commit, but the overall effect is as required by dynamic atomicity.

Weihl captures this notion of delay more precisely as follows: Given an execution *h*, define the relation *precedes*(*h*) to contain all pairs ⟨*a*, *b*⟩ (where *a* and *b* are activities) such that some operation invoked by *b* in *h* terminates after *a* commits in *h*.[1]

We can illustrate this notion by means of example executions. In the examples, suppose *x* is a set object with two operations, *insert* (to insert a new element) and *member* (to determine whether a given element is in *x*). In example executions, events are denoted by triples, indicating the type of the event, the object at which it occurred, and the activity involved. Thus, for example, the triple ⟨insert(2),x,a⟩ denotes the invocation of the insert operation with argument 2 at object *x* by activity *a*. Similarly, the triple ⟨ok,x,a⟩ denotes the termination, with result "ok," of an operation invoked by *a* on *x*.

Now let *h* be the following execution:

---

[1]Note that there is a distinction between an activity completing by committing or aborting, and an operation terminating by returning information. An activity may execute any number of operations before completing. In addition, it is possible for *b* to commit after *a* in *h*, yet for all of *b*'s operations to terminate before *a* commits; in this case the pair ⟨*a*,*b*⟩ is not in *precedes*(*h*).

⟨insert(2),x,a⟩
⟨ok,x,a⟩
⟨member(3),x,b⟩
⟨false,x,b⟩
⟨commit,x,b⟩
⟨commit,x,a⟩

Then *precedes*(*h*) is the empty relation (no operation invoked by *a* or *b* terminates after the other activity commits). However, if *h* is the execution

⟨insert(2),x,a⟩
⟨ok,x,a⟩
⟨member(3),x,b⟩
⟨commit,x,a⟩
⟨false,x,b⟩
⟨commit,x,b⟩

then *precedes*(*h*) contains the pair ⟨*a*,*b*⟩.

The relation *precedes*(*h*) captures our intuitive notion of delay: If *b* is delayed until after *a* commits in an execution *h*, then ⟨*a*,*b*⟩ will be in *precedes*(*h*). Thus, if two committed activities conflict, dynamic atomicity requires one of them to "precede" the other. Turning this statement around, if neither ⟨*a*,*b*⟩ nor ⟨*b*,*a*⟩ is in *precedes*(*h*), then *a* and *b* must not conflict in *h*. In other words, they must be serializable in the order *a* followed by *b* (written *a*-*b*) and in the order *b*-*a*.[2] In general, dynamic atomicity requires all executions *h* permitted by an object to satisfy the following property: The committed activities in *h* must be serializable in all total orders consistent with *precedes*(*h*).[3]

For example, the following execution *h* is atomic, but does not satisfy the requirements for dynamic atomicity:

---

[2] Serializability is defined with respect to the specifications of the objects involved. A history *h* can be serialized in a given order if the specifications of the objects permit the activities in *h* to be executed serially in that order so that they invoke the same operations as in *h* and receive the same results.

[3] As noted in [23], natural restrictions on executions guarantee that the "precedes" relation is a partial order, ensuring that there are total orders consistent with it.

Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, March 1984.

Theses in Progress

1. 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, expected January 1985.

2. Restivo, J. P. "Addition of Type Information to the Argus Debugger," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected June 1985.

Talks

1. Chiu, S-Y. "Debugging Distributed Computations in a Nested Atomic Action System,"
   DEC Systems Research Center, Palo Alto, CA, May 1984
   Xerox PARC, Palo Alto, CA, May 1984
   IBM Research Laboratory, San Jose, CA, May 1984
   Hewlett-Packard Laboratories, Palo Alto, CA, May 1984

2. Herlihy, M. P. "Replication Methods for Abstract Data Types"
   Harvard University, Cambridge, MA, February 1984
   Xerox PARC, Palo Alto, CA, March 1984
   DEC Systems Research Center, Palo Alto, CA, March 1984
   Stanford University, Palo Alto, CA, March 1984
   SRI International, Menlo Park, CA, March 1984
   University of California at Berkeley, Berkeley, CA, March 1984
   Carnegie-Mellon University, Pittsburgh, PA, March 1984
   MIT, Cambridge, MA, April 1984
   DEC Corporate Research, Hudson, MA, April 1984

3. Herlihy, M. P. "Issues in Process and Communication Structure for Distributed Programs," The Third IEEE Symposium on Reliability in Distributed Software and Database Systems, October 1983.

4. Liskov, B. H. "Argus: The Programming Language and System,"
   Xerox PARC, Palo Alto, CA, August 1983
   Cornell University, Ithaca, NY, November 10, 1983
   Brown University, Providence, RI, May 1, 1984

8. Weihl, W. E. "Data-Dependent Concurrency Control and Recovery," *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, Montreal, Canada, 63-75, August 1983.

9. Weihl, W. E. "Specification and Implementation of Atomic Data Types," MIT/LCS/TR-314, MIT Laboratory for Computer Science, Cambridge, MA, March 1984.

10. Weihl, W. E. and Liskov, B. "Implementation of Resilient, Atomic Data Types," to appear in *ACM Transactions on Programming Languages and Systems*.

Theses Completed

1. Beardsley, N. A. "An Algorithm for Compile-Time Detection of Uninitialized Variables in CLU," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, May 1984.

2. Chan, M. W. "Evaluation of Argus Through the Implementation of a Distributed Calendar System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

3. Della Fera, A. "SCOPE, A System for CLU Object Perusal and Editing," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

4. Gosling, C. "A Catalog for the Argus System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

5. Herlihy, M. P. "Replication Methods for Abstract Data Types," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

6. Lancaster, J. N. "Naming in a Programming Support Environment," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1983.

7. Walker, E. F. "Orphan Detection in the Argus System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

8. Weihl, W. E. "Specification and Implementation of Atomic Data Types,"

22. Walker, E. F. "Orphan Detection in the Argus System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

23. Weihl, W. E. "Specification and Implementation of Atomic Data Types," MIT/LCS/TR-314, MIT Laboratory for Computer Science, Cambridge, MA, March 1984.

24. Weihl, W. and Liskov, B. "Implementation of Resilient, Atomic Data Types." To appear in *ACM Transactions on Programming Languages and Systems*.

Publications

1. Dolev, D., Lynch, N. A., Pinter, S., Stark, E. W. and Weihl, W. E. "Reaching Approximate Agreement in the Presence of Faults." *Proceedings of the Third IEEE Symposium on Reliability in Distributed Software and Database Systems*, October 1983.

2. Herlihy, M. P. "Replication Methods for Abstract Data Types," MIT/LCS/TR-319, MIT Laboratory for Computer Science, Cambridge, MA, May 1984.

3. Lancaster, J. N. "Naming in a Programming Support Environment," MIT/LCS/TR-312, MIT Laboratory for Computer Science, Cambridge, MA, 1983.

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

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

6. Liskov, B. and Scheifler, R. "Guardians and Actions: Linguistic Support for Robust Distributed Programs." *ACM Transactions on Programming Languages and Systems* 5, 3 (July 1983), 381-404.

7. Liskov, B., et al. "Preliminary Argus Reference Manual," Programming Methodology Group Memo 39, MIT Laboratory for Computer Science, Cambridge, MA, October 1983.

11. Lampson, B. "Applications and Protocols." *Distributed Systems: Architecture and Implementation, Lecture Notes in Computer Science 105*, Goos and Hartmanis (eds.), Springer-Verlag, Berlin, 1981.

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

13. Lancaster, J. N. "Naming in a Programming Support Environment," MIT/LCS/TR-312, MIT Laboratory for Computer Science, Cambridge, MA, 1983.

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

15. Liskov, B. and Zilles, S. "Programming with Abstract Data Types," *Proceedings of ACM SIGPLAN Conference on Very High Level Languages, SIGPLAN Notices* 9, 4 (April 1974), 50-59.

16. Liskov, B., et al. "Preliminary Argus Reference Manual," Programming Methodology Group Memo 39, MIT Laboratory for Computer Science, Cambridge, MA, October 1983.

17. Marzullo, K. "Loosely-Coupled Distributed Services: A Distributed Time Service," Ph.D dissertation, Stanford University, Department of Computer Science, Stanford, CA, 1983.

18. Moss, J. E. B. "Nested Transactions: An Approach to Reliable Distributed Computing," MIT/LCS/TR-260, MIT Laboratory for Computer Science, Cambridge, MA, 1981.

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

20. Postel, J. "Internet Protocol," Report RFC 791, Defense Advanced Research Projects Agency, Information Processing Techniques Office, Arlington, VA, September, 1981.

21. Reed, D. P. "Naming and Synchronization in a Decentralized Computer System," MIT/LCS/TR-205, MIT Laboratory for Computer Science, Cambridge, MA, 1978.

References

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

2. Bernstein, P., Goodman, N., and Lai, M.-Y. "Two Part Proof Schema for Database Concurrency Control," *Proceedings of the Fifth Berkeley Workshop on Distributed Data Management and Computer Networks*, 71-84, February 1981,

3. Chan, A., et al. "The Implementation of an Integrated Concurrency Control and Recovery Scheme," Technical Report CCA-82-01, Computer Corporation of America, Cambridge, MA, March 1982

4. 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, to appear.

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

6. Eswaren, 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.

7. Gray, J. N. "Notes on Data Base Operating Systems," *Lecture Notes in Computer Science 60*, Goos and Hartmanis (eds.), Springer-Verlag, Berlin, 1978, 393-481.

8. Herlihy, M. P. "Replication Methods for Abstract Data Types," MIT/LCS/TR-319, MIT Laboratory for Computer Science, Cambridge, MA, May 1984.

9. Korth, H. F. "Locking Protocols: General Lock Classes and Deadlock Freedom," Ph.D dissertation, Princeton University, Department of Computer Science, Princeton, NJ, 1981.

10. Kung, H.T. and Robinson, J.T. "On Optimistic Methods for Concurrency Control," *ACM Transactions on Database Systems* 6, 2 (June 1981), 213-226.

concurrency control method employing state information. This method is general: it can be used to implement any concurrency control scheme that preserves serializability, but at the potential cost of increased message traffic and additional constraints on availability.

Herlihy's reconfiguration technique allows the availability properties of replicated data to be changed dynamically. The ability to reconfigure incurs a negligible cost when it is not used. When an object is actually reconfigured, the technique imposes a cost in the form of a temporary period of increased message traffic and reduced availability. A replicated reference counting scheme is introduced to discard information rendered obsolete by reconfiguration.

Herlihy also developed an extension to the method that increases availability in the presence of partitions. This technique preserves serializability, but not external consistency. It does not require explicit partition detection or static transaction classes, it is independent of the concurrency control method, and it imposes negligible costs when it is not used.

If queues were implemented in terms of files, then files must be read and written to implement both *Enq* and *Deq*, leading to the quorum choices shown in Figure 11-6. However, the dependency relation for queues allows more freedom in choosing quorums, as is shown Figure 11-7. (In this figure, we distinguish *Deq* events that terminate normally from those that terminate abnormally, since different final quorums can be used in the two cases.) This extra flexibility can be used to increase the availability of the *Enq* operation at the expense of decreased availability of the *Deq* operation. Such a tradeoff would allow clients of a printer service to continue queuing requests for printing even when the device itself is down.

Enq	(3,3)
Normal Deq	(3,3)
Abnormal Deq	(3,0)

Figure 11-6: Quorum Choices Available for Queues Implemented by Files.

Enq	(0,1)	(0,2)	(0,3)
Normal Deq	(5,1)	(4,2)	(3,3)
Abnormal Deq	(5,0)	(4,0)	(3,0)

Figure 11-7: Quorum Choices for Queues Implemented by Logs.

5.3. Additional Results

In addition to developing the basic replication method, Herlihy also extended the method to include concurrency control, to allow the system to be reconfigured, and to permit continued execution in the presence of partitions.

Although the method permits replication and concurrency control to be implemented independently, as was assumed above, a higher level of concurrency can be supported if the concurrency control method is integrated with the replication method. Herlihy developed a simple and efficient concurrency control method based on predefined operation conflicts. This method is optimal for the amount of information it uses: no concurrency control method based exclusively on predefined conflicts can support a higher level of concurrency. Nevertheless, the method has two disadvantages: because it does not take advantage of state information, it is inherently limited in the level of concurrency it can support, and it provides poor support for partial operations.

To remedy these shortcomings, Herlihy also developed a more powerful

Informally, a request depends on an event if omitting that event from a request's view might produce an incorrect response. For example, consider files with the standard read and write operations. *Read* requests depend on (all) *Write* events (events whose request part is a write), because the correct response to *Read* requires knowledge of prior *Write* events. *Write* requests do not depend upon prior *Read* or *Write* events, because the response to the *Write* is always the same. Similarly, *Deq* requests depend on prior *Enq* and normal *Deq* events (*Deq* events that return an item), because knowledge of these events is needed to determine which item to dequeue. *Enq* requests do not depend on any prior events, because *Enq*, like *Write*, has only one possible response. Expressing the dependency relation in terms of events allows a request to depend on just a subset of the events associated with a single operation.

Constraints on quorum intersections can be derived from the dependency relation as follows: To carry out a request, the initial quorum must intersect with the final quorum of all operation executions leading to events depended on by the request. It can be shown that choosing quorums in this manner leads to a correct implementation, and that no weaker set of constraints can guarantee correctness.

For example, the initial quorum for a read request must intersect the final quorum for every execution of a write operation, since read depends on all write events. However, the initial quorum of a write request needs to intersect nothing, since a write request depends on no events. The possible quorum choices for five DM sites are shown in Figure 11-5.

Read	(1,0)	(2,0)	(3,0)	(4,0)	(5,0)
Write	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)

Figure 11-5: Quorum Choices for Files Replicated at Five Nodes.

In this figure, each row shows possible choices for the initial and final quorums for a set of events; the row is labeled to identify the set of events. Thus "read" refers to all events associated with executions of the read operation, while "write" refers to all events associated with executions of the write operation. Each column shows a set of consistent choices for all the rows. Thus, if in carrying out a read request we read from an initial quorum consisting of two DM's (and write to none), then in performing a write operation we must write to a final quorum consisting of four DM's. In any given system, one of the columns must be chosen.

By providing direct support for objects of arbitrary abstract type, our replication method imposes fewer constraints on quorum choice than existing replication methods based on files. For example, consider the queue object discussed above.

some action A would not be visible to other operations that use that log on behalf of other actions until A terminates. Furthermore, if A aborts, then its modifications to the log would be undone.

As mentioned above, an entry in the log records an operation that has been applied to the object. Each log entry is stamped with a unique timestamp. This timestamp is selected by the concurrency control system in such a way that the timestamps define a total order that is consistent with the serialization order of the actions that cause the operations to be executed. For example, if logs were implemented as built-in atomic objects of Argus, then timestamps would be chosen as part of two-phase commit, and written into their associated entries automatically as part of phase 2.

An operation is executed in the following four steps.

1) When a TM receives a request from a client, it reads the logs from an initial quorum of DM's for that request. These logs are merged in timestamp order to construct a larger log called a *view*.

2) The TM chooses a response consistent with the object state as defined by the view.

3) The TM records the request and response by appending a new entry to the view, and sending the modified view to a final quorum of DM's for that event. Each DM in the final quorum merges the view with its resident log.

4) The response is returned to the client.

An analysis of the algebraic structure of an object's type is used to derive a *dependency relation*, which can then be used to derive a set of constraints on quorum intersections. The basic idea is that to carry out a request to execute an operation, it is necessary to observe the effects of some of the operation executions that occurred in the past. A dependency relation thus relates a request to a set of past executions. This notion of past executions is expressed in terms of *events*, which are request-response pairs.

For example, consider a queue object with an *Enq* operation to enqueue an item at the head of a queue and a *Deq* operation to dequeue an item from the tail of the queue. The *Enq* operation always terminates normally; an execution of *Enq* is represented by an event <Enq(x), ok>. The *Deq* operation terminates normally if the queue is non-empty, leading to an event of the form <Deq(), ok(x)>; however, if the queue is empty the operation terminates abnormally, described by an event of the form <Deq(), empty>.

can be detected by the absence of a response, we do not assume that the different kinds of failure can be distinguished: the absence of a response may indicate that the original message was lost, that the reply was lost, that the recipient has crashed, or simply that the recipient is slow to respond.

The basic containers for data are called *objects*. Each object has a *type*, which characterizes its behavior by defining a set of possible *states* together with a set of primitive *operations* that provide the (only) means to create and manipulate objects of that type. Each type has an accompanying *specification* that gives the meaning of the operations provided by the type. A *replicated object* is an object whose state is stored redundantly at multiple sites.

Replicated objects are implemented by two kinds of modules: Data Managers (DM's) and Transaction Managers (TM's). (Both the TM's and DM's could be Argus guardians.) The object's state is replicated among the DM's and managed by the TM's. DM's serve as long-term repositories for the object's state, while TM's execute the object's operations on behalf of the object's clients. To apply an operation to a replicated object, a client sends a request to a TM for the object. The TM reads the data from some collection of DM's, carries out a local computation, sends updates to some collection of DM's, and returns a response to the client. Each operation is executed atomically.

An operation's availability to a client depends on two factors: the client must first be able to locate a TM for the object, and the TM must in turn locate enough DM's to carry out the operation. Because the TM's do not interact directly with one another, new TM's can be created without affecting existing ones. Consequently, TM's can be replicated to an arbitrary extent, implying that the availability of the replicated object is determined by the availability of the DM's.

5.2. The Replication Method

Unlike most other replication methods, the object is not represented by a collection of copies; instead, each DM maintains a partial log of the operations that have been applied to the object. The method is based on the notion of a *quorum*, which is a set of DMs. Associated with each operation are a set of *initial quorums* and a set of *final quorums*. To execute the operation, it is necessary to read the logs from all members of one of the initial quorums, and update the logs of all members of one of the final quorums.

In the following we assume that concurrency control is implemented independently of the replication method. This could be accomplished by having each log be a built-in atomic object of Argus. In this case the system would manage read and write locks for the logs, and changes made by one operation to some log on behalf of

5. REPLICATION

Replicated data is data that is stored redundantly at multiple locations in a distributed system. Replication can enhance the availability of data in the presence of failures, increasing the likelihood that the data will be accessible when it is needed. Replication is needed to implement distributed programs in which a high level of availability is important, such as banking systems, airline reservation systems, authentication servers, and mail systems.

Herlihy [8] has developed a new method for managing replicated data. Unlike many methods that support replication only for uninterpreted files, his method makes use of type-specific properties of objects (such as sets, queues, or directories) to provide more effective replication. The method is both general and systematic. It is general because it is applicable to objects of arbitrary type, and it is systematic because constraints on correct implementations are derived directly from the algebraic structure of the data type in question. These constraints are both necessary and sufficient: any replicated implementation satisfying these constraints is correct, and no smaller set of constraints guarantees correctness. Herlihy's method is described below.

5.1. Model of Computation

The replication method is based on a model of computation similar to the one assumed by Argus. A distributed system consists of a collection of *sites* connected (only) by a communication network. A site consists of one or more processors, one or more levels of memory, and any number of devices. We assume that any site can communicate with any other when the network is functioning properly. We make no assumptions about the speed, connectivity, or reliability of the network.

The model admits two kinds of failures: site crashes and communication failures. When a site crashes, its resident data becomes temporarily or permanently inaccessible. We assume that all communication failures take the form of lost messages: garbled and out-of-order messages can be detected (with very high probability) and discarded. Transient communication failures may be hidden by lower level protocols, but longer-lived failures can result in situations where functioning sites are unable to communicate. One such situation is a network *partition*, in which the sites are divided into disjoint sets such that functioning members of different sets cannot communicate. Partitions are not the only such situations that can arise: for example, one site may be able to send messages to another, but not vice-versa.

A failure is detected when a site that has sent a message fails to receive a response after a certain duration. Although we assume that site crashes and lost messages

Notice that two deposit operations commute: The results they return and the final state of the account do not depend on the order in which the operations are executed. However, two withdraw operations do not commute: If the balance in the account is large enough to cover either but not both of the withdrawals, then both the results returned by the operations and the final state of the account depend on the order in which the operations are executed.

Now consider the following execution *h*, in which *a* first deposits \$10 in the account and commits, and then *b* and *c* concurrently withdraw \$4 and \$3, respectively, and then commit:

<deposit(10),y,a>
<ok,y,a>
<commit,y,a>
<withdraw(4),y,b>
<withdraw(3),y,c>
<ok,y,c>
<ok,y,b>
<commit,y,c>
<commit,y,b>

This execution satisfies the requirements for dynamic atomicity: *Precedes*(*h*) contains the pairs $\langle a,b \rangle$ and $\langle a,c \rangle$, and *h* is serializable in the orders *a*-*b*-*c* and *a*-*c*-*b*. However, since withdraw operations do not commute, none of the two-phase locking protocols based on commutativity can permit this execution, in which *b* and *c* execute withdraw operations concurrently, to occur.

Dynamic atomicity allows such an execution because the operations executed by *a* can be considered in deciding when those invoked by *b* and *c* can be scheduled. History-independent protocols, like those in [2], [6], [9], must schedule *b* and *c* the same way regardless of operations executed earlier by other activities like *a*, and so cannot permit *b* and *c* to execute withdraw operations concurrently.

Similar situations arise with any data type that behaves like a finite pool of resources, with operations to add objects to and remove objects from the pool: Dynamic atomicity permits activities to remove objects from the pool concurrently, but locking implementations (indeed, any history-independent protocol) will not permit this level of concurrency.

Locking protocols are clearly useful for many applications, and can be implemented relatively easily. The examples above illustrate, however, that there may be applications for which locking protocols are inadequate. Weihl presents several implementations that achieve more concurrency than is possible with locking. Not surprisingly, some of these implementations are more complex than the corresponding ones based on locking. It remains to be seen whether the increased concurrency is worth the added complexity of the implementations.

<member(2),x,a>
<insert(3),x,b>
<ok,x,b>
<false,x,a>
<member(3),x,c>
<commit,x,b>
<true,x,c>
<commit,x,a>
<commit,x,c>

satisfies the requirements for dynamic atomicity. *Precedes*(*h*) contains the single pair <*b*,*c*>, and *h* is serializable in the orders *a*-*b*-*c*, *b*-*a*-*c*, and *b*-*c*-*a*.

Weihl shows that if all objects in a system are dynamic atomic, then the activities in the system are atomic. The proof hinges on the fact that objects never "disagree" about the "precedes" relation: There is always at least one total order that is consistent with all of the local "precedes" relations. Thus, if each object ensures serializability in all total orders consistent with its local "precedes" relation, then objects will agree on at least one (global) serialization order. This means that dynamic atomicity satisfies our goals for a local atomicity property. Weihl also shows that dynamic atomicity is *optimal*: No strictly weaker (i.e., more permissive) property of objects suffices to ensure global atomicity.

4.2. Relationship to Locking

Dynamic atomicity is a property of objects, not a protocol or an implementation technique. It characterizes the behavior of objects implemented using two-phase locking protocols like those in [2], [6], [9]. These protocols are each based on some notion of "commutativity:" Activities are allowed to execute operations concurrently only if the operations "commute." Locking protocols based on commutativity have two structural limitations that prevent them from achieving all of the concurrency allowed by dynamic atomicity. First, they are conflict-based: Synchronization is based on a pair-wise comparison of operations executed by concurrent activities. In contrast, dynamic atomicity depends on the *sequences* of operations executed by activities. Second, the protocols are history-independent: Synchronization is independent of past history, in particular the operations executed by committed activities. In contrast, dynamic atomicity depends on the entire execution.

Consider, for example, a system containing a bank account object *y* with three operations: *deposit*, which adds a specified amount to the balance of the account; *withdraw*, which either removes a specified amount from the account if the balance will remain non-negative (terminating with result "ok"), or leaves the account untouched if the account has insufficient funds to cover the withdrawal (terminating with result "no"); and *balance*, which returns the current balance in the account.

<member(3),x,a>
<insert(3),x,b>
<ok,x,b>
<false,x,a>
<member(3),x,c>
<commit,x,b>
<true,x,c>
<commit,x,a>
<commit,x,c>

h is serializable in the order *a*-*b*-*c*, since the specification of the set object *x* allows the following execution:

<member(3),x,a>
<false,x,a>
<commit,x,a>
<insert(3),x,b>
<ok,x,b>
<commit,x,b>
<member(3),x,c>
<true,x,c>
<commit,x,c>

However, since *precedes*(*h*) contains only the single pair <*b*,*c*>, *h* must also be serializable in the orders *b*-*a*-*c* and *b*-*c*-*a* to be dynamic atomic. This is not the case; for example, the serial execution

<insert(3),x,b>
<ok,x,b>
<commit,x,b>
<member(3),x,a>
<false,x,a>
<commit,x,a>
<member(3),x,c>
<true,x,c>
<commit,x,c>

is not permitted by the specification of *x*.

As another example, the execution

5. Liskov, B. H. International Professorship in Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, January 23-27, 1984

"Introduction to CLU"
"Specifying Data Abstractions";
"Program Construction Using Abstractions";
"Using aBstractions in Programming Languages"

6. Liskov, B. H. International Professorship in Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, January 23-27, 1984

"The Argus Language and System"
"Concepts and Issues"; "Argus Features"; "Example";
"Subsystems"; "Implementation"; "User-Defined Atomic Data Types"; "Discussion"

7. Liskov, B. H. Advanced Course on Distributed Systems -- Methods and Tools for Specification, Munich, Germany, April 9 - 13, 1984

"The Argus Language and System"
"Concepts and Issues"; "Argus Features"; "Example";
"Subsystems"; "Implementation"; "User-Defined Atomic Data Types"; "Discussion"

8. Weihl, W. E. "Data-Dependent Concurrency Control and Recovery," Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Canada, August 17, 1983.

9. Weihl, W. E. "Specification of Atomic Data Types,"

Cornell University, Ithaca, NY, December 1, 1983
Harvard University, Cambridge, MA, March 5, 1984
MIT, Cambridge, MA, March 19, 1984
AT&T Bell Laboratories, Murray Hill, NJ, March 26, 1984
DEC Systems Research Center, Palo Alto, CA, April 2, 1984
University of California at Berkeley Berkeley, CA, April 3, 1984
IBM Research Laboratory, San Jose, CA, April 4, 1984
Stanford University, Stanford, CA, April 5, 1984
DEC Corporate Research, Hudson, MA, April 11, 1984
Tufts University, Medford, MA, April 12, 1984
Carnegie-Mellon University, Pittsburgh, PA, April 16, 1984

PROGRAMMING TECHNOLOGY

Academic Staff

A. Vezza, Group Leader

Research Staff

T. Anderson
D. Lebling
J. Licklider
R. Myhill
C. Reeve

Graduate Students

D. Buffo
D. Lee
P. Lim
I. Rahim
R. Sun
A. Yeh

Undergraduate Students

H. Baek
J. Dike
M. Duke
D. Elrod
D. Fagan
M. Herdeg
J. Hirata
C. Humphreys
Y. Jo-Kim
T. Kim
M. Kudisch
S. Kukolich
B. Law
S. Malone
M. McEntee
S. Meeks
C. Naylor
S. Pitts
G. Shaw
C. Shepard
J. Shivanandan
B. So
J. Wang

Support Staff

A. Finn
N. Mims

1. INTRODUCTION

This is the report of the Programming Technology Group. The main effort of the group continues to be development of a computer-based planning system to be used where voluminous data is available and necessary to support planning. Within the Programming Technology Group this year, development has focused on finishing and refining the following:

- Movement of a large system of software development tools, centered upon MDL, from the DECSYSTEM-20/TOPS-20 environment in which it was initially constructed, to an open set of environments including Vax-Unix and systems based on the 68000 family of chips.

- A planning aid based on the essential ideas of VisiCalc, but including symbolic computation, data management, and graphic display.

- A graphical programming and monitoring system.

The language MDL, that is the centerpiece of the system of software development tools being transported to the Vax and 68000 environments, is both an extension of LISP and a production language that supports programming in a more conventional Fortran-like style. The acronym "MIM," used in the remainder of the report, stands for Machine Independent MDL.

Currently the group's work is focused on implementation of the planning system at LCS headquarters and at DARPA. A Vax-11/780 was installed at LCS headquarters in the late winter of 1984 and LCS administration is expected to start using the planning system some time this summer or fall for a full range of financial and personnel planning activities. A Vax-11/780 has also been purchased for DARPA and a version of the planning system is expected to be operating there in the fall of 1984.

2. MIM COMPILER DEVELOPMENT

At the beginning of the reporting year, the three compilers that constitute the current set for MIM (MIMC, MIMOC20 and MIMOCVax) were all working adequately. However, increased usage of the compilers by the user community revealed some shortcomings in them. There were three areas of concern:

1) The code produced ran too slowly.

2) The compilers themselves ran too slowly.

3) The compilers ran in old MDL as opposed to MIM.

A large portion of the group's effort was spent addressing these issues and significant progress has been made. Both MIMOC20 and MIMOCVax, the open compilers for the DECSYSTEM-20 and Vax/Unix, were improved in a number of ways.

- The register allocation algorithm for MIMOC20 was completely overhauled . Optimizations were added to prevent unnecessary storing of temporaries in memory and to keep values in registers during tight loops. These optimizations already existed in MIMOCVax, so it was not necessary to implement them.

- A "look ahead" mechanism was added to both of the MIMOCs. This mechanism enables two or three different virtual MIMA instructions to be compiled together into less object code instructions. For example, if a series of MIMA instructions extracts a value from a structure, adds one to it and puts the result back into the structure, this can compile into one instruction on either the DECSYSTEM-20 or the Vax. Without the look ahead mechanism, this kind of combining of instructions would be impossible.

- In order to decrease the overhead involved in making function to function calls within a compiled package of functions, a faster internal calling sequence was added to MIMOC20. This also permits the compiler to throw away the entry information for the internal compiled functions. This optimization may be added to MIMOCVax at some later time.

- All three MIM compilers were found to be unacceptably slow for reasonable use. A major reason for this was the necessity of always compiling an entire file at one time when in fact only one or two functions had changed since the file was previously compiled. A partial recompilation facility has been added to all three of the compilers. This facility permits a user to recompile only those functions that have changed since the last time the program was compiled.

- Unfortunately using the precompilation facility precludes producing glued code in MIMOC20 and MIMOCVax. A program to glue code produced by MIMOC20 has been written to address this problem. This enables a user to take advantage of both the convenience of partial recompilation and performance of glued function calls. Since the Vax has variable length instructions, writing an after the fact glue program is much more complex. As of now, we haven't undertaken it.

- It is always dangerous to compile a compiler using itself because the possibility of introducing very obscure bugs is always present. These obscure bugs come about because if the compiler miscompiles itself, the error may not show up until something else is compiled. This is often referred to as the recursive compiler bug. In order to minimize the likelihood of recursive compiler bugs, we delayed moving the compilers from old MDL106 into MIM for as long as possible. We wanted to make sure things were very reliable before doing this. That level of reliability has been achieved and all three compilers are now running in MIM.

- Moving the compilers from MDL106 to MIM did introduce some new performance problems especially in I/O. MIM's I/O is very general and flexible but is also slower than MDL106 I/O. However, by writing special output programs that are specific to the compilers' needs, we were able to recover the performance that was lost.

- Additional benefits of moving the compilers into MIM include: the ability to compile larger programs, the capability of running the compilers on the Vax as well as the DECSYSTEM-20 and obviation of conditionals in target programs to get around incompatibilities between MIM and MDL106.

The fact that MIMOC20 and MIMOCVax do many similar things in many different ways has led us to rethink our approach to these programs. While it was appropriate to investigate different ways of writing open compilers early in the project, it has become apparent that this approach involves a lot of duplicated effort. We have just begun work on a table-driven, open compiler that will be easily adaptable to new machines. We believe this approach will result in a more maintainable system.

3. MIM DEVELOPMENT

Although machine-independent MDL (MIM) had, by the end of the last reporting period, reached a point where it could be used in place of old MDL for most purposes, it was still missing some features that MDL had, and its performance left something to be desired. During this year, many of its deficiencies were remedied. Where possible, new features appeared in both the TOPS-20 and the Vax/Unix versions of MIM; in view of the group's continued movement away from TOPS-20, unique features appeared only in the Unix version.

- The PURIFY facility of MDL allowed user programs and data structures to be made read-only, resulting in two advantages: the read-only structures were not examined by the garbage collector, thus speeding things up; and the memory occupied by read-only structures could be

shared among several processes, thus reducing the substantial paging load that MDL put on the system. We added this feature to both MIMs this year. Unix doesn't yet allow shared memory in this form, but the Unix implementation of PURIFY does act to increase the virtual memory available to a MIM process.

- GC-READ and GC-DUMP allow high-speed I/O of MDL data structures, while preserving sharing within the structures. This is another feature that existed in old MDL, but only appeared in MIM this year.

- Users can now build arbitrary MDL objects on the stack; thus, temporary data structures can be used freely without invoking the garbage collector, and without writing complicated recycling schemes. Old MDL had a much more restricted form of this; the implementation of this feature was made much easier because very little of it had to be done in assembly language.

- On Unix, we developed an interface to pipes, which allow easy communication between processes. This, in combination with a package for running inferior forks, allowed an undergraduate to develop an interface to the INGRES DBMS for use by the planning system.

- Since most of our Vax-11/750s have small disks, MIM, MIMC, and the Vax open-compiler were moved to a file server. In addition to saving space, this makes it possible to perform updates on all machines in one operation.

Further performance enhancements were made to MIM.

- The full-copy garbage collector was modified, in conjunction with the two open compilers, such that it is essentially hand-coded in critical areas. Since the "hand-coding" was done by the compilers, we were able to cause the same code to be generated for PURIFY, thus gaining the same speed advantage there at no cost.

- READ and PRINT were made much faster by careful rewriting of some critical sections, and by reduction of subroutine-calling overhead where possible.

- The EVAL/APPLY section of the interpreter also became much faster after some recoding.

Finally, MIM was made better able to survive in the real world. The Unix version

was moved from Berkeley Unix 4.1 to 4.2, which involved some major changes in system-dependent code. In conjunction with this, MIM was finally taught about the structure of virtual memory under Berkeley Unix. It now uses as little as it can; but it can get the maximum amount available when needed. This is particularly important on the Vax-11/750s with small disks that many of our users do development on. Reliability continued to improve in all areas, as the continued improvement in speed and features led to more extensive use of the system.

4. PLANNING SYSTEM

Work on the planning system PLAID has continued during the last year. Progress has been divided between improvements and enhancements of existing features on the one hand, and expansion of the system into the area of worksheet sharing and communication on the other.

Refinements and additions to many existing features of PLAID have been made, motivated by comments from local users of the system and observation of its behavior.

Major expansion of the system has been performed to enable sharing of worksheets among the members of a group of cooperating users. The user of the system is given detailed control over his own usage of other users' worksheets, and also of other users' usage of his own worksheets. The usage controls have enough flexibility that either very free or very controlled worksheet sharing is possible.

4.1. Incremental Improvements

In the area of incremental improvements, we will touch briefly on some of the changes made over the last year.

Major internal changes have been made to the user interface and command interpreter. It is expected that the current command interpreter is the final redesign. The new interpreter makes the definition of new commands and new objects easier, and also makes the Help and Options facilities available system-wide. In the new system, items are deposited in a list of relevant help or options information which is ultimately processed and displayed by the command interpreter. Consequently, only the command interpreter needs to know the details of how such information is finally displayed. One pleasant side-effect of the new regime was that Help and Option window placement expertise could be localized, and so in current releases, such windows rarely obscure useful information.

The Graph facility has been expanded to allow incremental addition and updating of such items as titles, labels on graph axes, and so on. The data being graphed may

be updated incrementally as well. For example, the number of bars in a bar graph may be changed without respecifying the entire graph. Finally, a Stacked-bar-graph style was added to the existing graph styles of XY, Polar, Bar, and Pie-chart.

Many status commands and the data to support them were added. Some of these additions, such as a "Show worksheets" command, were motivated by the implementation of the worksheet sharing facility.

PLAID is now able to understand the concept of time. Dates and times are now legitimate data and may be manipulated by equations. The user may now have time dependent information in worksheets. It is possible to set the current-time (from the point of view of a worksheet) as well, and thus explore the time-dependencies of a model.

As the system expanded, it became apparent that setting and resetting various options each time it was started was too tedious, and so a "tailor file" facility was added. The tailor file stores the variable settings for all the system-wide variables.

4.2. Worksheets as Procedures

PLAID has the ability to treat a worksheet as a procedure. The motivation is that once a model has been constructed, it is both tedious and inefficient to copy the entire model into every worksheet that wishes to employ it. A revenue forecasting model might be driven by four or five independent variables. It is in effect a procedure which takes these variables as arguments and produces output values. In a sensitivity analysis, one might want to try several different values for each variable. In normal circumstances, one laboriously changes the variables, then copies the results into ones worksheet. Some spreadsheet programs allow this to be done in an automated way.

In PLAID, the worksheet is called as a procedure. It takes as arguments a list of pairs: cells in the worksheet and the values you want in them. The old values are pushed, the worksheet is reevaluated, values are pulled out, and the old values are restored. The syntax looks like a "call by name". For example:

```
@MORTGAGE(PRICE:100000,INTEREST:13.5,TERM:30,PAYMENT)
```

would evaluate the mortgage worksheet with the named slots' values replaced, and return the contents of the PAYMENT slot.

4.3. INGRES Interface

During the past year an experimental database interface module was written (by Baek). This module allows records to be read or written from the INGRES database system into a worksheet. For example, a record would be read into rows of a worksheet, and the subrecord names retrieved as the column headers. Further work is expected to be done in this area. For example, it would be useful to be able to define a worksheet as containing the records which meet some selection criteria, and have its contents automatically updated if any of the criteria change.

4.4. Constraints and Undo

One way of seeing a worksheet is as a system of equations which is resolved each time it is changed. Given this model, it would useful to be able to place constraints on the values of these equations, and to undo changes that cause the constraints to be violated.

PLAID allows the user to associate a constraint with any cell of the worksheet. This constraint is in the form of a predicate which is reevaluated whenever the value of the cell changes. For example, the cell A21 might contain the equation

```
>A21 EXPENDITURES=@SUM(A1...A20)
```

which makes it, perhaps, the total of expenditures in a budget. In addition the cell can contain a "Check" equation:

```
Check EXPENDITURES =< 20000
```

which states that if the sum is greater than $20,000, it violates a constraint. Better still might be

```
Check EXPENDITURES =< REVENUES
```

where REVENUES is another cell containing total revenues.

It was in conjunction with constraint checking that the Undo facility was introduced to PLAID. Whenever a cell is changed, the old value is saved on an undo list. The Undo command restores the old cell contents to the state at the beginning of the previous command. In the above example, use of Undo would remove the entry that caused the sum to overflow the constraint.

It is desirable avoid the sort of problem that can arise when the user is shuffling many values around in a summed list whose total is constrained. For example, when the user is juggling the sizes of various line items in a budget, the sum would almost inevitably violate the constraints now and then. In PLAID, this problem can be avoided by manually turning off continuous evaluation and continuous constraint checking, and then either turning them on again or explicitly evaluating using the "Evaluate" key, which performs evaluation and checking.

The Undo facility is as yet unable to perform "general" Undos, because not all changes to worksheets or the PLAID environment are performed by changing cells in the worksheet. The mechanism will support a more general implementation, but it has not yet been installed.

4.5. Communication

PLAID allows users to get information from other users' worksheets, subject to the restrictions imposed by the protection scheme described in the next section. The goal of the communication scheme was to enable you to use someone else's worksheet without being unduly surprised and discomfited when he makes major changes to it. By the same token, it should not get in your way if you are using one of your own worksheets or don't care about changes because you expect them (when you are using a rapidly changing database, for example).

There are two "communications" commands, Use and Update. The former establishes a link between the current worksheet and some other, and the latter updates changed values taken from the other worksheet.

The full specification of Use is:

Use *worksheet version when-to-update*

where the latter two arguments have defaults.

Possible *versions* are *newest* and *release* (meaning newest "official" release). *Newest* version means to look for the file with the same first name as the *worksheet* argument and extension ".WS". *Release* means to look instead for an extension of ".WSR". Directories in the worksheet search path are examined in order. The search path may be specified explicitly by the user ("Set search-path") or implicitly; it is normally the set of all directories from which worksheets have been loaded.

When-to-update specifies the circumstances under which the worksheet is examined for changed cells. Possible arguments are *always*, *when-told* and *never*. *Always* means that whenever the using worksheet is loaded, the used worksheet is also loaded. This setting provides no insurance against changes in the used worksheet. On the other hand, there is no overhead in storing old values from the used worksheet. *Never* means that the values in the used worksheet are copied and never updated. In effect, you get a never-changing snapshot of the used worksheet. *When-told* is a middle ground. A snapshot of the used worksheet is kept in the using worksheet, but it is updated when the user specifically asks. Thus, at some cost in overhead, you can be insulated against surprises.

Issuing a new Use command changes the mode of usage for the worksheet being used, so (for example) *never* mode is not irrevocable.

The second part of the sharing facility is the Update command, which gives the user control over updating of values in worksheets he is using.

Update *worksheet method*

The *method* argument tells how the copy of *worksheet* used by the current worksheet should be updated. There are three methods available. "Full" means that all changed values from the worksheet are accepted without question or interaction. "Cut" means that all new values are rejected and the copy of the worksheet is in effect frozen. This is said to cut the connection between the two worksheets. Finally, "Ask" means that each change will be displayed to the user, and he will be given the opportunity to pass on each change individually. For each reference, the user has the option of accepting the change or cutting the link.

One result of this scheme is that a worksheet may only directly reference one version of another worksheet at a time. It may contain cut references from earlier versions, but active references to only one version.

4.6. Protection

PLAID provides three levels of protection, which may be applied to individual users or groups of users, and may refer to worksheets, regions or even individual cells.

A PLAID entity (worksheet, region, or cell) is by default inaccessible to users other than its owner. Read access to the values, read access to the model (the system of equations) and write access may be permitted by the owner. When a worksheet is used, the accessibility of the cells is set by looking up the access for the user from an access-control list contained in the worksheet. In effect, the worksheet is sanitized. Cells that the user cannot see look like they are empty. Attempting to write in a write-protected cell causes an error.

PLAID allows access to be controlled on the worksheet level if desired. It is also possible to permit or refuse access to regions or cells as well. Access may be permitted in smaller areas within large refused areas. For example, the owner may say:

```
Permit (user) TAA (access) read (region) A1.B18
Refuse (user) TAA (region) B12
```

The above grants the user TAA read access to all but one cell of a rectangular region.

In general, worksheets are files in an operating system (TOPS-20 or Unix), and as such must be readable to be Used. In the initial implementation of protection, no attempt has been made to resolve the contradictions inherent in "protecting"

something that can itself be read and potentially edited "out-of-band" of the PLAID environment. In the final implementation, worksheets may be accessed through some other mechanism that allows them to be read-protected on a file by file basis. Some analogue of an access-control job or "message vault" will be used.

4.7. PLAID and MDL

PLAID was converted during 1982-1983 to run on Unix in machine-independent MDL. During the last year, MDL under Unix became sufficiently well developed and complete to enable PLAID to move to a Vax-11/780 running Unix (Berkeley 4.2) as its primary development machine. Until then, modules were debugged and compiled on Tops-20 and then cross-compiled for Vax Unix and the object files moved to Unix. Now the situation is reversed.

5. GRAPHICAL PROGRAMMING AND MONITORING OF PROGRAM BEHAVIOR

The purposes of this project are (1) to discover some good ways to use graphics to facilitate the preparation and understanding of computer programs and (2) to develop a system for graphical programming and graphical monitoring of the interpretation of programs. The project is being carried out by a professor and a group of undergraduate students. They have been at work on it for about 20 months, and they have some impressions and the beginnings of a system. In order to summarize, it will be best to begin with the system, which represents just one of several approaches that the group has explored. The system is written in MDL, a language similar to LISP. LISPers can read MDL if they consider angle brackets to be parentheses and remember that you have to put a period in front of a symbol to represent the local value of the symbol -- and a very few other such things.

Thus far, the system deals only with small program components. The work is just at the point of combining small components to form larger ones and combining larger ones to form programs. Let us take as an example the preparation of a program component that determines whether or not the greatest of three numbers (the *MAX* of them) is or is not greater than the literal integer 100. In MDL, what the programmer wants to come out with is the function that is created by evaluating

```
<DEFINE NAME (X Y Z) <G? <MAX .X .Y .Z> 100>>
```

We have not been able to find a graphical approach that works faster or uses less space than simply writing that definition. Here is what we, as a graphical programmer, do to achieve essentially that result:

We begin with a running system, which includes a display screen and mouse. On

the display is a large square with a few pictograms in it. The square represents a general (i.e., as yet unspecified) MDL object. Our task is to develop or differentiate it, to specify it. Also on the display screen are three menus. One presents about eight alternative things to do. Another presents an array of about 70 basic MDL operators. And the third presents an array of about 30 MDL data types and classes. In this example, we shall deal with only a few of those menu items.

In the square are a few pictograms that constitute a menu. The most frequently used menu item, which we select with our mouse at the outset, directs the differentiation of the general MDL object into an applier, i.e., a form that applies an operator to operands. (For the sake of uniformity, "applies" is interpreted in a very general sense, and an applier can deal with special constructs (such as COND) as well as things that are unambiguously operators.) When we select the applier pictogram, the square gets a vertical line down its middle, to separate the right-hand *operator* half from the left-hand *operand* half, and we are asked to specify the operator. We select *G?* from the operator menu with our mouse. A box containing *G?* appears on the left-hand side and two boxes for operands appear on the right-hand side. In the latter two boxes are menu pictograms with which we can select whether to continue the differentiation with appliers or to terminate it by specifying data objects.

We want the top box on the right-hand side to represent the maximum of the three variables, so we select its applier pictogram. The system asks us to specify the operator, and we select *MAX* from the operator menu. Since *MAX* can take any number of operands, the system asks us how many. We say "3". Thereupon, a *MAX* operator box and three operand boxes fit themselves into the top right-hand rectangle. At this point, we can work on one of the three *MAX* operand boxes, or we can deal with *G?*'s need for a second operand. Let us go to the top *MAX* operand box.

From the top *MAX* operand box, we select the datum pictogram. The system wants to know whether we want to specify a datum value literally or to specify a variable or constant of the kind that would, in symbolic programming, be specified by name. (In graphical programming, the latter kind has a place, but not a name.) We indicate, by clicking a mouse button, the latter kind. Then we point to where we want it to reside. And then we point to a type or class in the data type and class menu. Let us select the integer pictogram. The system says that the standard illustrative value for an integer is 10000 and gives us a chance to supply an alternative value. (This value is not going to be part of the program, but it will go into a data base of data values for testing purposes.) We accept the standard value, and the system draws an integer pictogram in the place we selected.

Following the same procedure, we deal with the other two *MAX* operand boxes.

Then we turn to the second operand box of G?. This time we tell the system we want to specify a datum value literally. We type "100", and it replaces the number pictogram in the operand box. That completes the specification, which, despite the length of this description required only a few mouse selections and the typing of four numbers.

To see the function we have prepared, we select the function icon in the things-to-do menu, and the system displays the definition form for the new function, using the function name *TEST*:

```
<DEFINE TEST(i.1 i.2 i.3)<G?<MAX .i.1 .i.2 .i.3>100>>
```

and gives us an opportunity to substitute a name of our choosing. The system cannot yet, but soon will be able to, place a miniaturized copy of the graphical pattern we created into yet another menu area so that we can use it as a component in building larger units of program.

We can test the newly created function with the illustrative data we supplied. We accepted the standard value of 10000 for the first argument to *MAX*. Suppose that we specified 20000 and 30000 for the other two arguments. Then, if we select the *EVAL* icon from the things-to-do menu, the system would tell us that, with the specified data values, the result is *T*, which means *TRUE*.

There is a bit more to the system than just illustrated, but that may provide an idea of what is aimed at.

At present, we are beginning to work on (1) getting the system to deal with assemblies of components and (2) the incorporation of monitoring features [1]. The latter will let the programmer try out what he has done thus far and watch its graphical representation behave -- i.e., watch the unfolding of the interpretation process as the function resulting from the definition is applied to specified arguments.

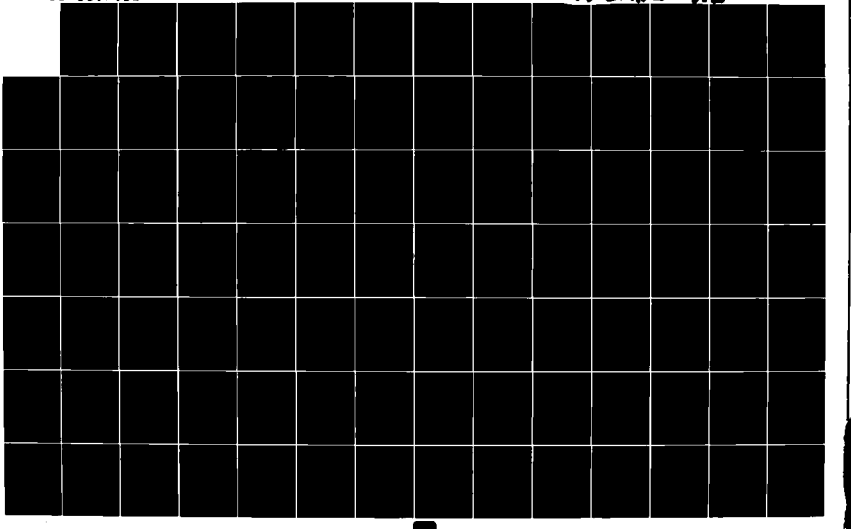
Finally, a few brief statements about problems encountered and about the place of graphics in programming:

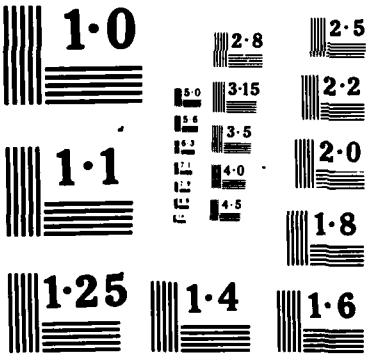
1) The most distressing problem is caused by the disproportion between the great complexity of a serious program and the small size of the display screen. With a very large sized display and a zooming facility, we think it would be possible to make a graphical programming and monitoring system that would be helpful to programmers working on large programs. With anything like 1024 x 768 pixels, it is necessary to have a lot of information off-screen where it is not really very graphic.

2) We have had trouble making the system run fast enough to be

AD-A158 299 LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 21 JULY 3/4
83-JUNE 84(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB
FOR COMPUTER SCIENCE M DERTOUZOS JUN 84

UNCLASSIFIED F/G 9/2 NL




1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
5·6

supportive instead of frustrating. What makes it slow, mainly, is calculation and not graphic display. Even though the images are thus far mainly just boxes inside boxes, it may be necessary to substitute storage of prepared images for calculation and recalculation of images on the fly.

3) There are several hundred basic programming constructs in a typical programming language, and nobody wants to learn several hundred new icons or pictograms before trying out a new approach to programming that may or may not help much. We have fallen back upon mnemonic abbreviations of names for the basic MDL operators. The number of data types is smaller, and it seems right to use icons to represent data types. The composition of pictographic representations of classes of data types from pictographic representations of individual data types seems like a good research problem [2].

4) There will be several thousand functions in a good program library, and it does not seem probable that, in a western language context in which people do not already know a lot of pictograms, a way will be found to represent library functions by icons.

5) It is natural, in developing a graphical programming system, to let the system watch over what the programmer does and refuse to let him make syntactic (and perhaps certain semantic) mistakes. This can be done also in symbolic programming contexts, as in the Programmer's Apprentice and in syntax-checking program-text editors, but it seems especially appropriate in a graphical system in which much of what is done is selection from among system-proffered alternatives.

6) Graphical representations of programs tend to portray the structure of programs better than do symbolic representations, even if the latter exploit indentation to achieve a quasi-graphical effect. A goal is to have a graphical display that, pinned up on a wall, will represent a large program effectively -- that when viewed from a distance will reveal the high-level structure and when viewed from close-up will explain the details.

7) Even if it proves too difficult to deal with the internal workings of large programs graphically, graphics may still be very useful in marshaling programs and applying them to data. Each function can show pictorially what data it needs to work on, and each data set can show pictorially what kinds of operators can operate successfully upon it. High-level control of information processing is mainly a matter of connecting

together selected operators and operands. Graphics promises to make it just a matter of pointing to pairs of things to connect, and finding that the members of a pair fit together neatly if appropriate for each other and refuse to join if inappropriate.

References

1. Naylor, C. M. "Graphically Representing MDL Programs as Trees," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

2. Shivanandan, J. U. "The Representation of Programmer-defined Data Types in Graphical Programming," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

Publications

1. Licklider, J.C.R. "Some Problems in Information Policy," in *Information Science in Action: System Design*, Debons, A. and Larson, A. (eds.), Boston, The Hague, Lancaster, Martinus Nijhoff in Cooperation with NATO Scientific Affairs Division, 1983, Volume II, 640.

2. Licklider, J.C.R. "The Future of Electronic Learning," in *The Future of Electronic Learning*, White, M. (ed.), Hillsdale/London, Lawrence Erlbaum Associates, 1983, Chap. 7, 71-85.

3. Licklider, J.C.R. "Information Technology, Education, and the American Future," in *Intelligent Schoolhouse: Readings on Computers and Learning*, Peterson, D. (ed.), Reston, Reston Publishing Co., Inc., 1984, 271-288.

Theses Completed

1. Naylor, C. M. "Graphically Representing MDL Programs as Trees," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

2. Shivanandan, J. U. "The Representation of Programmer-defined Data Types in Graphical Programming," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

3. Sun, R. "XDM: An Approach for Specifying Semantic Integrity Constraints in a Data Model," S.B. and S.M. theses, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, November 1983.

Theses in Progress

1. Buffo, D. "A Rule-based System for Financial Planning," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1984.

2. Lee, D. "Software Portability Relative to Data Base Management Systems' Software," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1984.

Talks

1. Licklider, J.C.R. "The Futures of Computers in Education," Lexington Public Schools, Lexington, MA, January 1984.

2. Licklider, J.C.R. "User Friendliness, Ease of Learning, Ease of Use," Bolt, Beranek, and Newman, Cambridge, MA, February, 1984.

REAL TIME SYSTEMS

Academic Staff

M. L. Dertouzos
R. H. Halstead
C. J. Terman
S. A. Ward, Group Leader
R. E. Zippel

Research Staff

D. Goddeau

GRADUATE STUDENTS

J. Arnold
P. Cosway
J. Eisen
R. Finn
S. Gray
A. Masterson
J. Mercier
P. Nuth
D. Nussbuam
R. Osborne
G. Pratt
J. Sieber
T. Sterling
M. St. Pierre
S. Seda
T. Teixeira
B. Williams

Undergraduate Students

V. Ali
P. Antaki
A. Bedonian
E. Broadway
J. Chang
A. Gilbert
P. Hood
R. Jenaaez
A. Kohli
R. Kukura
W. Liske
A. Litman
T. Maloney
A. Manzoor
H. Minsky
J. Pezaris
J. Noakes
S. Raman
D. Robinow
E. Seidman
W. Shang
A. Sieving
J. Vance
M. Wade
J. Wolfe
R. Zabih

SUPPORT STAFF

J. Hoppe
L. Kenen
E. Tervo
S. Thomas

Visitors

G.C. Clark
T. L. Goblick
M. Kitahara
J. Powell
M. Shen
G. Zientara

1. INTRODUCTION

During the 1983-4 calendar year there has been continuing progress on three major project areas, as well as several seed efforts which may lead to project-level research commitments in the coming year. The continuing work is in the areas of (i) Personal Workstations, (ii) Multiprocessor architectures, and (iii) VLSI development tools; subsequent sections of this report are devoted to each of these.

2. PERSONAL WORKSTATIONS

The NU personal computer and its companion network-oriented operating system, TRIX, have been active RTS projects for the past six years. Despite technical progress, the history of technology transfer aspects of the project has been generally discouraging during most of this period. We happily report a marked improvement during the past year.

2.1. NU Deployment

During the 83-84 academic year, Texas Instruments began production of a first-generation Nu machine based on technology developed at RTS and delivered, in accordance with the TI/MIT agreement, thirty production machines to our Laboratory. Each of these machines constitutes a competent and reasonably well-equipped personal computer comprising Motorola 68010 processor, 2Mb RAM, 84Mb disk, Ethernet, 8088 peripheral and diagnostic processor, and 1024-line bitmapped graphics; as delivered, it runs a TI-supported variant of the RTS 68000 UNIX port.

These thirty machines are being deployed within RTS to serve as a staple computing resource in support of our various research needs. To this end, we have recently ported many software packages on which RTS depends heavily to the TI Nu's; these include CAD tools, document production software, MultiLisp, TRIX (see below), and a number of graphics and other utilities. In addition, the TI Nus provide the base hardware environment for several experimental processors under development, including a waveform synthesizer (Miller, Robinow) and a novel dataflow processor (Antaki).

At this writing, about a dozen of the Nus have been installed and are being used routinely as tools for other research; we expect to have completed the remaining installations by September.

2.2. NuBus Developments

The architectural backbone of the Nu is the NuBus, a flexible 32-bit backplane bus designed to provide relatively technology-independent coherence across a wide range of system configurations and price/performance criteria. The original NuBus development was stimulated, to a great extent, by the conspicuous absence of an appropriate 32-bit bus standard within the computer industry, and a continuing goal of the Nu project has been the promotion of such a standard. From the earliest days of the Nu project, we have interacted with the IEEE P896 standards committee where NuBus ideas have been clearly influential if not adopted per se. While the IEEE committee has been a forum for valuable technical debate, it has also been plagued by strong proprietary interests which have made progress toward concrete standards painfully slow.

We have been delighted that Texas Instruments, our current industrial partners, shares our interest in promoting the NuBus as a standard and has devoted considerable manpower and other resources to that end. They have interested several additional manufacturers in the production of NuBus-based equipment, and a recent modification of the MIT-TI agreement allows NuBus sublicenses at very nominal royalties (shared between TI and MIT). Moreover, their active and persistent participation in the IEEE standardization effort (jointly with our own) has substantially furthered the influence of NuBus ideas and may result in the establishment of the NuBus itself as an IEEE standard. Guarded optimism on the latter front stems from a non-binding vote taken at a May committee meeting (held at LCS) to recommend the adoption of the NuBus as a standard; this action remains to be ratified by a mail vote of the entire committee membership.

2.3. TRIX Development

The progress report for last year briefly describes the architecture of TRIX 1.0, an operating system whose uniform communication semantics induce unique power and coherence on a network of interconnected personal computers. In last year's report we mentioned that two early prototype implementations had been constructed (single- and multi-processor versions), and that we were considering alternative targets for a more "serious" implementation which would realistically provide for the needs of a non-trivial user community. The goal for this latter implementation is the evaluation of TRIX in the environment for which it is intended, via the use of TRIX communications to serve higher-level user and application needs.

During the past year we selected the TI Nus as the target for the new TRIX implementation, primarily because of their availability within RTS, and have brought the new TRIX implementation to an operational status (Sieber, Goddeau). The current implementation runs the major UNIX packages and supports software

development (including the maintenance of TRIX itself). In addition to UNIX compatibility software, it provides transparent TRIX-oriented network communications and a primitive window system. While it clearly requires further development, it currently offers an adequate environment for the development of software by a community of benign users, and RTS software activities are gradually moving from UNIX to TRIX. We hope to continue to support TRIX development and encourage this migration of the user community, allowing us to accumulate substantive experience with realistic TRIX use during the next year.

3. MULTIPROCESSOR ARCHITECTURES

The multiprocessor architecture work has seen continued progress on many fronts. We are finally leaving a long gestation period devoted mainly to tool-building and stand ready to spend much more time on the basic research. Our work has been in roughly three areas: hardware construction, software construction, and performance analysis and modeling. Additionally, we have continued our research collaboration with Harris Corp., which is beginning to bear fruit.

3.1. Hardware Construction

A major accomplishment during this period has been the completion of debugging of the RingBus Interface Board, or RIB, a central component of the Concert multiprocessor being built to serve as a tool in our research (Osborne, Gray, Robinow, Halstead). Within the next month, several more copies of the RIB will be produced, and these will allow us to build increasingly larger subsets of the Concert multiprocessor, up to the planned 32-processor size. Currently, there are four operational subsections of the Concert machine, which have allowed substantial software development and experimentation to take place, even though the units are not linked to each other via RIB's, as will soon be the case. The largest of these units, the "Megatron," contains 8 MC68000 processors and 5.5 megabytes of memory, and has been operating reliably for a few months.

The Dynamic State Display (DSD) prototype, begun last year, has been completed (Nuth). The DSD is capable of measuring and displaying in real time various performance parameters of the Concert multiprocessor, such as the loading and access delays of the various buses in the system. Use of the DSD has already given us valuable insight into the performance of various programs, and has confirmed that, for most of the programs tested, the bandwidth of the Concert communications substrate has been adequate.

Another piece of performance-monitoring hardware that has been constructed is the Spektron (Becker, Sterling). The Spektron measures the distribution of interarrival times between specified events in Concert, and will help us to evaluate proposed probabilistic models of the behavior of programs on Concert.

The Concert Display Driver, begun during the Spring of 1983, has been completed (Osler). A direct-memory-access front end for loading display lists into the Display Driver has been built and debugged (Nussbaum). The combination of these two pieces of hardware is capable of displaying continuously changing gray-scale images on a raster-scan CRT, based on changing display lists computed in real time by multiprocessor programs running on Concert. Conversion of the display driver from monochrome to full-color operation is planned.

3.2. Software Construction

Software for the Concert multiprocessor falls into main categories: support and system software, and programs of direct research interest. Support software development has continued: notable items include a TRIX-like package for supporting a message-passing style of computation (Halstead), an improved debugger (Litman), a UNIX-like file system usable in a multiprocessor environment (Kitahara, Jenez), an editor, using the Display Driver, for constructing descriptions of three- dimensional objects (Broadway), and myriad programs for gathering useful statistics using the DSD (Nuth, Chang, Osborne).

Development of the MultiLisp language for parallel computation has continued (Halstead, Loaiza). The principal conceptual addition to MultiLisp during the past year has been the futures mechanism. A future is a token, or place-holder, for a value that has not yet been computed. When the value is computed, it replaces the future. In the meantime, if any computation really needs to know the value of the future, the computation is suspended until the value becomes available. Many computations that manipulate a value, for example, to store it in a data structure, do not really need to examine the value, however. Futures provide an opportunity for extra concurrency in execution by allowing the computation of a value to proceed concurrently with operations that determine how the value will be used.

Development of MultiLisp has proceeded to the point where there are now over 3000 lines of working code written in MultiLisp, including the MultiLisp compiler itself, which has been written (using futures) to exploit some amount of parallelism, though not yet a great deal. Experiments with other, more highly parallel programs, using the Megatron, show speedups of virtually a factor of eight when eight processors are used. Measurements using the DSD show only about 20% of the Megatron's communication bandwidth in use during these experiments, which augurs well for the potential of MultiLisp when more processors are used.

The Parallel Control Flow (PCF) paradigm for parallel computing has undergone further development (Sterling). Most of the tools for processing PCF programs are now in place, including an assembler (Noakes) and a compiler from the language Simultaneous Pascal (Sieving). An implementation of PCF on the Concert multiprocessor is now in progress at Harris Corp.

3.3. Modeling

An analytical study of the performance of the Concert multiprocessor, using stochastic models of program behavior, is being undertaken by Osborne. A principal accomplishment to date has been the construction of a model for the RingBus communication substrate of Concert using Markovian decision theory. Among the interesting results is the fact that there is no single optimal strategy for granting access to the RingBus: the optimal strategy depends on the probability of different kinds of requests arising in the future.

3.4. Industrial Collaboration

The first full year of our collaboration with Harris Corp. has just been completed. Our partners at Harris have spent most of this period acquiring the necessary resources and educating themselves about parallel processing. However, several efforts have been initiated which should bear tangible fruit within the next year:

1) Harris has begun the design of the GCM, a device similar to the RingBus but based on a crossbar switch (with the attendant higher performance), which will be at the heart of a version of the Concert multiprocessor being constructed at Harris for their use;

2) Harris has committed to re-engineering the DSD for production in printed-circuit form, for their own use and ours; and

3) Harris personnel are already helping with the development of support software for Concert in several areas.

3.5. Conclusion

The Concert multiprocessor project is finally emerging from a long gestation period during which necessary resources were accumulated, necessary tools built, and a critical mass of personnel accumulated. The next year should see many exciting results that will deepen our insight into multiprocessor software and architecture, and prepare the way for the design of systems capable of a more ambitious degree of parallelism.

4. VLSI DESIGN TOOLS

During the past year two major RTS research efforts have addressed the problems of simulation (Terman) and integrated VLSI design system (Zippel). These projects are detailed below.

4.1. Simulation

One focus of our research work in the area of CAD for VLSI circuit design continues to be in the area of simulation. In particular, the work reported below aims to increase the capacity of current simulation tools through (1) the use of multiprocessors for more computational horsepower, and (2) a reduction of simulation overhead by compiling in-line code where possible before starting simulation. Each of these areas is discussed in more detail below.

In previous years, we have reported on RSIM (Terman), an event-driven logic-level simulator for integrated circuit designs incorporating a very efficient linear network model. RSIM has seen increasing use outside the MIT community, and is now the major logic-level simulator at many university and industrial locations. Versions of this simulator are included by Berkeley and the University of Washington as part of their CAD tool distributions. Feedback from new users has led to improvements in the the basic timing calculations, the user interface, and the portability of the implementation. A major revision of RSIM incorporating these improvements is scheduled for release in August 1984.

Parallel Simulation of Digital LSI Circuits: Multiprocessor systems are becoming available at many organizations involved in LSI circuit design. These systems range from mainframes and workstations loosely coupled by local-area networks, to dedicated systems of processors tightly coupled through shared memory. The goal of the work described in this section is to develop logic level simulation algorithms which can take advantage of a multiprocessor system to improve simulation speed and capacity. This work complements the investigation now underway elsewhere (most notably Berkeley) of multiprocessor implementations of lower level circuit analysis programs. Taken together, these efforts hold the promise of a multi-level simulation capability with truly remarkable performance.

The initial goal of our work (Terman, Arnold) is to build a multiprocessor implementation of the RSIM algorithms. Given the wide variety of multiprocessor systems available, special care is being taken to ensure that the implementation uses a simple (and hopefully universal) set of interprocess communication primitives. Work on the project has just recently gotten underway; currently, there are three identifiable subprojects:

1) Conversion of the RSIM algorithms to use only fixed point integer arithmetic (Arnold). Since many of the multiprocessors incorporate CPUs (e.g., the MC68000) that do not support floating point calculations directly, this important step ensures that no great performance loss is incurred simply because of moving out of the VAX environment. The 32-bit arithmetic provided by most CPUs provides over 9 decades of dynamic range - enough to represent the electrical parameters in which we are interested.

References

1. Batali J. and Hartheimer, A. "The Design Procedure Language Manual," MIT Artificial Intelligent Laboratory, Cambridge, MA, 1980.

2. Horowitz, M.H., Penfield, P. and Rubinstein J. "Signal Delay in RC Tree Networks," *IEEE Trans. on CAD*, CAD-2:3, (1983), 202-211.

3. Katz R.H. "Managing the Chip Design Database," *Computer*, 16:12, (1983) 26-36.

4. Minsky, H. "A SCHEMA Interface for Signal Specification," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

5. Moon D.A. and Weinreb D.L. "Lisp Machine Manual," MIT Artificial Intelligence Laboratory, Cambridge, MA, 1982.

6. Rose M. "A Datapath Generator," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1982.

7. Williams, B. "Qualitative Analysis of MOS Circuits," *Journal of Artificial Intelligence*, submitted for publication.

8. Zippel R. "Capsules," *SIGPLAN Bulletin*, 18:6, 166-169, 1983.

Publications

1. Dertouzos, M.L. "Software Technology and Computer Architectures by 2000 AD," *Proceedings of the National Academy of Engineering, 19th Annual Meeting*, Washington, DC, November 1983, to appear.

2. Dertouzos, M.L., Arvind and R. Iannucci, "A Multiprocessor Emulation Facility," MIT/LCS/TR-302, MIT Laboratory for Computer Science, Cambridge, MA, October 1983.

3. Dertouzos, M.L. and Moses, J. "Coherence," MIT, School of Engineering, Project Athena, Cambridge, MA, September 1983.

4. Terman, C.J. "RSIM - A Logic-Level Timing Simulator," *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers*, Port Chester, NY, November 1983, 437-440.

machine 'learns' to recognize inputs reliably for that individual. The program emulates the behavior of a network of simple finite state machines, which first extract a set of basic features in the form of a string of symbols from a discrete, finite alphabet for each input character, and then compress the representation to yield a string which uniquely identifies the input. The compression can be considered to be an abstraction since this step allows recognition of characters whose input features do not exactly match that from which the compressed string was derived. This step enormously compresses the amount of data that must be stored and searched to perform the recognition task reliably. Although this research on hand-printed character recognition may someday lead to a useful end product, the primary purpose of this effort is to study and experiment with various aspects of the fundamental machine-learning/pattern-recognition model used here which is based on symbol processing rather than signal processing.

Capsule system is a partially successful attempt to provide mechanism through which the programmer can truly express what the program is intended to do.

5. STUDIES IN MACHINE LEARNING

Exploration of bases for nonstandard computer architectures (Goblick, Ward) has led to an interest in machine learning, i.e., systems which can improve their performance with experience when the desired system responses are presented together with the inputs. A universal goal among researchers in machine learning seems to be to emulate the powerful learning mechanisms of biological organisms, especially the higher forms like man. This image leads to the study of computing machines built using a large number of simple computing elements, each only moderately reliable, but all intricately interconnected to result in excellent overall system performance and reliability. One can imagine VLSI chips containing large arrays of identical, simple, neuron-like computing elements which are not programmed in the usual way, but are 'trained' by example to establish the proper interconnections between elements to perform specific tasks. One example of this approach is that of an associative memory to recognize patterns represented as vectors whose coefficients correspond to different features. The associative memory can be constructed out of a large array of identical, simple processors to compute the dot product of an input vector with each of the set of stored vectors in the memory.

While this image is appealing to learning theorists, it is tantamount to putting the cart before the horse by asking, "What sort of interesting learning tasks can be performed by machines composed of arrays consisting of certain kinds of computing elements?" The problem is that the theory of computation has little to say about models of computation that are complex enough to represent machines composed of such 'malleable' arrays of processing elements. Another approach is to first devise algorithms which perform a certain learning/recognition task adequately, and then consider separately the implementation of these algorithms in hardware and/or software, taking into account the constraints of available circuit and chip technology, machine architecture and software. The advantage of this approach is that devising working algorithms can be done independently of the technology used to implement the final solution. The disadvantage is that the algorithms may fail to meet desirable implementation criteria, such as the exploitation of massive parallelism.

To obtain some concrete experience in machine learning, a model was formulated and programmed for the computer recognition of hand-printed English characters (letters and numerals). This program, running on a T.I. Nu Machine, is 'trained' by an individual user who enters hand-printed characters via the mouse, together with the desired machine response for that character (entered via the keyboard), until the

merely moves the mechanism we are discussing into the compiler. This is the fundamental difference between the Lisp Machine's flavor system which takes the object oriented viewpoint, and CLU which is function oriented.

By an object we will mean something to which a message can be sent. This will result in one (or more values) being returned and the internal state of the object being changed. The action caused when a message is sent to an object is called an operation. A message is a string used to name some operation. It contains no internal structure. The piece of code executed when a message is sent to an object is called a method. Objects may also contain internal state which is kept in instance variables that may be referenced by the methods.

Every object belongs to a class of equivalent objects that have the same methods and the same set of instance variables. This equivalence class is called a collage. Objects are created by calling the function MAKE-OBJECT on a collage. The methods are actually part of the collage, so as operations are added to the collage, the instances of the collage also acquire them.

The specification for how a method is to be constructed is kept in a structure called a capsule. When a capsule is added to a collage, the code within the capsule is incorporated in one or more of the methods of the collage. Capsules also contain information describing what their pieces of code expect of the collage to which they are added (what operations and instance variables there are, for instance).

When the user creates a collage, he or she specifies a protocol that the instances of the collage must satisfy. The Capsule system is then responsible for determining which capsules provide the fragments of the protocol desired and which also fit together. It then combines these capsules to form the desired collage. Thus in the capsule system, the user specifies what the desired functionality of the objects to be created and the system decides on the implementation. This greatly simplifies the programmers work since there are far fewer concepts than implementations of concepts, and also because it allows different programmers to build on each others work far more.

When the capsule system was used to describe a portion of the stream code used in the Lisp Machine, we noticed that the protocol specifications seemed more verbose than we would have liked. Closer examination revealed that many of the comments we had penciled into the version of the code that used flavors were being translated into protocols. This reinforces our impression that the capsule system is partially an attempt to force the programmer to make the code that is written more precise.

We feel that if the programmer makes this effort, the programming system will be in a much better position to aid in the development of large software systems. The

synthesis modules of Schema are expected to rely heavily on the database tools for organization and the analysis tools for validation and guidance. In addition, the results of the synthesis modules include behavioral descriptions to guide analysis and correction and as guidance for lower level synthesis modules. In this section we will describe the basic organization of some of the topological synthesis tools.

The lowest layers of the synthesis layer consist of many small specialized modules that convert higher level specifications into augmented designs. An example of this sort of module would be a buffer generator. The parameters that govern the design of a buffer are the input and output level, noise margins, power consumption, input and output capacitances, rise and fall times, etc. All of these parameters can be specified at different process corners. Many of these parameters, like nominal noise margins, are inherited from the overall design specification. Given a subset of these parameters, the buffer generator would attempt to chose a topology and size the devices to meet the specification. The synthesis module may need to perform simulations to make these decisions, for instance, to evaluate the impact of Miller capacitances in some circuit, or to adjust the device sizes using a more accurate device model than is used in the initial synthesis steps. In a very real sense this module would be an expert in buffer design.

A slightly higher level expert would convert register array specifications (word size, depth, cycling and power considerations) into a high performance, correctly margined register array. Other experts will deal with programmed logic array (PLA), general combinational logic and arithmetic logic unit design. The existence of the analysis modules, buffer designer and similar experts raises the semantic level at which these higher level synthesis modules are specified.

Combining a number of these tools, a datapath and microcode generators could be built. At the highest level, "silicon compilers" can be constructed based on the abstractions provided by the low level "experts." They are responsible for balancing the performance/power/noise considerations among the various components.

4.3. Capsules

The Capsule system is progressing quite well. A preliminary implementation has been completed by Ramin Zabih and it is being used to build a simple algebraic manipulation system this summer. We expect to refine the ideas and implementation significantly in the next few months and should have a more generally usable implementation available by the end of the calendar year.

In the capsule system we have assumed that all actions occur by sending messages to objects-this is the object oriented viewpoint. Taking the dual point of view, where objects are passive and the correct function is chosen by the compiler,

Analysis Layer: The second layer, the analysis layer, contains those tools that extract information from a design. In addition simulators like Spice and RSIM, design rule checkers and circuit extractors we include those modules that extract higher level semantic information from designs. This includes those tools that combine behavior information with simulation results to determine how a design component failed. These tools produce answers like: The pullup does not supply enough current, and the state machine was not reset at the beginning of each major cycle. Intelligent front ends to design rule checkers are also needed. Given the circuit topology and masks together, they can do a better job dealing with more complex design rules like current or voltage sensitive rules, and they can give a far better characterization of what went wrong. Rather than a dozen spacing rule violations, it should be possible to indicate that two cells are too closely spaced.

In addition, the knowledge contained in these tools may be used to constrain or determine details in the design. For instance, tools that convert DC currents and voltages to device sizes use the same models and similar organization as do tools that compute DC currents and voltages. In particular, certain components of generic circuit optimizers and retiming tools might be components of the analysis layer.

We have designed a DC analysis module that allows the designer to specify DC electrical parameters (voltages, currents, power, noise margins) which the system will use to deduce initial values for device sizes. In addition it will derive these DC parameters from a completely sized schematic (with the help of SPICE). Naturally, as the design is polished This has two advantages. First, topological specifications are insulated from process variations. And second, although the initial device sizes may be modified as the design is polished the Schema has available detailed electrical information about the intent of the designer which can be used to validate the final design.

This particular approach involved developing a small computer algebra package for manipulating systems of algebraic constraints, which was done. In order to check out the package a slightly simpler problem was addressed determining the voltage transfer function a linear system. The result is a package that allows the designer to graphically specify a circuit involving linear components: resistors, capacitors, inductors and operational amplifiers. The Schema can then compute the symbolic voltage transfer function and when asked will generate Bode and Nyquist plots for particular values of the parameters of the devices. This educational tool was put together with two weekends, and indication of the speed with which interesting projects can be done using the software organization used in Schema.

Synthesis Layer: Much of the interest in new style design tools has concentrated on synthesis modules since these actually create designs. The

the wires are either horizontal or vertical, when devices are moved it makes sure that wires don't end up with negative length, and it ensures that there are no dangling wires. When a module is deleted, the wires connected to it are also deleted. In addition, when running wire between two points, devices can be inserted into the wire with a single mouse button. These facilities allow the designer to sketch schematics quite quickly, in much the same manner as they do on paper and with much cleaner and reliable results. The designers who have made use of the system have generally been quite happy with it and consider it comparable to hand sketching in speed.

Regardless of how the schematic is modified, when the procedure is examined it always reflects the schematic. Ideally, everything that is expressible textually would be easily expressible using the graphics editor. Though this is difficult, we are attempting to get as close to this goal as possible. The Daedalus system provides a similar interface to DPL for mask design. Our collaborators at Harris, Corp. are currently adapting this mask design system using the tools in Schema in anticipation of its incorporation. This will also give us the opportunity to incorporate the routing and compaction work of Rivest and Leiserson into the mask level data structures and procedures. We expect to include structures like self routing channels and switch boxes in the basic library.

Finally, there must be some long term storage system. This storage system must be reliable, allow multiple designers to access the same design component simultaneously and yet provide interlock mechanisms to ensure designers do not interfere with each other. Katz [3] has discussed some of the issues involved in such a system. Our needs and constraints are slightly different since the active portion of the database is always kept in the virtual memory of a Lisp Machine, and among our concerns is the effective cooperation of possibly isolated designers. The long term storage system is a repository for all aspects of the design, schematics, simulation results, floorplans, layouts, architectural sketches, textual notes about the design, and anything else the designer might provide.

The current design for the long term storage system provides each designer with his or her own hierarchy of projects. A project is some major design, typically a chip or system. Projects contain schematics, circuits, simulation specifications and results, floor plans, mask descriptions and so on. In general, more than one designer works on a project at a time. The hierarchy of each designer contains an entry pointing to the project itself. Two approaches are being taken to make the hierarchy more permanent-a somewhat complex file server that takes care of the necessary locking and coordination mechanisms and the more fundamental approach of directly addressing the problem of stable storage of Lisp expressions. Though we are leaning towards the second solution as being the correct solution in the long run, the modified file system will also be implemented for pragmatic reasons. This work is being pursued by Jeff Eisen.

discrete, possibly non-numeric values, are used for digital signals and symbolic quantities are used for higher level signals. The waveform bounds developed by the techniques of Rubinstein, Penfield and Horowitz [2] can be represented by using open sets as the waveform values. Allowing open sets for time coordinates permits us to represent the qualitative quantities needed for behavioral description [7].

The topological, waveform and physical representations are all specified by a procedure as was done in DPL [1]. When this procedure is invoked it creates the appropriate data structure. Arguments can be passed to the procedure to control the creation of the data structure in any way desired. Since the procedure is merely a slightly stylized Lisp program, it can perform any computation desired to determine what sort of structure to build. Arbitrary (local) intelligence can be placed in this procedure to create relatively intelligent synthesis modules. For instance, Mark Rose has built a simple ALU generator module that chooses the carry propagation scheme based on the word length and speed requirements specified by the designer [6].

This technology is used not only to specify topological structures, but also to specify waveforms. In conjunction with a constraint system provided by Schema, this provides an extremely powerful way of specifying both the inputs and outputs of simulations, especially when difficult extremes of processing (called processing corners) must be taken into account. In combination with the waveform bounds and qualitative structures mentioned above, the procedural capabilities are an exceedingly powerful mechanism for describing the behavior of circuits.

The procedural specifications of circuits, waveforms and other structures can be tedious. A graphics system has been developed for the circuit language that allows this procedure to be specified by drawing the circuit's schematic. The schematic entry system itself is rather interesting. We feel that unless designers find it easier to enter schematics using Schema than sketching them on paper, they would shy away from the system and would delegate its use to draftsmen, wasting all of the design knowledge that Schema contains.

The basic interaction mechanism of the schematic entry system is similar to that used by Daedalus. The designer uses a three button mouse to for almost all actions. The function of the three buttons can be modified by depressing any combination of four shift keys. A black bar just under the schematic display is kept constantly up to date with the function of each button on the mouse. The eliminates the need for the designer to go to a menu to select an object or issue a command. Furthermore, the designer rarely moves his hands from this basic position: right hand on the mouse, left on the shift keys.

The schematic entry system understands a certain amount about how schematics are supposed to look. For instance, it moves devices if necessary to ensure that all

Thus far, the design of Schema has concentrated on the topological domain. This is the area that is least thoroughly explored and where we intend to apply our knowledge engineering techniques first. Most designs begin with topological specifications and elaborate them into physical specifications, so it is reasonable to begin here. Our colleagues at Harris Corp. are study the physical domain now, and we plan to bring the two pieces of Schema together at the end of this summer.

Schema is implemented in three basic layers. The database layer contains the data structures used to represent pieces of designs and design processes. It provides the primitive operations needed to manipulate and manage design objects in a consistent manner. The analysis layer contains those routines that extract information from a design object: simulators, design rule checkers, feature extraction tools, etc. Those tools that create design objects are contained in the synthesis layer. Each of these layers is discussed in the following subsections.

Database Layer: The data base layer provides basic tools for building a large variety of structures used in integrated circuit design. The basic data structures are built using the Lisp Machine's flavor system [5] way that anticipates the Capsule system [8]. This fundamental choice gives us more modularization flexibility than any other programming technique available. In particular, it allows us to build the knowledge in layers, out of small modules, without significant performance penalties. The flavor and Capsule systems collapse the layers automatically. We will be revamping the existing code using Capsules during the next year to eighteen months.

A number of different facilities are needed to build the data objects of a VLSI design system. Among them are prototypes, hierarchical structures, and incremental creation (creation on demand for very large structures). We have built a library of flavors that implement these facilities in a modular, reusable and efficient fashion. Along with a large collection of other utilities, we have used these flavors to build circuit and waveform representations [4], the directory structure used to manage projects and the graphics presentation structures for schematics. This philosophy of constructing small flavors (Capsules in the future) that implement basic software concepts is being followed throughout the design of the system, and has already allowed us to leverage our software efforts quite effectively. We expect the library of these facilities to increase dramatically as mask level structures are introduced.

The existing structures for circuits and waveforms are general enough to deal with circuits not only at the transistor level, but also at the gate level, at the register transfer level and at any higher level where the computational structure can be expressed as a graph of communicating modules. Waveforms with any value set can be represented. Real numbers are used to represent analog voltages and currents,

the analysis job. Additional software is required to determine the failure mechanisms from the simulation results using the behavioral specifications.

If the analysis routines uncover deviations from the specified behavior, correction modules are responsible for modifying the design (including the behavioral description) to come closer to the desired behavior. If the problems are sufficiently bad, the original specification itself may need to be modified. In simple cases, where modifications are device size adjustments or other well understood changes, software tools will be provided to close the loop. In general though, we leave the correction phase to the humans. This is one aspect of design that is extremely poorly understood and where human creativity can be best exploited.

This general iteration structure may be repeated at several levels by the designer-each loop refining one aspect or level of the design. The analysis routines may indicate fundamentally insoluble problems with the specifications, in which case the designer would have to undo some higher level decisions previously thought to be correct.

Synthesis modules make use of this structure to converge to device sizes or to determine the basic topology or floorplan of a circuit. Such a synthesis module would use simplified correction modules as a heuristic means of generating solutions to design problems. In sufficiently well understood situations, the heuristic correction module might be completely adequate in which case the designer would need never be bothered by the correction phase.

We have divided the design of integrated circuits into two domains: topological design and physical design. Each of these domains can be explored at multiple levels of detail. At the higher abstraction levels (low detail), topological design is concerned with architectural/functional specifications, register transfer descriptions and logic design. At the lowest abstraction levels topological design deals with the particular transistor topologies used to implement logical functions and the detailed electrical circuit design issues of the chip. The physical domain has a similar hierarchy. At the highest abstraction level, we are concerned with pin count, power utilization and speed. At lower levels, the concern moves to floor plans, power grids and other global organizing structures. At the lowest abstraction levels, our primitives are sticks representations or detailed mask specifications.

In our observations of real designers we have observed that they spend a large amount of time going back and forth between the different design domains, refining the design a bit at each step. One of the biggest problems in design now is that designers are not able to move between the domains quickly enough. Thus bad interactions between the domains or misconceptions between the different designs are not discovered until the final layout-very late in the design cycle.

management tools for VLSI design. Second, the internal structures and procedures are organized to match as closely as possible designers' semantics. Third, there are mechanisms to capture the designer's intent and where possible, all system initiated inquiries of the designer ask for the designer's intent, not how to solve the problem at hand. Finally, all tools are organized to so they can be easily used both by higher level tools as well as by designers. The following paragraphs give the basic user interaction model for Schema and illustrates the type of tools that make up Schema.

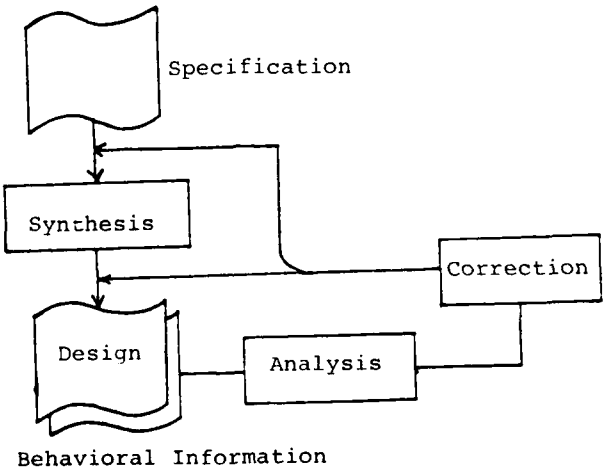


Figure 13-1: Interaction with Schema

Figure 13-1 gives a view of the designer's interaction with Schema at some level in some design domain. The designer initially provides a specification of the desired module. A synthesis module converts this specification into a design. Designs consist of a detailed topological or geometrical specification and a behavioral description-the synthesis module's intent. The behavioral description indicates how the components selected and connected by the synthesis modules are intended to satisfy the designer's original specifications.

By using simulation or some other checking mechanism, the analysis modules attempt to verify that the design meets the designer's original goals, as specified to the synthesis module or otherwise made known to the system. If the design does not meet these goals, the analysis tools are expected to indicate the components that did not meet their behavioral specifications and in what way they caused the overall design to fail. Most existing simulation tools (e.g. Spice, RSIM) perform only part of

third value "X", or "unknown". If an expression evaluates to true, the corresponding path exists; if the expression evaluates to false, no path exists. Since nodes can have X values, expressions involving node values can evaluate to X; in this case, the corresponding path may or may not exist. The values of the four node equations can be combined in a simple way to determine the final logic value of a node. In other words, all information about paths in the original network is now stored in the node equations, where it can be efficiently utilized.

Some circuit configurations have very simple connection paths during actual operation of the circuit, but the circuits can appear very complicated when no information is known about the values of various control lines. This is especially true of a circuit containing mosfet switching logic, such as a barrel shifter or tally circuit. The logic equations for a node in such circuits can become very large - in some cases, large enough to be impractical. The compiler monitors the size of the equations under construction; if they grow too large, the compilation is aborted and the node is flagged. At simulation time, the value of a flagged node is determined using the normal switch-level simulation algorithm. Using this technique, the speed-up in simulation afforded by the use of logic equations can be enjoyed by circuits even where 100% conversion to equations is not possible. Here we see one of the advantages of a general-purpose computer over special purpose simulation hardware: reversion to ordinary switch-level simulation for especially complicated circuits is easily accomplished by general-purpose computers, but can be next to impossible for special-purpose hardware.

A prototype implementation of ESIM2 was built during the last year, and its performance compared to that of the original ESIM simulator. ESIM2 was found to be 50 to 100 times faster. Interestingly, the more complex the circuit, the greater the speed-up; a not entirely unexpected phenomenon since the cost of re-examining the network in the original simulator grows rapidly as network complexity increases. The increase in performance provided by ESIM2 should permit the simulation of substantially larger circuits than before possible.

4.2. Schema Developments

Schema is an integrated environment for doing VLSI design. By providing canonical representations (for circuits and waveforms), interfaces and other support tools, it serves as a shell into which a large number of different electronic design tools can be plugged. It is, in a very real sense, an operating system for electronic design. In this section we describe a bit of the Schema philosophy in addition to mentioning the progress we have made in actually building the system.

There are four basic tenets behind the organization of Schema. First, Schema provides a coherent and cohesive set of data structures, databases and database

(e.g., command interpretation, access to the network data base, model evaluation), and their performance suffers as a result. Happily, much of this overhead can be eliminated through the application of traditional compilation techniques. This is the theme of this section, and the motivation for the development of ESIM2 (Terman), a combination compiler/simulator. ESIM2 compiles a network into a set of simulation subroutines, one for each strongly connected subnetwork (i.e., pieces of the network connected through one or more mosfet source/drain connections). Each subroutine computes the new value of one or more nodes from their old values and the values of other nodes in the network. While performing the same simulation calculation as earlier switch-level simulators, these subroutines arrive at the answer much more quickly - performance improvements of two orders of magnitude are typical. There has been much interest recently in special purpose hardware for simulation (e.g, the IBM Yorktown Simulation Engine, and the Zycad Logic Evaluator). It may be that such developments are premature, and that substantially better simulation performance can still be obtained from general purpose computers.

The switch-level simulation algorithm incorporated in RSIM determines the value of a node from information about the node's current connections to VDD and GND. The information is re-gathered each time a new value is calculated for the node. In most cases, only a small number of potential paths exist from a node to VDD or GND. This suggests that it might be economical to determine ahead of time the conditions for which a path exists to, say, GND. For example, the output of a NOR gate with inputs A and B is pulled down if either A or B is non-0. The existence of a pulldown path can be determined by evaluating the expression "A or B"; a search of the network is not required to discover which pulldowns are currently conducting.

ESIM2 builds a set of four equations for each node in the network:

DH_A	An expression indicating under what conditions a path of conducting n-channel and/or p-channel devices exists from node A to VDD.
DL_A	An expression indicating under what conditions a path of conducting n-channel and/or p-channel devices exists from node A to GND.
WH_A	Same as DH_A except the path contains at least one depletion device.
WL_A	Same as DL_A except the path contains at least one depletion device.

The equations involve ordinary Boolean operations, extended to accommodate a

2) Partitioning the network among the available processors (Terman). Traditional static partition methods (e.g., min-cut) can be used to ensure that interprocessor communication is kept to a minimum. However, partitions built in this way often result in an unbalanced simulation load, where a few of the processors are responsible for a bulk of the simulation computation. A pseudo-dynamic partitioning scheme is under investigation. Here a short simulation of the whole network is metered to determine which subnetworks are the most active. This information is used to weight each subnetwork, and the static partitioning technique is augmented to minimize communication costs subject to the constraint of keeping the total weight of each processor more or less equal.

3) Integrating the interprocess communications with the local simulation algorithm (Arnold). The naive event-wheel implementation of the RSIM algorithm introduces an unwanted synchrony between processors since simulated time cannot advance on a particular processor until it is sure that it will not receive any more incoming events for the current simulated time. Thus all processors would run in lock-step; since the simulation calculation is relatively short, the communication overhead would be large, thus obviating many of the advantages of parallel simulation. To ease this constraint, we are proposing to implement a checkpoint/rollback scheme for event handling. Every so often the state of the event queue and each network node is saved. Whenever an event arrives which is to be scheduled for a time earlier than the current simulated time, the simulation state is restored from the latest checkpoint earlier than the new event. The new event can then be scheduled normally, and simulation proceeds. A periodic global synchronization keeps the storage associated with checkpoints within manageable proportions.

The initial implementation is for the Concert multiprocessor, made available to us under the offices of Prof. Halstead. We also anticipate running some experiments with our network of Texas Instrument NU machines, which will allow one to compare the performance tradeoffs between a loosely-coupled environment with the tightly-coupled Concert environment. Finally, both Harris Corporation and BBN have volunteered time on their multiprocessors, which will allow us to transfer our technology outside of MIT.

Simulation Using a Pre-compiled Network Model: A major motivation for new simulation technology is the desire to improve simulator performance. It seems that digital computers ought to be well suited for the simulation of digital logic. Unfortunately, current simulation schemes involve several layers of interpretation

5. Ward, S. "Present Status and Future Trends of Personal Computers," *Proceedings of JEIDA '83*, Tokyo, Japan, October 1983, to appear.

Theses Completed

1. Antaki, P. "A Circular Pipelined Bus Architecture," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

2. Bedonian, A. "A Microcoded Arithmetic Engine for Computer Graphics," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

3. Broadway, E. "A Three-Dimensional Graphics Editor," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

4. Burger, E. "MOS Simulation Techniques," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

5. Chiarchiaro, W. "Data Transmission via FM Radios and Protocols for a Packet Radio Network," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

6. Cosway, P. " A Multilevel Memory and Interface for Display Terminals," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1984.

7. Hood, P. "A Real Time Development System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

8. Gilbert, A. "A Critique of the SmallTalk-80 Graphics System," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

9. Goddeau, D. "A Multiple Processor Implementation of the TRIX Operating System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1983.

10. Mercier, J. "Performance Analysis of the UNIX Operating Systems," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1983.

11. Miller, J. "A Polyphonie Digital Music Synthesizer With Real-Time Computer Control," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

12. Minsky, H. "A SCHEMA Interface for Signal Specification," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

13. Noakes, M. "An Assembler for PCF," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

14. Pasieka, M. "An Examination of Architectures for Interfacing to the NuBus," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

15. Pezaris, J. "INAD: An Intelligent Automated Dialer," S.B. thesis, Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

16. Osborne R. "Modeling and Evaluating the Performance of the Concert Multiprocessor," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

17. Sieber, J. "TRIX: An Operating System Supporting Network Communications," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1983.

18. Sterling, T. "Parallel Control Flow Mechanisms for Dynamic Scheduling of Tightly Coupled Multiprocessors," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, January 1984.

19. Stewart, W. "A Solution to a Special Case of the Synchronization Problem," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, September 1983.

20. Terman, C. "Simulation Tools for LSI Design," Ph.D. dissertation MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, August 1983.

21. Vance, J. "A Lisp to MultiLisp Compiler: Installing Cost-Effective Parallelism," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

22. Wade, M. "A More Efficient Video Display Terminal Management Scheme," MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

Theses in Progress

1. Masterson, A. "A Distributed Graphics System for the Nu Machine," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, Expected June 1985.

2. Robinow, D. "A Window System For Trix: A Network-Oriented Operating System," MIT Department of Electrical Engineering and Computer Science,Cambridge, MA, May 1984.

3. Teixeira, T. "Compiling Programs to Meet Performance Requirements," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, Expected September 1984.

Talks

1. Dertouzos, M.L. "Networked Computer Systems: Issues and Trends," MIT Industrial Liaison Program Symposium, Cambridge, MA, May 14, 1984.

2. Dertouzos, M.L. "Software Technology and Computer Architectures by 2000 A.D.," National Academy of Engineering 19th Annual Meeting, November 2, 1983.

3. Dertouzos, M.L. "Knowledge Base and The Fifth Generation," Technology Transfer Institute, Arlington, VA, November 1983.

4. Dertouzos, M.L. "Long Term Trends in Information Technology," Harris Corporation, Melbourne, FL, September 1983.

5. Dertouzos, M.L. "Computer Tribes and Computation by the Yard: The Emerging Multiple-Processor Revolution," EECS Colloquium, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, March 5, 1984.

6. Dertouzos, M.L. "Tomorrow's Exciting Information Technology," Analog Devices General Technical Conference, Boxborough, MA, March 20, 1984.

7. Dertouzos, M.L. "American Industry and the University: New

Possibilities for Hi-tech Cooperation," Japan Society, New York, NY. April 6, 1984.

8. Halstead, R.H. "MultiLisp: A Language for Symbolic Computing on Parallel Machines,"
   N.Y.U. Courant Institute of Mathematical Studies, NY, September 1983.
   Lawrence Livermore National Laboratories, Livermore, CA, October 1983.
   University of California at Berkeley, Berkeley, CA, October 1983.
   Carnegie-Mellon University, Pittsburgh, PA, January 1984.
   Siemens A.G., Munich, W. Germany, March 1984.
   Bell Laboratories, Holmdel, NJ, April 1984.

9. Terman, C. "RSIM - A Logic-Level Timing Simulator," IEEE International Conference on Computer Design; VLSI in Computers, Port Chester, NY, November 1983.

10. Ward, S. "Present Status and Future Trends of Personal Computers," JEIDA '83, Tokyo, Japan, October 1983.

11. Ward, S. "Computers, Communications, and Coherence," Siemens A.G., Munich, W. Germany, March 1984.

12. Ward, S. "The TRIX Distributed Operating System and Tomorrow's Personal Computers," ILO Symposium on Networked Computer Systems, MIT, Cambridge, MA, May 1984.

13. Zippel, R. "Introduction to the Lisp Machine: Part I" MIT Laboratory for Computer Science, Cambridge, MA, March 1984.

14. Zippel, R. "The Design and Analysis of High Performance MOS Circuits," MIT Department of Electrical Engineering and Computer Science Colloquium, Cambridge, MA, April 1984.

15. Zippel, R. "Introduction to the Lisp Machine: Part II" MIT Laboratory for Computer Science, Cambridge, MA, May 1984.

16. Zippel, R. "Circuit Design Using Schema," MIT Spring 1984 VLSI Research Review, Cambridge, MA, May 1984.

17. Zippel, R. "Architectures of Artificial Intelligence," Sikorsky Aircraft, Stratford, CN, May 1984.

SYSTEMATIC PROGRAM DEVELOPMENT

Academic Staff

J. Guttag, Group Leader

Research Staff

D. Detlefs

Graduate Students

R. Forgaard
R. Kownacki
R. Richards
K. Yelick
J. Zachary

Undergraduate Students

R. Robbins
S. Subramanian
M. Walters

Support Staff

E. Pothier

Visitors

P. Lescanne
J. Remy

1. INTRODUCTION

Over the last year the Systematic Program Development Group has worked on two related projects, the Larch Specification System and the REVE Rewrite Rule Laboratory. The work on Larch has been supported primarily by DARPA, and the work on REVE primarily by the NSF.

2. LARCH

2.1. Summary of Status and Plans

The Larch Project is developing tools and techniques intended to aid in putting formal specifications to productive use.

The central component of the project is a two-tiered approach to specification and a family of specification languages to support this approach. Each Larch language has a component derived from a programming language and another component common to all programming languages. We call the former interface languages, and the latter the shared language.

We use the interface languages to specify program modules. Specifications of the interface that one module presents to other modules often rely on notions specific to the programming language, e.g., its denotable values or its exception handling mechanisms. Each interface language deals with what can be observed about the behavior of programs written in its programming language. Its simplicity or complexity is a direct consequence of the simplicity or complexity of the observable state and state transformations of the programming language.

The shared language is used to specify abstractions that are independent of the programming language. These abstractions are used within specifications written in the interface languages. The role of shared language specifications is similar to that of abstract models in some other styles of specification.

We have completed the design and documentation of the Larch Shared Language [4][5]. We have also designed an interface language for CLU.

2.2. The Larch Family of Specification Languages

Some important aspects of the Larch family of specification languages are:

- *Composability of specifications.* We emphasize the incremental construction of specifications from other specifications. Larch has mechanisms for building upon and decomposing specifications as well as for combining specifications.

- *Emphasis on presentation.* Reading specifications is an important activity. To assist in this process, we use composition mechanisms defined as operations on specifications, rather than on theories or models.

- *Interactive and integrated with tools.* The Larch languages are designed for interactive use. They are intended to facilitate the interactive construction and incremental checking of specifications. The decision to rely heavily on support tools has influenced our language design in many ways.

- *Semantic checking.* It is all too easy to write specifications with surprising implications. We would like many such specifications to be detectably ill-formed. Extensive checking while specifications are being constructed is an important aspect of our approach. Larch was designed to be used with a powerful theorem prover for semantic checking to supplement the syntactic checks commonly defined for specification languages.

- *Programming language dependencies factored.* We feel that it is important to incorporate many programming-language-dependent features into our specification languages, but to isolate this aspect of specifications as much as possible. This prompted us to design a single shared language that could be combined in a uniform way with different interface languages.

- *Shared language based on equations.* The shared language has a simple semantic basis taken from algebra. Because of the emphasis on composability, checkability and interaction, however, it differs substantially from the "algebraic" specification languages we have used in the past.

- *Interface languages based on predicate calculus.* Each interface language is based on assertions that can be translated to first order predicate calculus with equality. Each interface language incorporates programming-language-specific features to deal with constructs such as side effects, exception handling, and iterators. Equality over terms is defined in the shared language; this provides the link between the two parts of a specification.

By referring to a shared language component in an interface specification, we gain the advantage that every symbol in an assertion is precisely defined within the specification language itself. In some other specification methods there is a reliance

on an interpretation for symbols in an assertion, where the interpretation comes from outside the specification language. In contrast, some other methods provide an assertion language defined within the specification language, but restrict the symbols to come from a fixed set of primitives. We gain the advantage that the user is able to provide just the symbols necessary to write the assertions in the body of a specification.

2.3. The Larch Editing Tool

A year ago, Joe Zachary completed the design of a syntax-directed editing tool that facilitates the reading and writing of Larch specifications [23]. This work proceeded from the premise that the time has arrived to construct specialized tools to support the specification process.

Currently, a writer faces a series of sequential syntactic and semantic checks in producing a Larch specification. Context-free constraints are defined relative to free text; context-sensitive checks, relative to sentences in the context-free language; and semantic constraints, relative to syntactically correct text. This style of definition lends itself to a batch style of text production, with the editing and multiple checking phases distinct from one another.

This style of support is not satisfactory, because it focuses upon detecting errors some time after they are committed. Errors are often symptomatic of some more fundamental confusion and should be pointed out to the writer as they occur. For this reason, the editing and checking phases in the Larch editor are integrated. The editor is interactively involved in the incremental production and checking of specification text. A full range of error prevention, avoidance, and detection is available automatically at all times during the development of a specification.

This year the editor was partially implemented. Suresh Subramanian built the kernel of the editor, which constructs and manipulates an abstract representation of Larch specifications [21]. This module directly incorporates the details of Zachary's design. Rich Robbins designed and implemented a terminal display scheme [19]. His design addresses the problem of efficiently updating a display in the face of incremental changes to an abstract text. Dave Detlefs is currently integrating these two modules with a user interface to produce a working editor.

2.4. Checking Larch Specifications

Larch shared language specifications must meet a variety of semantic constraints in addition to the syntactic constraints that are associated with any formal language. This semantic checking frequently reveals subtle mistakes. Incorrect specifications are as easy to write as incorrect programs. Just as the correctness of a program can

be partially established by testing it, we can partially establish the correctness of a formal specification by proving theorems about it. Ron Kownacki has developed techniques for automatically verifying that certain semantic requirements are met by a formal specification in the Larch shared language [12].

The shared language is an algebraic specification language. Each shared language specification has an associated first-order theory which is essentially a theory of equality on terms. The semantic constraints on shared language specifications are defined with respect to that theory. We have investigated the requirements of *consistency* and *completeness*. These properties are defined in a fashion that is similar to the corresponding definitions for theories in first-order predicate calculus. A specification is consistent if its associated theory does not contain contradictory formulas. Completeness is related to the adequate definition of operators.

We address two independent aspects of the completeness and consistency requirements as they are formulated for the shared language. Each is considered in a purely mathematical setting as well as from an implementation point of view. We have derived necessary and sufficient conditions that guarantee that these properties hold in an arbitrary shared language specification. These conditions must form the formal basis for any automated checking procedures.

The properties of completeness and consistency are undecidable for arbitrary theories. As there can exist no effective checking procedures, our goal has been to develop automated tests that usually succeed in practice. We have found such checking procedures for a subset of the language. The common styles of axiomatization that we have observed have greatly influenced their design.

We cannot yet provide an assessment of how well the automatic checking will work in practice, because an implementation does not yet exist. The automatic checks are designed to be used in conjunction with a theorem proving system that is based on term rewriting system technology. The effectiveness of the procedures hinges on the assumption that certain advances in term rewriting systems research will be forthcoming shortly. This assumption is justified by the rapid progress that is being made in the area. In the interim period, we can assess our progress by noting that our semantic checking problems have been reduced to other problems that are well-known to, and under investigation by, researchers in the field.

There are limitations to our results that should be noted. The automated tests have been developed for only a subset of the shared language. It is unclear whether our methods can be extended to encompass the full language, although they will certainly form an important component of any comprehensive semantic checking system. Another inherent limitation of our approach is its dependence upon certain styles of axiomatization. However, this dependence seems to be justified by the undecidability of the problems.

A final limitation is that we have not always considered the checking procedures in the context of the complete specification environment. Shared language specifications can be combined and modified in various ways. We would like to minimize the amount of checking that must be repeated after these manipulations. However, changes to a specification do not correspond directly to changes in the associated theory. For this reason, we consider semantic checking of an incrementally changing specification to be a separate research problem.

The implementation of these tests will rely heavily upon the technology of term-rewriting systems. REVE (see Section 3) will provide the necessary support for this project.

3. THE REVE THEOREM PROVER

The availability of automatic tools for reasoning about Larch specifications is essential to our specification approach, and has influenced the design of Larch itself. Accordingly, we have devoted considerable effort to the further development of REVE, a theorem prover for equational theories. REVE will eventually be used to verify the consistency of specifications written in the Larch shared language, and to help analyze the meaning of such specifications (Section 2.4). REVE is now in use in many laboratories in the United States and abroad.

Several procedures are known theorem proving using rewriting techniques. The most important of these is the Knuth-Bendix completion procedure [11], which attempts to "complete" a rewriting system to make it confluent, and thus provide a decision procedure for the equational theory of the rewriting system. Knuth-Bendix requires the use of a unification algorithm, and a reduction ordering on terms to prove that the rewriting system terminates. The termination proof is the chief impediment to automatic theorem proving with Knuth-Bendix: using current technology, user interaction is usually required to assist the system in making the appropriate decisions during the proof. Many systems, such as Affirm [17], offer no support for the termination proof.

REVE was developed in two stages. REVE 1 [13] was conceived and implemented by Pierre Lescanne during his visit with SPD in 1980-82. It included one of the first implementations of Knuth-Bendix to deal effectively and flexibly with termination. REVE 2 [3] was engineered from the ground up by Randy Forgaard and Dave Detlefs, expanding upon Lescanne's ideas. In addition, REVE 2 seeks to provide 1) a solid source code base upon which to build, 2) automatic theorem proving capabilities, suitable for embedding in other applications, and 3) a friendly and powerful user interface. REVE 2 has been modularized and documented to meet the first goal, including a complete set of abstractions pertinent to rewriting applications. We have made substantial progress toward the second goal by incorporating

features into REVE 2 that allow termination proofs and Knuth-Bendix to proceed nearly automatically in an intelligent fashion. The third goal has been addressed with a flexible command interpreter that provides a rich set of commands, explicit control over rewriting and termination proof strategies, on-line help facilities, and detailed error messages.

The following sections discuss the most significant aspects of REVE. Section 3.1 outlines our current work on automatic termination proofs in REVE. Section 3.2 describes REVE's failure-resistant Knuth-Bendix implementation. Section 3.3 discusses REVE's role as a rewrite rule laboratory, and our work with General Electric in this regard. Finally, Section 3.4 presents the REVE work, currently underway, that will be the major implementation thrust in the coming year.

3.1. Toward Automatic Proofs of Termination for Rewriting Systems

REVE provides a number of simplification orderings [1] for use by Knuth-Bendix. These orderings on terms are a powerful means of proving termination: as each equation is turned into a rewrite rule, the rule is oriented so that the left-hand side is greater (under the ordering) than the right-hand side. Recent simplification orderings are parameterized on a precedence (partial ordering on equivalence classes of the basic function symbols) and status map (for each function symbol, the relative importance of its arguments in the simplification ordering).

Though simplification orderings are more satisfactory for automatic termination proofs than other reduction orderings, most simplification ordering implementations require that the precedence and status map be given *a priori*. This limitation is a significant obstacle in the development of automatic termination proof techniques, since there is no easy way to automatically choose an appropriate precedence and status map beforehand.

To address this difficulty, REVE allows the precedence and status map to be extended incrementally. Whenever an equation cannot be ordered under the current precedence and status map, the equation is presented to the user, and the user chooses extensions that might allow the equation to be ordered. The recursive decomposition ordering with status (RDOS) [8][14][15] gives partial support for this method by suggesting precedence extensions that might order the equation. Even with the suggestions from RDOS, however, the termination proof is not automatic. RDOS provides no suggestions for making function symbols equivalent in the precedence. There is no automatic means for deciding on the appropriate subset of the RDOS suggestions to use in extending the precedence. RDOS gives no suggestions for extending the status map.

To solve these problems, we developed and implemented the direct synthesis

monotonic path ordering with status (DSMPOS) in June 1984. When an equation cannot be ordered, the implementation of DSMPOS provides a complete set of minimal precedence and status map extensions that will order the equation. If some equation is encountered for which there are no such extensions, REVE's "undo" facility can be used to back up to some decision point and choose a different extension for a previous unorderable equation. When such a scheme is systematically pursued, it amounts to an automatic depth-first search for some precedence and status map that will prove the termination of the rewriting system. Presently, REVE presents the complete set of minimal extensions to the user, and the user chooses one. We are currently implementing a facility for automatically iterating over the possible extensions and automatically undoing when a dead end is reached.

Although DSMPOS and RDOS are the same ordering when the precedence and status map are total, RDOS can order some equations that are not orderable by DSMPOS under a partial precedence and status map. In February 1984, we developed the exhaustive monotonic path ordering with status (EMPOS), which is more powerful than RDOS, and has some of the automatic features of DSMPOS.

Both DSMPOS and EMPOS are based on the monotonic path ordering with status (MPOS), an improvement to the recursive path ordering with status (RPOS) [10]that makes the ordering monotonic in the status map. In both RPOS and RDOS, extending the status map may change the orientation of previous rewrite rules, or even make previously ordered rules unorderable. The monotonicity properties of MPOS solve these problems.

3.2. A Failure-Resistant Implementation of Knuth-Bendix

REVE's Knuth-Bendix implementation is an improved version of Huet's [6], which is itself a more efficient version of the original [11]. The chief improvements of the REVE implementation over Huet's are:

- REVE does not require that the precedence and status map be given *a priori*. They may be incrementally extended as each unorderable equation is considered.

- The user may interrupt and "undo" the completion process at any time, to return to any previous decision point. This might be used, for example, to choose a different precedence or status map extension for a previous unorderable equation.

- REVE's Knuth-Bendix does not necessarily fail when an unorderable equation is found. If there are no acceptable precedence or status map

extensions for an unorderable equation, the user may postpone the equation.

- REVE incorporates a postponement scheme that assists in applications where automatic operation is required, including the automatic postponement of large equations.

- REVE computes smaller critical pairs first, which can expedite the completion process.

3.3. Rewrite Rule Laboratory

While the progress of research into rewriting systems has been significant, it has been impeded by the inordinate difficulty of implementing and using the increasingly complex algorithms prevalent in state-of-the-art term rewriting research. The crux of the problem is twofold: the large effort required to build state-of-the-art software and the difficulty of acquiring usable software from others. The difficulty of acquiring or constructing good rewriting software serves both to slow down the work of those already involved in studying or using term rewriting systems and to inhibit the entry of new researchers into the field. It affects theoretical work as well as application-oriented work.

To help bridge the software gap, REVE provides some limited experimentation facilities, and has been carefully modularized with layered, rewriting-related abstractions to permit fast implementations of new algorithms using REVE's modules. In the past year, researchers in several laboratories have successfully written applications that use some or all of the modules in REVE.

3.4. Equational Term Rewriting Systems

The correctness of Knuth-Bendix requires that the rewriting system terminate at each step of the procedure. This requirement, together with the limitations of classical unification, disallows the use of useful permutative equations such as commutativity.

Jouannaud and Kirchner [7][9] have addressed this problem by extending the Knuth-Bendix procedure to operate upon an *equational* term rewriting system (ETRS): a rewriting system together with a set, E, of equations. The equations in E are not converted into rules; instead, extended Knuth-Bendix exploits a finite and complete unification algorithm for the equational theory of E. In a cooperative venture, Helene Kirchner and Claude Kirchner of the Centre de Recherche en Informatique de Nancy, France, are developing REVE 3, which will incorporate the Jouannaud-Kirchner procedure. Their implementation makes use of the generalized

- whenever the algorithm produces a bisection, it is guaranteed to be optimal (i.e., the algorithm also produces a proof that the bisection it outputs is an optimal bisection), for any kind of graph,
- the algorithm works well both theoretically and experimentally,
- the algorithm employs global methods such as network flow instead of local operations such as 2-changes, and
- the algorithm works well for graphs with small bisections (as opposed to graphs with large bisections, for which random bisections are nearly optimal).

2.6. Benny Chor

My research during the last year was primarily concentrated in cryptography. The main results are contained in two papers. One was written jointly with Oded Goldreich (to appear in *FOCS, October 1984*) [5], *and the other with Ron* Rivest (submitted to Crypto84) [6].

In the first paper, we prove that RSA least significant bit is $1/2 + 1/\log^{cN}$ secure, for any constant c (where N is the RSA *modulus*). This means that an adversary, given the ciphertext, cannot guess the least significant bit of the plaintext with probability better than $1/2 + 1/\log^{cN}$, unless he can break RSA.

Our proof technique is strong enough to give, with slight modifications, the following related results:

1) The *log N* least significant bits are simultaneously $1/2 + 1/\log^{cN}$ secure.

The above also holds for Rabin's encryption function.

Our results imply that Rabin/RSA encryption can be *directly* used for pseudo random bits generation, provided that factoring/inverting RSA is hard.

In the second, we introduce a new knapsack type public key cryptosystem. The system is based on a novel application of finite fields arithmetic, following a construction by Bose and Chowla. Appropriately choosing the parameters, we can control the density of the resulting knapsack. In particular, the density can be made high enough to foil "low density" attacks against our system.

and data type declarations. The syntax and the semantics of the typed lambda calculus can be made to play a crucial role in such descriptions.

Within this framework I studied different formalisms that can be used to add product and sum types to the the classical typed lambda calculus that has just functional types. I compared the reduction theories corresponding to these approaches from the perspective of strong normalization and Church-Rosser results.

Another topic that I researched is that of a unitary model theory for type-tree and typed lambda calculus. Within this topic I studied the various definitions of models of the lambda calculus that appear in the literature with particular emphasis on the extent to which these models behave like first-order algebras.

More generally, when we allow the types to satisfy constraints (domain equations), we are driven toward considering as models (small) cartesian closed categories. My current research goal is to describe a coherent model theory for this situation that would parallel and generalize the one for the type-free lambda calculus by using cartesian closed categories instead of lambda algebras.

2.5. Thang Bui

My major research activity of this year was to analyze the performance of graph bisection heuristics. This work was done in collaboration with S. Chaudhuri, T. Leighton, and M. Sipser.

The graph bisection problem is the problem of finding the minimal set of edges of a graph whose removal will divide the graph into two disjoint subgraphs of equal size. This problem is known to be NP-complete. There are some well-known heuristics for solving this problem, e.g., the Kernighan-Lin algorithm. However, no provable performance of any of these heuristics are known. The result of our work is a polynomial time algorithm. For every input graph, this algorithm either outputs the minimum bisection of the graph or halts without output. More importantly, we show that the algorithm chooses the former course with high probability for many natural classes of graphs. In particular, for every $d > 3$, all sufficiently large n and all $b = o(n^{1-2/(d+1)})$, the algorithm finds the minimum bisection for almost all d-regular graphs with n nodes and bisection width b. For example, the algorithm succeeds for almost all 5-regular graphs with n nodes and bisection width $o(n^{2/3})$. The algorithm differs from other graph bisection heuristics (as well as from many heuristics for other *NP*-complete problems) in several respects; most notably:

- the algorithm provides exactly the minimum bisection for almost all input graphs with the specified form, namely the d-regular graphs with $2n$ nodes and bisection width $b = o(n^{1-2/(d+1)})$, instead of only an approximation of the minimum bisection,

obtain a decision procedure using different notions of normal forms. Indeed Bercovici proved that any $(\beta\eta)$-normal form of a given term is reachable through a particular strategy and so the (possibly infinite) set of these forms are relatively easy to compute and can be used to compare terms. They are of course mostly useful for finding terms that are not provably equal. This result was used for finding unsolvable terms that are not provably equivalent.

Another strategy has been investigated that was inspired by the fixed point semantics [22]. Another direction of work was based on a suggestion of Michael Karr. He proposed a reduction system that extends the $(\beta\eta)$ rules plus x(Yx) --> Yx and is weak Church Rosser. A refinement of the ideas used for Tait's proof of strong normalization seem to be required to make the method work in this case.

2.3. Ravi Boppana

This year I have been doing research on the complexity of Boolean circuits. The circuit model that I consider allows alternating levels of AND gates and OR gates, with each gate allowed unbounded fanin and fanout. The size of a circuit is the number of gates it contains, and its depth is the number of alternating levels.

Furst, Saxe, and Sipser [9], and independently Ajtai [3], have shown that the parity function (which is 1 if an odd number of its variables are 1) requires more than polynomial size to compute on constant depth circuits. Unfortunately, the gap between their lower bound and the best known upper bound is large, and the exact complexity of the parity function is still a major open problem.

I haven't solved the above problem, but I have been able to solve a weaker problem. Consider the model of monotone circuits, which are circuits which have no negations appearing in them. In [4], I showed that computing the majority function (which is 1 if at least one half of its variables are 1) on constant depth circuits requires exponential size.

This result implies exponential lower bounds for other functions as well, including graph connectivity and detecting large cliques in graphs. These lower bounds are obtained using constant depth reductions.

Currently I am interested in extending these results to the case nonmonotine circuits. Even improved lower bounds for depth 3 circuits would be very interesting.

2.4. Val Breazu-Tannen

My research addresses problems encountered when trying to give adequate semantic descriptions for programming languages that allow higher-order procedure

2. MEMBER REPORTS

2.1. Amihood Amir

Temporal Logic: Previously no complete set of connectives was known for nonlinear time models. A method of constructing time models with a complete set of connectives is shown in [1] and examples of nonlinear expressively complete models are brought using this construction method.

VLSI Algorithms: Planarity checking for connectivity of modules was shown by Ron Pinter to be possible in linear time. A faster, direct algorithm is given in [2] for unflippable modules. If the algorithm can be adapted to the flippable case, it could offer an alternative to Hopcroft and Tarjan's planar graph embedding algorithm in some cases.

Lambda Calculus: Conditions are sought under which a fragment of a model - base type and some base to base functions - can be embedded in a fixed or least fix point type frame.

2.2. Irina Bercovici

Research in denotational semantics (see for example [22]) has strengthened interest in a number of extensions of typed λ-calculus. The major issues are decidability of equivalence and non equivalence of terms, complexity of decision procedures. The traditional decision process for establishing equivalence of λ-terms is: compute the normal forms and compare them. This idea works in systems whose equivalence relation is based on reduction rules that are Church-Rosser and weakly normalizable. One often proves these properties by actually proving only weak Church-Rosser, but strong normalization.

Bercovici analyzed these properties for λ calculus with subjective pairing. A stronger and corrected version of a paper by Gandy [10] proving strong normalization for pairing and direct sum was obtained. This new version (besides correcting flaws in the existing proofs) extends the method to incorporate the axioms of subjectivity for the two new operations and also does not require the assumption that the system is weakly normalizable. Another proof of strong normalizability for the system with subjective pairing was provided (joint work with Michael Karr, Harvard) using Tait's method.

The real challenge is proving decidability for another extension of the calculus, namely the one endowed with the constant Y (at all appropriate types) and the axiom $Yx = x(Yx)$. Using the rule $Yx \rightarrow x(Yx)$ together with (β) and (η) one obtains a Church Rosser system which is obviously not terminating. Still one may hope to

1. INTRODUCTION

The Theory of Computation Group has had a productive and prolific year, with research progressing on many fronts. Specifically, work was done in the areas, among others, of:

- models of the (typed and untyped) lambda calculus
- VLSI placement and routing algorithms
- graph bisection heuristics
- analysis of the security of the RSA cryptosystem
- new knapsack-type public-key cryptosystems
- database theory
- pseudo-random number generation
- knowledge complexity
- new signature schemes resistant to adaptive chosen-message attacks
- systolic arrays
- relativized computations
- algorithms on groups
- algorithms for sliding-block puzzles
- parallel computation
- "fat-tree" supercomputer architectures
- complexity of unification procedures
- complexity-based theory of randomness
- circuit complexity of finite functions

The progress of the Theory Group is summarized in the following reports by the individual members of the group.

Undergraduate Students

P. Gaber
D. Jilk
J. Kilian
M. Novick
S. Rao

Support Staff

A. Benford
I. Radzihovsky
K. Story

THEORY OF COMPUTATION

Academic Staff

P. Elias
S. Goldwasser
C. Leiserson
T. Leighton
N. Lynch
S. Micali
A. Meyer
G. Miller
R. Rivest, Group Leader
M. Sipser

Research Staff/Visitors

A. Amir
M. Ben-Or
I. Bercovici
C. Dwork
H. Fell
J. Friedman
O. Goldreich
H. Heller
S. Homer
P. Kanellakis
D. Kfoury
L. Levin
K. Lieberherr
S. Schwarz
B. Trakhtenbrot
A. Shamir
R. Street
S. Zachos

Graduate Students

P. Berman
S. Bhatt
V. Breazu-Tannen
R. Boppana
T. Bui
J. Buss
B. Chor
S. Cosmadakis
L. Greenberg
W. Gassarch
T. Hagerup
R. Hirschfeld
M. Komichak
D. Kornhauser
M. Lepley
M. Maley
J. Mitchell
A. Moulton
C. Phillips
M. O'Connor
A. Sherman
P. Shor
S. Sur
S. Trilling
L. Yedwab
P. Weiss

4. Walters, M.G. "An Illustrated Object Code Optimizer," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

Theses in Progress

1. Forgaard, R. J. "A Program for Generating and Analyzing Term Rewriting Systems," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1984.

2. Yelick, K.A. "A Generalized Approach to Equational Unification", S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1984.

Talks

1. Forgaard, R.J. "REVE: A Term Rewriting System Generator with Failure-Resistant Knuth-Bendix," NSF Sponsored Workshop on Term Rewriting Systems;

2. Guttag, J.V. "Where and Wither Formal Specifications,"
   GTE Laboratories;
   University of Delaware;

3. Guttag, J.V. "An Overview of the Rewrite Rule Laboratory Project," NSF Sponsored Workshop on Term Rewriting Systems.

4. Guttag, J.V. "The Relationship of Mechanical Theorem Proving to Programming Methodology," IFIP Working Group 2.3

5. Guttag, J.V. "The Larch Family of Languages,"
   Wang Institute;
   IFIP Working Group 2.2.

21. Subramanian, S. "The Implementation of an Abstract Editor for Larch," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

22. Yelick, K.A. "A Generalized Approach to Equational Unification", S.M. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected 1984.

23. Zachary, J.L. "A Syntax-Directed Tool for Constructing Specifications," S.M. Thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, March 1983.

Publications

1. Forgaard, R. and Guttag, J.V. "REVE: A Term Rewriting System Generator with a Failure Resistant Knuth-Bendix," *Proceedings of a Workshop on Term Rewriting*, April 1984.

2. Guttag, J.V., Kapur, D. and Musser, D.R. (eds.), *Proceedings of an NSF Workshop on the Rewrite Rule Laboratory*, General Electric Corporate Research and Development Report No. 84GEN008, April 1984.

3. Guttag, J.V. and Horning, J.J. "An Introduction to the Larch Shared Language," *Proceedings of the IFIP Congress '83*, Paris, France, 1983.

4. Guttag, J.V. and Horning, J.J. "Preliminary Report on the Larch Shared Language," MIT/LCS/TR-307, MIT Laboratory for Computer Science, Cambridge, MA, October 1983.

5. Zachary, J.L. "Type Inference in MEDEE", Centre de Recherche en Informatique de Nancy, 83-R-043, Paris, France, August 1983.

Theses Completed

1. Kownacki, R. W. "Semantic Checking for Formal Specifications," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

2. Robbins, R. E. "Display Support for a Structure Editor," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

3. Subramanian, S. "The Implementation of an Abstract Editor for Larch," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

Orderings," University of Illinois, Department of Computer Science, Urbana-Champaign, IL, February 1980, to appear.

11. Knuth, D.E. and Bendix, P.B. "Simple Word Problems in Universal Algebras," in Computational Problems in Abstract Algebra J. Leech (ed.), Pergamon, Oxford, England, 1970, 263-297.

12. Kownacki, R.W. "Semantic Checking of Formal Specifications," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

13. Lescanne, P. "Computer Experiments with the REVE Term Rewriting System Generator," *Proceedings of the 10th ACM Symposium on Principles of Programming Languages*" Austin, TX, January 1983, 99-108.

14. Lescanne, P. "Uniform Termination of Term Rewriting Systems: Recursive Decomposition Ordering with Status," *Proceedings of the 6th CAAP*, Cambridge University Press, Bordeaux, France, March 1984.

15. Lescanne, P. "How to Prove Termination? An Approach to the Implementation of a new Recursive Decomposition Ordering," *Proceedings of an NSF Workshop on the Rewrite Rule Laboratory*, September 6-9, 1983. Also General Electric Corporate Research and Development Report No. 84GEN008, Schenectady, NY, April 1984, 109-121.

16. Levesey, M. and Siekmann, J. "Unification of A + C Terms (Bags) and A + C + I Terms (Sets)," Institut fur Informatik I, Universitat Karlsruhe, Interner Bericht Nr., March 1976.

17. Musser, D.R. "Abstract Data Type Specification in the Affirm System," *Journal of IEEE TSE*, 6,1 (January 1980), 24-32.

18. Plotkin, G.D. "Building-in Equational Theories," in Machine Intelligence, Meltzer, B. and Michie (eds.), Edinburgh University Press, Edinburgh, Scotland, Volume 7, 1972, 73-90.

19. Robbins, R.E. "Display Support for a Structure Editor," MIT Laboratory for Computer Science, Cambridge, MA, June 1984.

20. "Stickel, M.E. "A Unification Algorithm for Associative-Commutative Theories," *Journal of the ACM*, 28, 3, (July 1981), 423-434.

References

1. Dershowitz, N. "Orderings for Term-Rewriting Systems," in *TCS* 17, North Holland, 279-301, 1982, Also Preliminary version in *Proceedings of 20th IEEE Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, October 1979, 123-131.

2. Fages, F. "Associative-Commutative Unification," *Proceedings of the 7th CADE*, Napa Valley, CA, 1984.

3. Forgaard, R. "A Program for Generating and Analyzing Term Rewriting Systems," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, to appear.

4. Guttag, J.V. and Horning, J.J. "Preliminary Report on The Larch Shared Language," MIT/LCS/TR-307, MIT Laboratory for Computer Science, Cambridge, MA, October 1984. Also Technical Report CSL-83-6, Xerox Palo Alto Research Center, Palo Alto, CA, September 1983.

5. Guttag, J.V. and Horning, J.J. "An Introduction to the Larch Shared Language," *Proceedings of IFIP-83 Congress*, 1983

6. Huet, G."A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm," *JCSS*, 23, 1, (August 1981) 11-21.

7. Jouannaud, J.P. "Church-Rosser Computations with Equational Term Rewriting Systems," Centre de Recherche en Informatique de Nancy, Nancy, France, January 1983.

8. Jouannaud, J.P. Lescanne, P. and Reinig, F. "Recursive Decomposition Ordering," *Proceedings 2nd IFIP Workshop on Formal Description of Programming Concepts*, Garmisch-Partenkirchen, W. Germany June 1982. Also "Recursive Decomposition Ordering and Multiset Orderings," MIT/LCS/TM-219, MIT Laboratory for Computer Science, Cambridge, MA, June 1982.

9. Jouannaud, J.P. and Kirchner, H. "Completion of a Set of Rules Modulo a Set of Equations," Technical Note, SRI International, Computer Science Laboratory, Menlo Park, CA, April 1984. Preliminary version in *Proceedings of the 11th ACM Symposium on Principles of Programming Languages*, Salt Lake City, UT, January 1984.

10. Kamin, S. and Levy, J.J. "Attempts for Generalizing the Recursive Path

- *Minimality:* The set of unifiers produced must be the maximally general set.

- *Efficiency:* Comparisons must be made between different unification algorithms for the same E-theory, and between E-unification and alternative approaches to embedding the same equational information in the system.

The AC-unification algorithm was studied in detail. The best known algorithms [20][16][2], are not minimal. Although minimality can be achieved in any E-unification algorithm by searching the resulting substitution set for pairs of substitutions in which one is a factor of the other, this check is very expensive. Minimality is important from an efficiency standpoint because otherwise a large set of unifiers will have to be handled by the surrounding application. We have found a modification to the published algorithms that eliminates many non-minimal unifiers and improves the running time on many practical examples.

implemented. In the implementation, new E-unification modules can literally be plugged in by adding them to the list of currently supported algorithms.

To justify the soundness of our approach, a formal analysis was carried out. First, a proof of correctness was found for the generalized algorithm to show that all substitutions produced were in fact unifiers. Second, a proof of completeness was developed. Proving termination has proven to be the hardest part of the analysis. Given terminating E-unification algorithms, it is possible that the combined unification algorithm will not terminate, although no examples of this have been found. We are currently looking for sufficient conditions upon equational theories that assure termination.

In addition, the unification algorithms for the associative-commutative and empty theories were implemented as the most useful theories in practice and as examples for testing the generalized algorithm. The following section discusses the AC-unification algorithm as well as some of the criteria used in choosing AC-unification as a useful kind of E-unification. The implementation effort was started in October, 1983 and a working system was completed in January of 1984; another month was spent optimizing and improving the implementation, particularly the associative-commutative algorithm.

AC-Unification: To provide examples for testing the generalized algorithm, the unification algorithm for associative-commutative theories was implemented. One reason for choosing AC-theory is that AC operators such as multiplication and addition occur frequently in practice. A second reason is that these axioms cause difficulties in automated reasoning systems. In PROLOG, for example, if a relation is declared to be commutative using a normal procedure declaration, the program will loop. The associative and commutative axioms cause similar problems in term rewriting systems. Finally, unlike some equational theories in which unification may be undecidable or inherently non-terminating, an effective unification algorithm is known for AC-unification.

There are five properties of unification algorithms that we consider desirable, listed in decreasing order of importance:

- *Correctness:* Extraneous substitutions that are not unifiers of the input terms cannot be produced.
- *Completeness:* The set of unifiers produced must at least contain all of the maximally general unifiers.
- *Termination:* This is not always essential, depending upon the problem domain.

unification strategy designed by Kathy Yelick of SPD, as well as her implementation of a unification algorithm for AC theories [22] (see below).

Unification: Given two terms containing function symbols and variables, the classical unification problem is to find a uniform replacement of terms for variables that renders the two terms equal. For example, the terms f(g(x), g(y)) and f(y, z) can be unified by replacing y with g(x) and z with g(g(x)). We are usually interested in the variable to term mapping, called a *substitution*, which makes the terms equal, and not just in the resulting unified term. In general, there are an infinite number of unifying substitutions for two terms, but in classical unification all unifiers are an instance of a unique *most general unifier*.

The idea of extending the unification process to *equational unification* was first suggested by Plotkin [18]), who showed how it can be used in conjunction with resolution-based theorem provers. In equational unification, or E-unification, a substitution unifies two terms if applying the substitution to the input results in terms that are provably equal in a given equational theory. In some equational theories there is not a single most general unifier but rather of set of *maximally general unifiers* which characterize the set of unifying substitutions.

Generalized Unification: Much of the work to date in equational unification has been theoretically motivated--deciding whether there exists a unification algorithm for a particular equational theory and what form the algorithm takes. The theoretical bent of the field has led to some simplifying assumptions on the structure of terms; many of the algorithms are developed to handle terms whose operators all have the same properties.

Yelick has defined a generalized approach to the unification process by showing how algorithms for different equational theories may be combined in some cases to yield unification procedures for terms with *mixed* sets of operators. For example, given a unification algorithm for the associative-commutative, or AC theory, and another for the abelian group theory, we can automatically combine the algorithms to unify terms such as x + (y * (-z)), where * is associative and commutative and + and - are operators of an abelian group.

The first stage in developing the generalized algorithm was an analysis of the unification process in different equational theories to determine what kinds of theories can be automatically combined and which steps in the unification process are independent of the equational theory being used. A theoretical description for the generalized algorithm was made which involved deciding on the exact interface and functionality of all E-unification algorithms as well as the encompassing module.

A second step was an implementation to support the generalized algorithm and allow simple, automatic extension to new E-unification algorithms as they are

2.7. Stavros Cosmadakis

We continued our research on the implication problem for functional and inclusion dependencies. Specifically, we concentrated on the problem of inferring functional dependencies in a pairwise consistent multi-relational database. We showed that the problem is undecidable in general, but becomes decidable if the functional dependencies satisfy an acyclicity condition. Under that condition, we also showed that the problem is finitely controllable, i.e., unrestricted implication coincides with finite implication. These results were obtained by exploiting a new graph-theoretic methodology introduced in [7].

2.8. Oded Goldreich

During this year, my main interests have been in cryptography and especially in two topics: strong pseudo random g ierators and the bit security of the plaintext when encrypted using the RSA scher).

2.9. Shafi Goldwasser

Algorithmic Randomness: Goldreich, Goldwasser and Micali [11] investigated the field of algorithmic randomness. The goal is to develop a coherent theory of randomness based on computational difficulty. The approach is very constructive. In view of applications, particular attention has been devoted to finding the most powerful properties of randomness that can be achieved by polynomial-time simulations.

The most significant result is a novel algorithm that transforms any one-way function (i.e., a possible candidate is the RSA function) and a secret randomly selected k-bit input 'i' to a pseudo random function f_i from k-bit strings to k-bit strings. These functions can be used in many applications. They are easy to select (by selecting their k-bit index) and easy to evaluate (on inputs i and x, $f_i(x)$ is computable in polynomial-time). Still they cannot be distinguished from truly random functions by any polynomial-time algorithm that does not know i but can ask for and receive the value of the function at inputs of its choice. The new construction has applications to cryptography, complexity theory and distributed computing.

Cryptography: Professor Goldwasser and Professor Blum (Berkeley) proposed a new probabilistic Public-Key Encryption Scheme [12]. Every message, in this system, has many possible encodings, each of which is selected by the sender at random via a sequence of coin flips. The security of this system is based on the assumption that the integer factorization problem is intractable. In particular, no polynomial time bounded line-tapper can compute any *partial information* about messages from their encodings, unless factoring integers is in polynomial time.

Partial information is defined to be *any* function (polynomial time, non-recursive...) defined on the message space. No deterministic encryption scheme, such as RSA, could possibly pass such security requirements. Our scheme is comparable in efficiency with existing deterministic schemes. It requires constant data expansion and is at least as fast at the RSA cryptosystem.

Cryptographic protocols are based on the secrecy of some private knowledge and should preserve such secrecy. The big open problems in the area are how to check the correctness of a cryptographic protocol and when the principle of modularity design can be correctly applied (i.e., under which conditions *correct* subprotocols can be combined in complex protocols so as to preserve correctness). Goldwasser, Micali and Rackoff [14] presented a novel computational complexity measure of knowledge and applied it to cryptographic protocols. They measured how much knowledge is released by the execution of a protocol. This lead to a theorem fundamental for checking correctness of protocols and to discovering sufficient conditions for correctly applying modular design.

Goldwasser, Micali and Rivest presented a new signature scheme [13]. The novelty of the scheme is that the ability of foregoing messages is *proved computationally equivalent to the problem* of integer factorization (in previous schemes it was only proved that the ability to factor would imply ability to forge, but not vice versa). Moreover, it is proved that forging remains hard even if an adversary is allowed to ask and receive signatures of messages of his choice; i.e., the scheme is secure against chosen ciphertext attack. Such a security feature is essential for some applications. For example, a notary public has no control over the messages that he has to sign. He is thus particularly vulnerable against chosen signature attack.

A. Yao shows how to use the Exclusive-Or function to transform a *weak* one-way function (a function which is hard to invert on a polynomial fraction of its domain) into an *unapproximable* Boolean predicate (a Boolean predicate which can be computed no better than by guessing at random, by any polynomial time, probabilistic algorithm). Professor Goldwasser gave a simplified proof of this theorem and showed how to construct a Boolean predicate which is unapproximable if and only if factoring is intractable using the Exclusive-Or operator. This predicate has applications to Pseudo Random Bit Generators and Public-Key Encryption.

2.10. Ron Greenberg

I have completed two major projects during my first year at MIT. The first one is an nMOS chip containing two processors for a linear-array systolic priority queue with variable length keys. This helped educate me about VLSI design and systolic systems rather than being an original research endeavor. The second project is a

systolic simulator intended to provide capabilities for describing systolic systems in a concise way, simulating them, and performing analyzes and transformations such as retiming. I believe that I have made a good start, but the simulator would benefit from the addition of some more descriptional aids and tools for analysis and transformation. Hopefully, this project will be put to practical use in high-level architectural investigations, such as "fat tree" simulations.

Currently, I am beginning to look into several issues raised by Professor Leiserson's work on fat trees.

2.11. Ray Hirschfeld

Almost all oracles separate P and NP, which lends credence to the conjecture that these two classes are in fact distinct. No analogous result is known for P vs. the intersection of NP and co-NP. Ray Hirschfeld is working on a version of this problem in which the machines can take as much time as they wish, but can make only a limited number of oracle queries. This may provide insight into the time-bounded case, which in turn will shed some light on the non-relativized situation. The relationship between these two complexity classes is of practical importance, since, for example, their equality implies that factoring integers is "easy."

2.12. Daniel Kornhauser

Daniel Kornhauser and Gary Miller found a polynomial time algorithm for deciding the solvability of a certain pebble motion problem on graphs, which is a generalization of Sam Loyd's '15-puzzle' and is related to problems of memory management in totally distributed computer systems. This decision algorithm generalizes an algorithm of R.M. Wilson which applied only to biconnected graphs with one blank vertex.

They also obtained (with the help of Paul Spirakis at NYU) matching lower and upper bounds of $O(n^3)$ for the number of moves required to solve the pebble problem. Their approach allows solutions to be efficiently planned. They hope that the algebraic methods used for this problem can be applied to geometrical movers' problems arising in robotics.

Kornhauser and Miller also studied the related problem of the diameter of permutation groups given by generators. They extended a result of Driscoll and Furst, who proved a polynomial upper bound on diameter when the generators are bounded cycles, to a moderately exponential upper bound for special cases of arbitrary length cycles.

2.13. Tom Leighton

Professor Leighton's current research interests are centered on problems in the areas of parallel computation, VLSI design and probabilistic analysis of algorithms. In the area of parallel computation, Professor Leighton is developing algorithms and networks (such as the mesh of trees) for very fast parallel computation. Work during the past year was highlighted by the discovery of the first N-node, bounded-degree network capable of sorting N numbers in O(log N) steps. Since sorting is a generalization of packet routing, the resulting network is universal in the sense that it can simulate any computation on any network with at most an O(log N) factor of delay in time, the least possible. In addition to establishing the existence of universal computers, the new work may also have applications to the design of supercomputers.

In the area of VLSI design, Professors Leighton and Bhatt developed a framework for solving VLSI graph layout problems. Graph layout problems model placement and routing problems, and the new framework provides techniques for minimizing such cost functions as area, wire crossings and propagation delay. The framework can also be effectively used to design regular and configurable layouts, to assemble large networks of processors using restructurable chips, and to configure networks around faulty processors.

In the area of algorithms, Professor Leighton is studying the average case behavior of heuristics for NP-complete problems such as bin packing and graph bisection. Such problems have many important practical applications, but no polynomial time algorithms are known to solve these problems. As a result, heuristics are used in the hope that they will work well most of the time. Professor Leighton is developing techniques that can be used to mathematically analyze the average case behavior of the heuristics. The initial work in this area has been very encouraging. Already, tight bounds have been discovered for the behavior of several commonly used bin packing heuristics, and a graph bisection heuristic was discovered that is guaranteed to work optimally for almost all graphs in several natural classes of graphs. The bin packing results, in particular, are also surprising since they disproved several widely believed conjectures.

2.14. Charles Leiserson

Professor Leiserson has continued his investigations in the areas of VLSI algorithms and parallel computation. By combining work that has been done in VLSI layout theory and routing networks, he has developed a new notion of universal routing network which includes the cost of hardware needed to build the routing network. He has discovered such networks called "fat trees," whose layout properties are based on Leighton's tree of meshes. Fat trees are universal in that

they can simulate every interconnection network, using only slightly more time than does an optimal implementation of the network, given the same hardware resources. Previous notions of universality have not included the cost of hardware.

The work on fat trees was stimulated by the Connection Machine project in the AI Laboratory. He and Professor Rivest contributed to the original design of Professor Knight's cross-omega network.

In the area of systolic and semisystolic design, Professor Leiserson has developed many transformations for optimizing semisystolic circuits. The tools include retiming, slowdown, broadcast and census elimination, resetting code motion, coalescing, and interlacing, to name a few.

Miller Maley and Charles Leiserson have been studying the problem of planar routing and routability. We have begun to develop a substantial theory which we hope to incorporate into a layout compactor that does optimal jog introduction. Cynthia Phillips and Charles Leiserson have developed an O(n lg n) time algorithm for finding the connected components of a set of rectangles in the plane. The algorithm is space-efficient in the sense that the amount of primary memory required is proportional only to the number of rectangles cut by a scan line. For VLSI applications, this number is about the square root of n, which means that the geometric design rules can be checked for extremely large chips.

Professor Leiserson has also developed several interesting chip architectures. One chip is a fast concentrator chip. This chip has a set of input lines and fewer output lines. Each input line has either a zero or a one. The ones indicate the presence of messages, and zeroes indicate the absence. The chip maps as many input lines with messages to output lines, and does it by going through only lg n gates. Also, with Ray Hirschfeld and Peter Gabor, Charles Leiserson has developed an area-efficient chip to do multiplication of complex numbers.

2.15. Miller Maley

Charles Leiserson and Miller Maley have examined the problem of routing wires on a single layer between fixed modules when the topology of the layout is given. They have discovered a succinct characterization of the set of such layouts that have legal routings, and used it to develop an efficient algorithm for testing routability. Recently, Maley has solved an important special case of the routing problem, where modules and wires are constrained to lie in a rectilinear grid. Over the summer, this work will be extended to more general routing problems, and should also lead to the solution of a problem of VLSI layout compaction with automatic jog introduction.

2.16. Silvio Micali

Algorithmic Randomness: Goldreich, Goldwasser and Micali investigated the field of algorithmic randomness [11]. The goal is to develop a coherent theory of randomness based on computational difficulty. The approach is very constructive. In view of applications, particular attention has been devoted to finding the most powerful properties of randomness that can be achieved by polynomial-time simulations.

The most significant result is a novel algorithm that transforms any one-way function (i.e., a possible candidate is the RSA function) and a secret randomly selected k-bit input 'i' to a pseudo random function f_i from k-bit string to k-bit strings. These functions f_i's can be used in many applications. They are easy to select (by selecting their k-bit index) and easy to evaluate (on input i and x, $f_i(x)$ is computable in polynomial-time). Still they cannot be distinguished from truly random functions by any polynomial-time algorithm that asks for and is given the value of the function at various inputs. The new construction has numerous applications to cryptography, complexity theory and distributed computing.

Cryptography: Cryptographic protocols are based on the secrecy of some private knowledge and should preserve such secrecy. The big open problems in the area are how to check correctness of a cryptographic protocol and when the principle of modularity design can be correctly applied (i.e., under which conditions *correct* subprotocols can be combined in complex protocols so to preserve correctness). Goldwasser, Micali and Rackoff [14] presented a novel computational complexity measure of knowledge and applied it to cryptographic protocols. They measured how much knowledge is released by the execution of a protocol. This lead to a theorem fundamental for checking correctness of protocols and to discovering sufficient conditions for correctly applying modular design. In particular a very efficient protocol for signing contracts has been found by Ben-Or, Micali and Shamir [15].

Goldwasser, Micali and Rivest presented a new signature scheme [13]. The novelty of the scheme is that the ability of foregoing messages is *proved* computationally equivalent to the problem of integer factorization (in previous schemes it was only proved that the ability to factor would imply ability to forge, but not vice versa). This scheme is even more attractive. It is proved that forging remains hard even if an adversary is allowed to ask and receive signatures of messages of his choice; i.e., the scheme is secure against chosen ciphertext attack. Such a security feature is essential in some applications. For example, a notary public has no control over the messages that he has to sign. He is thus particularly vulnerable against chosen signature attack.

2.17. Gary Miller

During the year we have begun a new research effort in fault tolerant computing in a highly parallel environment. We are interested in computational networks which lend themselves to a maximum amount of parallelism even at the cost of restricting the individual elements to be extremely simple. We are also considering the possibility that the devices may error.

At this point we have been considering boolean circuits with possible feedback. More generally, we consider weighted directed graphs where each node is a threshold device which takes on states, say, ∓ 1. We call this a network of threshold devices.

The calculation performed by a network of threshold devices will depend on the order in which the devices update there state. We have considered several different update rules including; simultaneous or synchronous, random, and worst-case updating.

Margaret Lepley and myself have been able to construct networks which when updated randomly run with only an exponentially small probability of error. In particular, we have constructed polynomial size random networks which simulate synchronous ones with only log degradation in time and only exponentially small probability of error.

We are now considering environments where devices may fail to update correctly. The goal is to construct random networks which may also have errors. We are fairly close to such a construction.

2.18. John Mitchell

Unification algorithms are used in theorem provers, PROLOG interpreters, and other symbol-manipulation tasks. Several current research efforts are aimed towards solving symbolic problems more quickly by using many processors in parallel. With Cynthia Dwork (MIT) and Paris Kanellakis (Brown University), I proved that unification is a complete problem for the class of problems that are solvable in polynomial time [8]. Since most complexity theorists do not believe that all problems computable in polynomial time can be solved appreciably faster in parallel, it is unlikely that a substantial speed-up of unification can be achieved using parallel computation. Our result holds for a number of variants of unification.

Programming language designers have studied compile-time type checking for several years. It is difficult to permit flexible use of procedures and still do compile-time checking. One important kind of "permissiveness" is to allow programmers to write polymorphic procedures, procedures that accept many different types of

parameters. In [16][17], I studied an implicit approach to polymorphism based on algorithms that infer types for typeless programs. In forthcoming work with Gordon Plotkin (Edinburgh), we formulate a strongly-typed language with polymorphic functions and a very flexible kind of data abstraction. The formal semantics of this language is studied in [18].

2.19. Andy Moulton

During the past year, I have developed algorithms to lay power and ground wires on a VLSI chip. These algorithms form part of the Placement-Interconnect (PI) System, developed under Professor Ronald Rivest. As such, they take advantage of the way in which PI places the logic modules and pads. The power-ground algorithms first lay a ground ring outside the pads, then lay a power ring inside the pads. The algorithms lay a tree that connects the logic modules' terminals to the ground pad, then lay branches that connect the logic modules' terminals to the power ring without crossing the ground wires.

I finished my master's thesis, which fully describes the algorithms mentioned above [19]. This thesis also discusses a technique of keeping the power wires from crossing the ground wires. A Hamiltonian cycle goes through every module of the chip in such a way that the ground terminals lie inside the Hamiltonian cycle while the power terminals lie outside. Routing the ground tree inside the cycle and the power tree outside the cycle ensures that the wires of one tree do not cross the wires of the other tree. This technique is described in a paper I presented at the ACM IEEE 20th Design Automation Conference [20].

2.20. Cynthia Phillips

I am writing a paper with Charles Leiserson on a space and time efficient algorithm for finding the connected components of rectangles in the plan. I plan to continue work on space efficient algorithms for computational geometry. I'll also be working with Charles on "fat trees".

2.21. Ronald L. Rivest

During the past year, I have worked on the following:

1) I have continued working on the PI Placement and Interconnect system, which is being implemented primarily by Alan Sherman, Andy Moulton, and Susmita Sur. Andy Moulton has now finished his M.S. thesis on the power/ground routing phase. Alan is working on the placement and resizing code, and Susmita has been working the resizer as well.

2) I have been studying the theoretical aspects of low-level sensori-motor intelligence, and have been developing theories about how intelligence can be make sense of low-level inputs. The major themes are the importance of expectations and the treatment of expectations as "first-class" sensory inputs.

3) I have worked with Benny Chor to develop a new knapsack-type public-key cryptosystem, which seems to avoid the weaknesses present in earlier knapsack-type cryptosystems.

4) I have worked with Shafi Goldwasser and Silvio Micali on a new signature scheme, which is provably invulnerable to chosen-signature attacks. This result is somewhat paradoxical, since previous attempts to prove such invulnerability have actually led to weaknesses.

5) I have begun work with Laura Yedwab on formalizing the end-game in GO using Conway's "number system".

2.22. Alan Sherman

Under the supervision of Ronald Rivest, Alan Sherman has been exploring issues in the theoretical foundations of cryptographic strength. In particular, Sherman has been investigating formal definitions of cryptographic strength, relationships among algebraic and security properties of cryptographic functions, and the strength of multiple encryption. The goal of this research is to understand in a precise mathematical sense what makes cryptosystems secure.

In addition, Sherman has also been studying the problem of placing modules on a custom VLSI chip. As a member of the PI project, Sherman has developed and implemented an efficient heuristic placement algorithm based on a mincut strategy. More recently, Sherman has been working on provably good algorithms for determining the orientation of modules whose approximate placements are already known.

2.23. Michael Sipser

I am continuing research on methods for proving lower bounds on the computational complexity of combinatorial problems. I have been investigating the analytic sets as a potential infinite analog to NP, and the classical theorem stating that the analytic sets are not closed under complement as a means of shedding light on the NP vs. co-NP question. I can give a new, purely combinatorial proof of this theorem [21].

David Barrington, a graduate student in the mathematics department, and I are investigating finitary analogs to topological notions such as measure and category. Ravi Boppana, a graduate student in the EECS department, and I are studying problems in circuit complexity. Steve Trilling, a graduate student in the EECS department, and I are studying questions on the definition of random sequences and the random oracle hypothesis.

2.24. Steve Trilling

I am working with Mike Sipser on a complexity-based theory of randomness. The eventual hope is to find a suitable definition of "random sequence" and a set of conditions under which such a sequence could be efficiently computed. These conditions, if they are true, will be strong enough to imply non-trivial facts about complexity classes (i.e., perhaps $P = R$). Furthermore, the conditions should, hopefully, capture some intuitive notion of "random".

References

1. Amir, A. and Gabbay, D. "The Cover Theorem for Finite H-dimension Time Structures," to appear in *Theoretical Computer Science*, 1984.

2. Amir, A. "A Direct Linear-Time Planarity Test for Unflippable Modules," to appear in *Networks*, 1984.

3. Ajtai, M. "Σ^1_1-Formulae on Finite Structures," *Annals of Pure and Applied Logic*, 24 (1983) 1-48.

4. Boppana, R. "Threshold Functions and Bounded Depth Monotone Circuits," *16th ACM Symposium on Theory of Computing*, Washington, DC, 1984, 475-479.

5. Chor, B. and Goldreich, O. "RSA/Rabin Least Significant Bits are 1/Poly(log N) Secure," MIT/LCS-TM-260, MIT Laboratory for Computer Science, Cambridge, MA, May 1984.

6. Chor, B. and Rivest, R.L. " A Knapsack Type Public Key Cryptosystem Based on Finite Fields Arithmetic," to appear in *Proceedings of Crypto 84*, Santa Barbara, CA, August 1984.

7. Cosmadakis, S.S. and Kanellakis, P.C. "Functional and Inclusion Dependencies: A Graph Theoretic Approach," *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, April 1984, 24-37.

8. Dwork, C., Kanellakis, P. and Mitchell, J.C, "On the Sequential Nature of Unification, MIT/LCS/TM-257, MIT Laboratory for Computer Science, Cambridge, MA, March 1984.

9. Furst, M., Saxe J. B. and Sipser, M. "Parity, Circuits and the Polynomial-time Hierarchy," *22nd IEEE Symposium on Foundations of Computer Science*, 1981, 260-270.

10. Gandy R.O. "Proofs of Strong Normalization," in To H.B. Curry, Seldin and Hindley (eds.) 1980.

11. Goldwasser, S., Goldreich, O. and Micali, M. "How to Construct Random Functions," MIT/LCS/TM-244, MIT Laboratory for Computer Science, Cambridge, MA, November 1983.

12. Goldwasser, S. and Blum, M. "An *Efficient* Probabilistic Public-Key Encryption Scheme Which Hides All Partial Information," to be presented at Crypto '84, Santa Barbara, CA, August 1984.

13. Goldwasser, S. "A 'Paradoxical' Solution to the Signature Problem," with Silvio Micali and Ronald Rivest, *25th Symposium on the Foundations of Computer Science*, October 1984.

14. Goldwasser, S. "The Knowledge Computable from a Communication," with Silvio Micali and Charles Rackoff, in preparation.

15. Micali, S. "Contract Signing by Delayed Signatures," with Michael Ben-Or and Adi Shamir, in preparation.

16. Mitchell, J.C. "Coercion and Type Inference," *Proceedings of the Eleventh ACM Principals of Programming Languages Conference*, January 1984.

17. Mitchell, J.C. "Type Inference and Type Containment," International Symposium on the Semantics of Data Types, Sophia-Antipolis, France, to appear, 1984.

18. Mitchell, J.C. "Semantic Models of Second-order Lambda Calculus," submitted for conference presentation, 1984.

19. Moulton, A. S. "Routing the Power and Ground Wires on a VLSI Chip," M.S. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

20. Moulton, A/ S. "Laying the Power and Ground Wires on a VLSI Chip," *ACM IEEE 20th Design Automation Conference*, 1983, 754-755.

21. Sipser, M. "A Topological View of Some Problems in Complexity Theory," to appear in the *Proceedings of the Conference on Mathematical Science*, 1984.

22. Trakhtenbrot, B., Halpern, J. and Meyer, A. "From Denotational to Operational Semantics for Algol-like Languages: An Overview," in Logic of Programs, Proceedings, Clarke and Kozen (eds.) LNCS, Springer, 1983.

Publications

1. Boppana, R. "Threshold Functions and Bounded Depth Monotone Circuits," *16th ACM Symposium on Theory of Computing*, Washington, DC, 1984, 475-479.

2. Goldreich, O. "On Concurrent Identification Protocols," MIT/LCS/TM-250, MIT Laboratory for Computer Science, Cambridge, MA, December 1983.

3. Goldreich, O. "On the Number of Close-and-equal Bits in a String (With Implications on the Security of RSA's L.S.B.)," MIT/LCS/TM-256, MIT Laboratory for Computer Science, Cambridge, MA, March 1984.

4. Rivest, R.L. and Shamir, A. "How to Detect An Eavesdropper," *Communications of the ACM*, (1984).

5. Goldwasser, S. and Rackoff, C. "On Using the XOR Function as a Security Amplifier: Applications to Factoring Based Encryption," presented at Eurocrypt '84, Paris, France, April 1984.

6. Micali, S. "How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin," with Mike Luby and Charles Rackoff, *Proceedings 24th Annual IEEE Symposium on Foundations of Computer Science*, November 1983.

7. Micali, S. "How to Construct Random Functions," with Oded Goldreich and Shafi Goldwasser, MIT/LCS/TM-244, MIT Laboratory for Computer Science, Cambridge, MA, November 1983.

8. Micali, S. "An Oblivious Transfer Provably Equivalent to Factoring," with Michael Fischer and Charles Rackoff, presented at Eurocrypt '84, Paris, April 1984.

9. Micali, S. "The Knowledge Computable from a Communication," with Shafi Goldwasser and Charles Rackoff, in preparation.

10. Micali, S. "A 'Paradoxical' Solution to the Signature Problem," with Shafi Goldwasser and Ronald Rivest, *25th Symposium on the Foundations of Computer Science*, October 1984.

11. Kornhauser, D., Miller, G. and Spirakis, P. "Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications," to

appear in *Proceedings of the 25th Annual Symposium on the Foundations of Computer Science*, 1984.

12. Leiserson, C. and Saxe, J. "A Mixed-Integer Linear Programming Problem Which is Efficiently Solvable," *21st Allerton Conference on Communication, Control, and Computing*, October 1983.

13. Leiserson, C. "Systolic and Semisystolic Design," *IEEE International Conference on Computer Design: VLSI in Computers*, 627-632.

14. Leiserson, C. and Pinter, R. "Optimal Placement for River Routing," *SIAM Journal on Computing*, 12, 3, (August 1983) 447-462.

15. Gabor, P. "A Comparison of VLSI Design for Complex Multiplication," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA.

16. Bently, J., Johnson, D., Leighton T. and McGeoch, C. "An Experimental Study of Bin Packing," *Proceedings 21st Allerton Conference on Communication, Control and Computation*, October 1983.

17. Bently, J., Johnson, D., Leighton, T., McGeoch C. and McGeoch, L. "Some Unexpected Expected Behavior Results for Bin Packing," *Proceedings 16th Symposium on Theory of Computing*, May 1984, 279-288.

18. Chung, F.R.K., Leighton, T. and Rosenberg, A.L. "Diogenes: A Methodology for Designing Fault-Tolerant VLSI Processor Arrays," *Proceedings IEEE Symposium on Fault-Tolerant Computing*, June 1983, 26-31.

19. Karp, K., Leighton, T., Rivest, R., Thompson, C., Vazirani, U. and Vazirani, V. "Global Wire Routing in Two-Dimensional Arrays," *Proceedings 24th Conference on Foundations of Computer Science*, November 1983, 453-459.

20. Kleitman, D., Leighton, T., Lepley, M. and Miller, G. "An Asymptotically Optimal Layout for the Shuffle-Exchange Graph," *Journal of Computer and System Sciences*, 26, 3, (June 1983) 339-361.

21. Leighton, T. Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks, MIT Press, Cambridge, MA, September 1983.

messages required if communication is asynchronous [4]. However, the proof in [3] does not extend to the case of synchronous communication. It is, therefore, quite natural to ask whether the $\Omega(n \log n)$ lower bound can be extended to the synchronous case, or whether there are algorithms that somehow make use of the synchrony to limit the number of messages transmitted.

We obtain both positive and negative answers to our question of whether synchrony helps. On the one hand, we show that if processor id's are chosen from some countable set (such as the integers), then there is an algorithm which uses only O(n) messages in the worst case. The processors may initiate the algorithm at different rounds, and do not know the value of n. Unlike the earlier algorithms, our algorithm does not only use comparisons on id's - it uses the numerical value of the id's to count rounds. The number of synchronous rounds used by our algorithm can be very large in the worst case.

On the other hand, we show that both the departure from the comparison model and the possibility of using a large number of rounds are necessary in order to obtain a linear communication algorithm. More specifically, if the algorithm is restricted to use only comparisons of id's, then we obtain an $\Omega(n \log n)$ lower bound for the number of messages required in the worst case. Alternatively, if the number of rounds is required to be bounded by some t in the worst case, then there is a (very fast-growing) function f(n,t) which has the following very interesting property. If id's are chosen from any set T having at least f(n,t) elements, then any t-bounded algorithm requires $\Omega(n \log n)$ messages in the worst case. We achieve this result by giving a transformation from any algorithm to a comparison algorithm. Both our lower bound results hold even in the case that the number of processors in the ring is known to each processor, and all the processors are known to start at the same round.

6. DISTRIBUTED NETWORK ALGORITHMS

Shmuel Zaks has been working on design and analysis of algorithms for networks of processors, where no central controller is present and no common clock is available. The underlying communication networks usually dealt with are the simplest kinds: rings, trees and complete graphs.

The model usually used consists of a network of processors, each with a unique identity known initially only to itself. Communication is done via messages which arrive in order and after a finite delay, but no a priori bound for that delay is known. Assuming that the computation cost and the queueing cost in each processor is negligible compared to the communication cost, it is customary to measure the complexity of such algorithms by the total number of messages sent during any possible execution. Previous work that is related to the work described here

4.3. Communication Patterns in a Distributed System

Consensus problems arise in numerous guises and have traditionally been studied in several forms. In "Patterns of Communication in Consensus Protocols," Dwork and Skeen explore the relationships among several different versions of the consensus problem. To do this we define a new notion of reducibility. We first define for any protocol P a partial ordering on the message-sending steps of an execution of P. Intuitively, the sending of message m1 precedes the sending of m2 if and only if the contents of m1 may be known to the sender of m2 when m2 is sent. We call this partial ordering the *communication pattern* of the execution.

For any protocol Q, let the *scheme* of Q denote the set of all communication patterns of failure-free executions of Q. A problem may be characterized by the set of schemes of protocols for the problem. We say P1 *reduces to* P2, written P1 $\leq$ P2, if and only if the set of schemes for P1 contains the set of schemes for P2.

Intuitively, if P1 reduces to P2, then any protocol for P2 can solve P1 by relabeling local states and padding messages. Consequently, the message complexity (measured in number of messages) of P1 is not greater than the message complexity of P2. Our method of characterizing and comparing problems is the principal contribution of this paper. Given our taxonomy we use this notion of reducibility to examine the relationships among six practical problems with varying safeness and liveness properties.

This work will appear in this summer's Symposium on Principles of Distributed Computing.

5. ELECTION OF A LEADER IN A DISTRIBUTED RING OF PROCESSORS

Nancy Lynch collaborated with Greg Frederickson of Purdue on some work involving bounds on the number of messages required to elect a leader in a synchronous ring of processors. In this problem, there are n processors, which are identical except that each has its own unique identifier. At various points in time, one or more of the processors "wakes up" and initiate their participation in an algorithm to decide on a leader. The relevant resources for such a distributed computation are the total number of messages used and the amount of time expended from the time that the first processor wakes up.

This problem has been studied by many researchers [[4], [5], [34], etc.]. The best previous deterministic algorithms have used O(n log n) messages for either bidirectional or unidirectional rings. These algorithms work for both the synchronous and asynchronous models, and use comparisons of id's only. In addition, Burns has established a lower bound of Ω(n log n) on the number of

4.2. Byzantine Agreement

Brian Coan has been working on randomized Byzantine agreement algorithms. Such algorithms operate in a system of processes some of which can fail. The computation proceeds in a series of rounds in which messages are sent and received over a network that is fully connected and reliable. At each round, a process can use the toss of a coin when it decides on its future actions. Correct processes toss fair coins and send messages according to their programs. Failed processes can send arbitrary messages. Each process starts the algorithm with an input value. The goal is that after sending some messages each process will produce an answer. There are two requirements on the answer produced by a correct algorithm. If the same input was distributed to all processes, then this value must be the answer of all correct processes. In any event, all correct processes must agree on some common answer value.

Coan has developed a randomized Byzantine agreement algorithm that terminates in an expected number of rounds that is smaller than the known lower bound (due to Fischer and Lynch [17]) on the number of rounds required by a deterministic Byzantine agreement algorithm. Further improvements have been made in the algorithm by Chor. The algorithm is of interest for several reasons. It is simple and efficient enough to be of practical importance. Also, it is an example of a situation where randomization improves on the problem solving power of a system of computers. Although randomization is vital to the algorithm, it is used sparingly. The expected number of coin tosses per process is less than one. The algorithm incorporates some ideas from a deterministic Byzantine agreement algorithm developed previously by Turpin and Coan [38].

The best previously known randomized algorithm is due to Ben-Or [3]. It was primarily developed to operate in an asynchronous system, but also operates in a synchronous system. Ben-Or's algorithm has the disadvantage that it either requires a large number of rounds of communication or a large number of processes. In the synchronous case, the new algorithm improves on this substantially; although, unfortunately, the new algorithm appears not to generalize to the asynchronous case.

A related algorithm is due to M. Rabin [35]. Rabin's algorithm is more efficient than our new algorithm; however, it achieves this efficiency at the expense of requiring more powerful processes. In particular, his algorithm requires authentication and a global coin toss by means of Shamir's shared secret. The shared secret must be pre-computed by a trusted administrator. There are two disadvantages of this. It reduces the amount of autonomy of individual nodes and the shared secret represents an expensive resource that is partially consumed each time the algorithm is executed.

Results of [18] have shown that the problem of reaching distributed consensus in the presence of even a single faulty processor (even if it can only fail by stopping) is unsolvable in a completely asynchronous environment. Results of [9] have refined those of [18] by showing that the problem cannot be solved even in the case where only the communication (but not the processors) is asynchronous or in the case where only the processors are asynchronous. We considered the question of weakening the requirements so that (1) processors never reach disagreement, no matter how asynchronously the system runs, and (2) if there is ever some sufficiently long interval of time during which the system behaves synchronously, then agreement is guaranteed. We assume that the processors have no way of knowing when this interval occurs.

We have found an algorithm requiring 2f + 1 processors (where f is the maximum number of faulty processors) which solves this problem for the simplest kinds of failures - just stopping, or else omitting some messages. The algorithm is similar in general style to some "two-phase commit" algorithms in the literature. Our algorithm can be modified to tolerate Byzantine faults with authentication, using 3f + 1 processors, and to tolerate Byzantine faults without authentication, using 4f + 1 processors. (A more complicated and costly algorithm can also be designed for this latter case, using only 3f + 1 processors.) All of these bounds are tight.

All of the results described above were first proved for the case where the processors are always synchronous (i.e. have built-in "clocks" which go at the same rate) but communication is subject to variations in its synchrony. We later discovered that all of the results still hold for the case where both the processors and the communication system are subject to variations. The basic idea in the latter case is to have the processors implement fault-tolerant versions of Lamport's clock [28], and use those clocks in place of the built-in clocks, to control the execution.

We also considered the case where communication is assumed to be synchronous but processors are asynchronous. We have obtained algorithms and lower bounds for various types of faults in those cases as well. Most, but not all of these results are tight.

An interesting sidelight is that our algorithms also work under an alternative style of "partially synchronous" assumption. Namely, we might assume that there exist upper bounds on message delivery time (or on the ratio between speeds of processors), but that those bounds are not known to the processors themselves. If assumptions in this style are used in place of assumptions about intervals of synchronous behavior, the same algorithms still work.

This work will appear in this summer's Symposium on Principles of Distributed Computing.

The special structure of state-transition specifications can be exploited to obtain proof techniques. The *Correctness Theorem* gives sufficient conditions for an implementation to be correct with respect to state-transition specifications. A *Completeness Theorem* shows that correctness proofs of the form suggested by the Correctness Theorem exist for correct implementations, assuming the specifications satisfy certain well-formedness properties. A *Consistency Theorem* gives sufficient conditions for showing state-transition specifications to be consistent.

An important concept for structuring specifications and correctness proofs is the notion of *rely-* and *guarantee-conditions*. A rely-condition expresses conditions a module relies on its environment to provide, and a guarantee-condition expresses what the module guarantees to provide in return. Significant simplification in proofs of correctness can result if specifications are written so that what each module guarantees to its neighbor is exactly what the neighbor relies on the module to provide. Under these conditions, the rely- and guarantee-conditions "cut" the interdependence between modules in much the same way as a loop invariant cuts the dependence of one iteration of a loop of the preceding and succeeding iterations. In the thesis, a theorem is proved that shows how this idea can be captured in terms of a useful proof technique.

The techniques developed in the thesis are applied to obtain specifications and rigorous proofs of correctness for three example implementations: a *synchronizer module*, for synchronizing the access of a number of user processes to a single shared resource, a *resource manager module*, which allocates a finite set of resources in response to user requests, and a *message transmission module*, which uses the alternating bit protocol to construct a reliable message transmission service from an unreliable substrate.

4. DISTRIBUTED CONSENSUS

A large amount of effort went into various problems of distributed consensus this year. Nancy Lynch and Cynthia Dwork collaborated with Larry Stockmeyer on results about reaching consensus in a partially synchronous environment. Brian Coan obtained several results about Byzantine agreement. Cynthia Dwork collaborated with Dale Skeen in work on communication patterns for distributed consensus algorithms.

4.1. Reaching Consensus in a Partially Synchronous Environment

Cynthia Dwork, Nancy Lynch and Larry Stockmeyer of IBM San Jose collaborated in work involving the problem of reaching consensus in a partially synchronous environment.

3. FOUNDATIONS OF A THEORY OF SPECIFICATION FOR DISTRIBUTED SYSTEMS

Gene Stark's Ph.D. dissertation, entitled: "Foundations of a Theory of Specification for Distributed Systems," is near completion. This thesis accomplishes the following tasks: (1) the development of a mathematical model of distributed/concurrent computer systems, which incorporates as fundamental notions the concepts of *hierarchy* and *modularity*, (2) the use of the model as a basis for the investigation of a particular approach, called *state-transition specification*, to specifying and proving the correctness of distributed systems, and (3) the application of the state-transition specification and proof techniques to some nontrivial example problems. Important features of this work include: the integration of the notions of hierarchy and modularity into a theory of specification, the ability to specify both safety (invariance) and liveness (eventuality) properties of systems, and to reason about these properties in a single framework, and the development, as an integral part of the technique of state-transition specification, of a systematic method for constructing proofs of correctness.

The mathematical model provides the following primitive notions: An *event* is an instantaneous observable occurrence during the operation of a computer system. An *observation* is a record of the event occurrences that took place during a particular execution of a system. The *behavior* of a system is the set of all observations that can be produced by the system, in all its executions. *Abstraction maps* and *decomposition maps* capture formally the notions of hierarchy and modularity. An abstraction map maps an observation representing a detailed view of system execution into an observation representing a less detailed view of the same execution. A decomposition map is a vector of projections, each of which takes an observation for a "composite" system and localizes that observation to a particular component module. From these primitive notions, it is possible to define, in a specification-language-independent and programming-language-independent way, the notions of *implementation* and *correctness* of an implementation, as well as the notion of *consistency* of specifications.

The purpose of a specification is to describe a set of allowable observations for the module being specified. In a state-transition specification, this set of allowable observations is described by a pair $\langle M, V \rangle$, where M is a kind of nondeterministic machine that generates an observation as it executes, and V is a subset of the set of computations of M, called the set of *valid* computations. An observation is allowable if and only if it is produced in some valid computation. The state-transition approach appears to provide a natural, straightforward strategy for turning an intuitive understanding of the desired function of a module into a formal specification. Safety properties are incorporated into the description of the machine M, and liveness properties are captured by the set of valid computations V.

Our algorithm can be modified to allow a faulty process which has been repaired to synchronize its clock with the other nonfaulty processes. Let p be the process to be reintegrated into the system. We assume that p can awaken at an arbitrary time during an execution, perhaps during the middle of a round. It is necessary that p identify an appropriate round i at which it can obtain all the clock values from nonfaulty processes. Since p might awaken during the middle of a round, it will first orient itself by observing the arriving messages, allowing part of a round to pass before it begins to collect messages. After orienting itself, p continues to collect clock values during an interval of time long enough for p to hear from all nonfaulty processes. Immediately after p determines it has waited long enough, it carries out the same averaging procedure described earlier and determines a correction to its clock.

The problem solved by this algorithm is only that of maintaining synchronization of local times once it has been established. There is, of course, the separate problem of establishing such synchronization in the first place among processes whose clocks have arbitrary values. Our second algorithm is a variant of our first, and is used to establish the initial synchronization. We have also designed an interface between the two algorithms, since we envision the processes running our second algorithm until the desired degree of synchronization is obtained, and then switching to the maintenance algorithm.

The structure of the algorithm is similar to that of the algorithm which maintains synchronization. It runs in rounds. During each round, the processes exchange clock values and use the same fault-tolerant averaging function as before to calculate the corrections to their clocks. However, each round contains an additional phase, in which the processes exchange messages to decide that they are ready to begin the next round. This algorithm also synchronizes the clocks to within about 4ϵ.

Our algorithms will appear in this year's Symposium on Principles of Distributed Computing, in a special session devoted to clock synchronization.

Several results similar to last year's lower bound result [31] appeared this year in [10]. It became apparent that all of these results are addressing questions which are special cases of a single general problem: that of determining possible closeness of synchronization for arbitrary network graphs with arbitrary communication uncertainties. With Joe Halpern, we spent some time attempting to solve the general problem, but have so far not obtained any interesting results.

Mandy Ng has simulated our clock synchronization algorithm as a UROP project.

timer to go off at a specified time in the future. Formally, timers are treated similarly to messages between processes. The system is interrupt-driven in that a process only takes a step when a message (i.e., a timer or a message from another process) arrives. We consider only the case where the message network is completely connected. All messages are delivered within a fixed amount of time plus or minus some uncertainty.

Keeping the local times of processes in a distributed system synchronized in the presence of arbitrary faults is important in many applications and is an interesting problem in its own right. Taking into account the clocks' drift from real time and varying message delivery times makes the problem more realistic and more challenging. In order to be truly useful, a solution to this problem must allow faulty processes that have recovered to be reintegrated into the system. Our first algorithm meets these requirements, assuming that the clocks are initially close together and that fewer than one third of the processes are faulty. (A result in [10] indicates that this proportion of faulty processes is the largest that any algorithm, which does not use unforgeable digital signatures, can tolerate.)

Our algorithm runs in rounds (as do those in [29], [24] and [33]), resynchronizing at each round to correct for the clocks drifting out of synchrony. The size of the adjustment made to a clock at each round is independent of the number of faulty processes. At each round, n^2 messages are required, where n is the total number of processes. The closeness of synchronization achieved depends only on the initial closeness of synchronization, the message delivery time and its uncertainty, and the drift rate. We give explicit bounds on how the difference between the clock values and real time grows.

At the beginning of each round, every nonfaulty process broadcasts its clock value and then waits a bounded amount of time on its clock, long enough to ensure that clock values are received from all nonfaulty processes. After waiting, the process averages the arrival times of all the messages received, using a fault-tolerant averaging function. The resulting average is used to calculate an adjustment to the process' clock.

The fault-tolerant averaging function is derived from those used in [12] for reaching approximate agreement. The function is designed to be immune to some fixed maximum number, f, of faults. It first throws out the f highest and f lowest values, and then applies some ordinary averaging function to the remaining values. We choose the midpoint of the range of the remaining values, to be specific. The properties of the fault-tolerant averaging function allow the distance between the clocks to be halved (roughly) at each round. Consequently, the averaging function can be considered the heart of the algorithm.

This algorithm can maintain a closeness of synchronization of approximately 4ε, where ε is the uncertainty in the message delivery time.

1. OVERVIEW

This year, the Theory of Distributed Systems Group worked a large variety of individual problems within the theory of distributed computing. Particular emphasis was placed on problems and models involving time, and on reliability issues. Specifically, we studied the following problems:

(1) Software clock synchronization,

(2) Foundations of a theory of specification for distributed systems,

(3) Distributed consensus,

(4) Election of a leader in a distributed ring of processors,

(5) Distributed network algorithms,

(6) Diagnosis of faulty components, and

(7) Distributed network resource allocation.

In addition, other work, not directly related to theory of distributed computing, included:

(8) Unification,

(9) Combinatorics and graph algorithms, and

(10) Development of a CLU parser-generator.

2. SOFTWARE CLOCK SYNCHRONIZATION

Jennifer Lundelius and Nancy Lynch have continued work during the past year on the problem of synchronizing clocks in a distributed system. The formal model developed earlier to describe systems of distributed processes with local clocks has been polished (and a simple "language" for describing processes in this model has been developed). Last year's major result, a lower bound on the closeness with which clocks can be synchronized in the presence of uncertainty of message delay, has been written up this year and submitted for publication [31]. This year's major results are two new fault-tolerant algorithms, one to maintain synchronization among processes whose clocks initially are close together, and another to establish synchronization in the first place.

In our model, each process has a read-only physical clock. By adding a correction to the physical clock time, the process obtains a local time. The process can set a

THEORY OF DISTRIBUTED SYSTEMS

Academic Staff

N. A. Lynch, Group Leader

Graduate Students

B. Coan
J. Lundelius
E. Stark

Undergraduate Students

M. Ng
N. Savasta

Support Staff

E. Pothier

Visitors

B. Awerbuch
J. Burns
C. Dwork
S. Zaks

19. Micali, S. "The Knowledge Computable from a Communication,"
University of California, Berkeley, March 1984
University of Chicago, March 1984
MIT, Laboratory for Computer Science, May 1984
Harvard University, May 1984

20. Maley, M. "VLSI Routing of Planar Interconnections," VLSI Research Review, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, May 1984.

MIT, Cambridge, MA, March 1984.
AT&T Bell Labs, Homdale, NJ, March 1984.
IBM, Yorktown Heights, NY, April 1984.
Xerox PARC. Palo Alto, CA, April 1984.
IBM, San Jose, CA, April 1984.

10. Rivest, R.L. "Reflections on Artificial Intelligence," Stanford University, Stanford, CA, 1984.

11. Rivest, R.L. "On the Crossing Placement Problem," Stanford University, Stanford, CA, 1984.

12. Rivest, R.L. "Estimating a Probability Using Finite Memory," Stanford University, Stanford, CA, 1984.

13. Rivest, R.L. "An Overview of Special-Purpose VLSI Implementations of the RSA Cryptosystem," CRYPTO 84 Conference , Paris, France, 1984.

14. Sherman, A. "The PI System's Algorithms for Placing Modules on a Custom VLSI Chip," MIT VLSI Research Review, Cambridge, MA, December 17, 1983.

15. Sherman, A. "Basic: A Dangerous Impediment to Computer Literacy," presented to the PRISM program for the gifted and talented at Lafayette High School, Williamsburg, VA March 30, 1984,

16. Micali, S. "How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin,"
    Universitat Zurich, Institut fur Angewandte Mathematik, August 1983
    24th FOCS Conference, Tucson, AZ, November 1983
    University of Chicago, March 1984

17. Micali, S. "How to Construct Random Functions,"
    University of Toronto, December 1983
    University of Rome, January 1984
    University of Pittsburgh, February 1984
    University of California, March 1984
    IBM, San Jose, CA, March 1984
    University of Chicago, March 1984

18. Micali, S. "An Oblivious Transfer Provably Equivalent to Factoring," Eurocrypt '84, Paris, France, April 1984

2. Goldwasser, S. "An Efficient Probabilistic Encryption Scheme," Yale University, May 1984.

3. Goldwasser, S. "Strong Signature Schemes,"

   University of California at Berkeley, June 1983
   Carnegie-Mellon University, October 1983
   Massachusetts Institute of Technology, October 1983
   The Technion, Haifa, Israel, January 1984

4. Goldwasser, S. "Digital Signatures and Authentication," IBM/MIT Joint Conference on Security, Cambridge, MA, May 1984.

5. Goldwasser, S. "How to Construct Random Functions,"

   Weizmann Institute, Rehovot, Israel, January 1984
   Hebrew University, Jerusalem, Israel, January 1984
   Massachusetts Institute of Technology, February 1984
   IBM-Yorktown Heights, May 1984
   Yale University, May 1984
   Harvard University, May 1984
   University of Washington, Seattle, June 1984
   Stanford University, June 1984

6. Goldwasser, S. "Knowledge Complexity,"

   Yale University, June 1983
   IBM-San Jose, June 1984

7. Goldwasser, S. "On Using the XOR Function as a Security Amplifier: Applications to Factoring Based Encryption,"

   Massachusetts Institute of Technology, November 1983
   Eurocrypt '84, Paris, France, April 1984

8. Goldwasser, S. "Use of Cryptography in Today's Technology," North-East Regional ACM Conference on Integrating the Workplace, Lowell, MA, March 1984.

9. Mitchell, J. "Types in Lambda Calculus and Other Programming Languages,"

22. Leighton, T. "Circulants and the Characterization of Vertex-Transitive Graphs," *Journal Research National Bureau of Standards*, 88, 6, (November-December 1983), 395-402.

23. Leighton, T. "On the Decomposition of Vertex-Transitive Graphs into Multicycles, *Journal Research National Bureau of Standards*, 88, 6, (November-December 1983), 403-410.

24. Leighton, T. "New Lower Bound Techniques for VLSI," *Mathematical Systems Theory*, 17, 1, (April 1984), 47-70.

25. Leighton, T. "Parallel Computation Using Meshes of Trees," *Proceedings 1983 International Workshop on Graphtheoretic Concepts in Computer Science*, M. Nagl and J. Perl (eds.), Trauner-Verlag, Linz, West Germany, 1984, 200-218.

26. Leighton, T. "Tight Bounds on the Complexity of Parallel Sorting," *Proceedings 16th Symposium on Theory of Computing*, May 1984, 71-80.

27. Leighton, T. and Rivest, R. "Estimating a Probability Using Finite Memory," *Proceedings of the Conference on the Foundations of Computation Theory*, August 1983, Lecture Notes in Computer Science Series # 158, 255-269.

28. Leighton T. and Rosenberg, A. "Automatic Generation of Three-Dimensional Circuit Layouts," *Proceedings IEEE Conference on Computer Design*, November 1983, 633-636.

29. Bui, T., Chaudhuri, S., Leighton, T. and Sipser, M. "Graph Bisection Algorithms with Good Average Case Behavior," Submitted to the 25th FOCS Conference.

Thesis in Progress

1. Maley, M. "Routing and Compaction of Planar VLSI Layouts," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected September 1984.

Talks

1. Kornhauser, D. "Generalized '15-puzzles' and the Diameter of Permutation Groups," MIT Laboratory for Computer Science, Cambridge, MA, April 19, 1984.

includes that of Burns [4], Dolev et al. [11], Frederickson and Lynch [20], Gallager et al. [22] and Peterson [34].

Several problems that require reaching common knowledge have been studied. The problems considered include that of deciding distinctness and that of finding a majority value in the set of initial values. Lower bounds are derived in [21], primarily for ring networks. Following Frederickson and Lynch [20], we consider comparison algorithms to obtain lower bounds on message complexity for these problems. For unrestricted algorithms the bit complexity is usually of interest. We study it for the distinctness problem, using the technique developed in Yao [39]. The problems considered are quite fundamental: algorithms that achieve common knowledge often involve one of them. Thus, it is believed that this study will help our understanding of the complexity of other problems on various networks.

The sorting problem has been extensively studied for sequential algorithms; in Zaks [40] such a study is made for distributed networks. We study the problems of sorting and ranking processors that have (not necessarily distinct) initial values. Assuming a tree network, an algorithm for the ranking problem is presented that uses, in the worst case, a number of messages that is shown to be best possible. The algorithm is then extended to perform sorting, using again the fewest number of messages in the worst case. The expected behavior of these algorithms is analyzed for various classes of trees. The ranking algorithm is an improvement upon the algorithm in Korach et al. [27] in its worst case performance, its average case performance, and its use of concurrency.

Other work has dealt with distributed algorithms for a complete network. For such algorithms, it is generally assumed that a node cannot distinguish between its incident edges. In Korach et al. [26], [25] this model is used, and finding a spanning tree is shown to require strictly fewer messages than finding a minimum spanning tree (thus partially answering a question raised by Gallager [22]). In Santoro et al. [37] we further investigate the difference between the problem of finding a minimum spanning tree and finding any spanning tree when the underlying network is complete. We show that in a certain model of a complete network - where the processors have a partial information of where the edges are leading to - the distinction between these two problems is even more profound; namely, spanning tree is shown to be as easy as possible (linear in the size of the network), while minimum spanning tree is shown to be as hard as possible (quadratic in the size).

Some of the preceding work [26] will be presented at this summer's Symposium on Principles of Distributed Computing.

Baruch Awerbuch has been working on the problem of breadth-first search (BFS) in a distributed network. In a synchronous network there exists a fairly simple algorithm which performs breadth-first search in O(V) time and O(E) messages.

Surprisingly enough, performing BFS in an asynchronous network turns out to be a much more difficult task. Awerbuch [2] proposes to implement BFS by synchronizing the network and using a synchronous algorithm. This approach yields O(V**2) messages and O(V log V) time; its communication cost may be high for sparse networks. The algorithm due to Fredrickson [19] requires O(VE**1/2) both in terms of messages and time.

Awerbuch has recently found a new distributed BFS algorithm, which requires just O(E + V**1.6) messages and time. The idea of the algorithm is to use partial synchronization and to change the degree of synchronization according to the density of the network. An open question is whether this result can be further improved.

7. DIAGNOSIS OF FAULTY COMPONENTS

Nancy Lynch has been collaborating with researchers at Draper Laboratories, on

problems of fault-tolerance arising in real-time processing systems with high reliability requirements. Among the problems considered was a problem of diagnosing and replacing faulty components in a long-lived system, where the symptoms of faults arise for pairs of components. (For example, a transmitter-receiver pair may fail to complete a transmission successfully, but it might not be known which component is faulty.)

The problem turns out to be formalizable as an on-line version of the vertex cover problem. It has been shown that a trivial deterministic algorithm exists which can achieve a worst-case ratio of two between the number of components replaced and the number actually faulty. Moreover, no deterministic or probabilistic algorithm can do any better. (That is, no probabilistic algorithm can achieve an expected ratio better than two, where the expectation is taken over the probabilistic choices in the algorithm only.)

On the other hand, if reasonable probabilistic assumptions are made about the patterns of occurrence of faults, then a ratio near 3/2 is achievable.

8. DISTRIBUTED NETWORK RESOURCE ALLOCATION

Nancy Lynch collaborated with Mike Fischer and Nancy Griffeth in completing some older work on optimal allocation of resources in a tree network. There are two sets of results. The first [15] deals with optimal placements of resources within a tree; the placements are intended to optimize the expected cost of a minimum matching between the resources and requests arriving according to a probability distribution. A new result obtained this year is an upper bound on the expected cost

of certain near-optimal placements within arbitrary trees. (Previously, we only had results for balanced trees.)

The second [16] is a difficult probabilistic analysis of a distributed algorithm which matches arriving requests to resources placed within the network. This analysis has been polished up.

The following work, while not directly related to the theory of distributed computing, was carried out within our research group.

9. UNIFICATION

Cynthia Dwork has collaborated with John Mitchell and Paris Kanellakis in work on unification. *Unification of terms* is an important step in resolution theorem proving [36] with applications to a variety of symbolic computation problems. In particular, unification is used in PROLOG interpreters [6], type inference algorithms [33], and term rewriting systems [23].

Informally, two symbolic terms s and t are unifiable if there is some way of substituting additional terms for variables in s and t so that both become the same term. All occurrences of a variable x in both s and t must be replaced by the same term. For example, the terms f(x,x) and f(g(y),g(g(z))) may be unified by substituting g(z) for y and g(g(z)) for x. A unification problem like "unify f(t1,t2) and f(t3,t4)" may be decomposed into two subproblems "unify t1 and t3" and "unify t2 and t4". However, these two problems cannot be solved entirely separately in parallel. If some variable x occurs in both t1 and t4, for example, then the solutions to the subproblems must be coordinated so that both substitute the same term for x.

There are several variations of the unification problem. For example, a type inference algorithm may construct labeled graphs which represent terms that must be unified. An acceptable result of unification, in this case, may be a labeled graph with a cycle. Labeled graphs with cycles represent types defined by recursion [32], or, if interpreted as terms, represent "infinite terms" to be substituted for variables. Using the infinite term f(f(f(...))), we can unify x and f(x), something we could not do otherwise. Unrestricted unification also appears in many PROLOG interpreters; those omitting the *occur* test [6]. Another variation on unification is the special case in which the labeled graphs are from a class of tree-like directed acyclic graphs. The complexity of unification on tree-like graphs is precisely the complexity of unification on symbolic representations of terms, as opposed to the complexity as a function of the size of more concise graph representations. For this case it was known that unification is hard for co-NLOGSPACE [30]. This did not exclude the possibility of fast parallel algorithms (i.e., algorithms requiring time poly-logarithmic in the length of the input). Moreover, no lower bound was known for unrestricted unification. In

"On the Sequential Nature of Unification", Dwork, Kanellakis, and Mitchell ([14]) show that all of the above variants of unification are log-space complete for P, and hence are unlikely to have fast parallel solutions.

One important special case of unification can be solved quickly in parallel. This problem, called *term matching*, arises in term rewriting. A term s matches a term t if t is a substitution instance of s. The rewrite rule $l \rightarrow r$ may be used to rewrite a term t whenever l matches t. [14] shows that matching can be accomplished in $\log^2 n$ time on a PRAM, where n is the length of the input, using a number of processors polynomial in n. The algorithm combines parallel transitive closure of a directed acyclic graph with parallel computation of connected components of an undirected graph. Also, matching is in NLOGSPACE, and for tree-like graphs it is in DLOGSPACE.

10. COMBINATORICS AND GRAPH ALGORITHMS

Shmuel Zaks has worked on several problems involving combinatorics and graph algorithms. Revisions have been made in [41] and [7]. The paper [41] presents a new algorithm for generating all the permutations of the numbers 1,...,n; it generates the next permutation by reversing a certain suffix of its predecessor. In [7] a very general enumeration formula for occurrences of patterns in certain families of trees (like ordered trees or binary trees) is derived, using an extension of the Cycle Lemma of Dvoretzky and Motzkin [13]. Many known results in this area fit within its framework. This formula is very handy in analyzing expected-time behaviors of algorithms acting on these families of trees.

Using a new correspondence between ordered trees and non-crossing partitions, and using results previously stated in terms of trees, some notes are made in Dershowitz and Zaks [8] regarding the set of non-crossing partitions.

A planarity test for unflippable modules is developed in Zaks [42]; it improves upon a previous algorithm of Amir [1] in two aspects: it does not require any additional memory, and it can be easily implemented by a parallel algorithm.

11. A CLU PARSER-GENERATOR

Neil Savasta wrote a new parser-generator program, patterned after the widely-used C program YACC. This new program was integrated into the undergraduate compiler course this spring.

12. PLANS

The Theory of Distributed Systems Group plans to continue working on problems involving reliability and time in distributed systems, as well as problems involving particular high-level constructs for reliable distributed computing.

Plans for the coming year include visits by Drs. Baruch Awerbuch, James Burns, Cynthia Dwork, Paris Kanellakis and Michael Merritt.

References

1. Amir, A. "A Direct Linear-time Planarity Test for Unflippable Modules," MIT Laboratory for Computer Science, Cambridge, MA, May 1984.

2. Awerbuch, B. "Efficient Network Synchronization Protocol," *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, Washington, D C, April 30-May 2, 1984.

3. Ben-Or, M. "Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols," *Proceedings of Second Annual ACM SIGACT-SIGOPS Agreement Protocols, Proceedings of Second Annual Symposium on Principles of Distributed Computing*, August 17-19, 1983.

4. Burns, J. E. "A Formal Model for Message Passing Systems," Technical Report 91, Indiana University, September 1980.

5. Chang, E. and Roberts, R. "An Improved Algorithm for Decentralized Extrema-finding in Circular Configurations of Processes," (1979) 281-283.

6. Clocksin, W.F. and Mellish, C.S. "Programming in Prolog," Springer-Verlag (1981).

7. Dershowitz, N. and Zaks, S. "Patterns in Trees," *Proceedings of the 9th Colloquium on Trees in Algebra and Programming*, Bordeaux, France, March 1984.

8. Dershowitz, N. and Zaks, S. "Ordered Trees and Non-crossing Partitions," to appear.

9. Dolev, D., Dwork, C. and Stockmeyer, L. "On the Minimal Synchronism Needed for Distributed Consensus," *23rd Symposium on the Foundations of Computer Science*, November 1983.

10. Dolev, D., Halpern, J. and Strong, R. "On the Possibility and Impossibility of Achieving Clock Synchronization," *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, Washington, DC, April 30-May 2, 1984.

11. Dolev, D., Klawe, M. and Rodeh, M. "An O(nlogn) Unidirectional Distributed Algorithm for Extrema Finding in a Circle," *Journal of Algorithms*, 3 (1982), 245-260.

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

13. Dvoretzky, A. and Motzkin, T. "A Problem of Arrangements," *Duke Mathematical Journal*, 14, 305-313.

14. Dwork, C., Kanellakis, P. and Mitchell, J. "On the Sequential Nature of Unification," to appear in *Journal of Logic Programming*, 1, 1 (1984).

15. Fischer, M., Griffeth, N. Guibas, L. and Lynch, N. "Optimal Placement of Identical Resources in a Distributed Network," *Proceedings of 2nd International Conference on Distributed Computing*, 1981

16. Fischer, M., Griffeth, N., Guibas, L. and Lynch, N. "Probabilistic Analysis of a Network Resource Allocation Algorithm," *AMS Workshop on Probabilistic Algorithms* (Abstract Only), June 1982.

17. Fischer, M. and Lynch, N. A. "A Lower Bound for the Time to Assure Interactive Consistency," *Information Processing Letters*, 14, 4 (June 1982), 183-186.

18. Fischer, M., Lynch, N. and Paterson, M. "Impossibility of Distributed Consensus with One Faulty Process," MIT/LCS/TR-282, MIT Laboratory for Computer Science, Cambridge, MA.

19. Frederickson, G. N. "A Single Source Shortest Path Algorithm for a Planar Distributed Network," to appear in *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, August 27-29, 1984, Vancouver, BC, Canada.

20. Frederickson, G. N. and Lynch, N. A. "The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring," *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, Washington DC, April 30 - May 2, 1984, 493-503.

21. Gafni, E., Loui, M. C., Tiwari, P., West, D. and Zaks, S. "Lower Bounds on Common Knowledge in Distributed Algorithms," to appear.

22. Gallager, R. G., Humblet, P. A. and Spira, P. M. "A Distributed Algorithm for Minimum Spanning Tree," *Transactions of Programming Languages and Systems*, 5, 1 (1983), 66-77.

23. Guttag, J.V., Kapur, D. and Musser, D.R. "On Proving Uniform Termination and Restricted Termination of Rewriting Systems," *SIAM J. Computing*, 12, 1 (1983), 189-214.

24. Halpern, J., Simons, B. and Strong, R. "Fault-tolerant Clock Synchronization," to appear in *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, August 27-29, 1984, Vancouver, BC, Canada.

25. Korach, E., Moran, S. and Zaks, S. "Finding a Minimum Spanning Tree Can be Harder than Finding a Spanning Tree in Distributed Networks," Technical Report-126, IBM Scientific Center, Haifa, Israel, November 1983.

26. Korach, E., Moran, S. and Zaks, S. "Tight Lower and Upper Bounds for Some Distributed Algorithms for a Complete Network of Processors," Technical Report-124, IBM Scientific Center, Haifa, Israel, November 1983, to appear in *Proceedings of the 3rd Annual ACM Symposium on Principles of Distributed Computing*, August 27-29, 1984, Vancouver, BC, Canada.

27. Korach, E., Rotem, D. and Santoro, N. "Distributed Algorithms for Ranking the Nodes of a Network," *Proceedings of the 13th Southeastern Conference on Combinatorics, Graph Theory and Computing*, 1982, 235-246.

28. Lamport, L. "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, 21, 7 (July 1978).

29. Lamport, L. and Melliar-Smith, P. M. "Synchronizing Clocks in the Presence of Faults," SRI International Report, (March 1982).

30. Lewis, H. R. and Statman, R. "Unifiability is Complete for Co-NLOGSPACE," *IPL*, 15, 5 (1982), 220-222.

31. Lundelius, J. and Lynch, N. "A New Fault-tolerant Algorithm for Clock Synchronization," to appear.

32. MacQueen, D., Plotkin, G. and Sethi, R. "An Ideal Model for Recursive Polymorphic Types," *Proceedings 1984 Principles of Programming Languages*.

33. Marzullo, K. "Loosely-Coupled Distributed Services: A Distributed Time Service," Ph.D. dissertation, Stanford University, Stanford, CA, 1983.

34. Peterson, G. L. "An O(nlogn) Unidirectional Algorithm for the Circular Extrema Problem," *Transactions of Programming Languages and Systems*, 4 (1982), 758-762.

35. Rabin, M. "Randomized Byzantine Generals," *Proceedings 24th Annual Symposium on Foundations of Computer Science*, November 7-9, 1983.

36. Robinson, J. A. "A Machine Oriented Logic Based on the Resolution Principle," *Journal of the ACM*, 12, 1 (1965), 23-41.

37. Santoro, N., Urrotia, J. and Zaks, S. "Distributed Algorithms for Spanning Trees and Minimum Spanning Trees in a Complete Network with a Weak Sense of Orientation," to appear.

38. Turpin, R. and Coan, B. "Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement," *Information Processing Letters*, 18, 2 (February 1984), 73-76.

39. Yao, A. C. "Some Complexity Questions Related to Distributive Computing," *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, New York, 1979, 209-213.

40. Zaks, S. "Optimal Distributed Algorithms for Ranking and Sorting," to appear.

41. Zaks, S. "A New Algorithm for Generation of Permutations," to appear in *BIT*.

42. Zaks, S. "A Simple Linear-time Planarity Test for Unflippable Modules," to appear.

Publications

1. Dershowitz, N. and Zaks, S. "Ordered Trees and Non-crossing Partitions," to appear.

2. Dershowitz, N. and Zaks, S. "Patterns in Trees," *Proceedings of the 9th Colloquium on Trees in Algebra and Programming*, Bordeaux, France, March 1984.

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

4. Dwork, C., Lynch, N. and Stockmeyer, L. "Consensus in the Presence of Partial Synchrony," to appear in *Proceedings of the 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 27-29, 1984, Vancouver, BC, Canada.

5. Dwork, C. and Skeen, D. "Patterns of Communication in Consensus Protocols," to appear in *Proceedings of the 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 27-29, 1984, Vancouver, BC, Canada.

6. Dwork, C., Kanellakis, P. and Mitchell, J. "On the Sequential Nature of Unification," to appear in *Journal of Logic Programming*, 1, 1 (1984).

7. Fischer, M., Griffeth, N. and Lynch, N. "Optimal Placement of Identical Resources in a Distributed Network," to appear in *Information and Control.*

8. Fischer, M., Griffeth, N., Guibas, L. and Lynch, N. "Probabilistic Analysis of a Network Resource Allocation Algorithm," to appear in *Information and Control.*

9. Fischer, M., Lynch, N. and Paterson, M. "Impossibility of Distributed Consensus with One Faulty Process," to appear in *Journal of the ACM.*

10. Frederickson, G.N. and Lynch, N.A. "The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring," *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*, Washington, DC, April 30-May 2, 1984, 493-503.

11. Gafni, E., Loui, M. C., Tiwari, P., West, D. and Zaks, S. "Lower Bounds on Common Knowledge in Distributed Algorithms," to appear.

12. Lundelius, J. and Lynch, N. "A New Fault-tolerant Algorithm for Clock Synchronization," to appear in *Proceedings of the 3rd ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, August 27-29, 1984, Vancouver, BC, Canada.

13. Lundelius, J. and Lynch, N. "An Upper and Lower Bound for Clock Synchronization," to appear in *Information and Control.*

14. Lynch, N. "Concurrency Control for Resilient Nested Transactions," to appear.

15. Lynch, N., Arjomandi, E. and Fischer, M. "Efficiency of Synchronous Versus Asynchronous Distributed Systems," *Journal of the Association For Computing Machinery*, 30, 3 (July 1983), 449-456.

16. Lynch, N.A. and Fischer, M.J. "A Technique for Decomposing Algorithms Which Use a Single Shared Variable," *Journal of Computer and System Sciences*, 27, 3 (December 1983) 350-377.

17. Santoro, N., Urrotia, J. and Zaks, S. "Distributed Algorithms for Spanning Trees and Minimum Spanning Trees in a Complete Network with a Weak Sense of Orientation," to appear.

18. Turpin, R. and Coan, B. "Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement," *Information Processing Letters*, 18, 2 (February 1984), 73-76.

19. Zaks, S. "A New Algorithm for Generation of Permutations," to appear in *BIT*.

20. Zaks, S. "Optimal Distributed Algorithms for Ranking and Sorting," to appear.

Thesis Completed

1. Savasta, N. "Implementation of a Compiler-compiler," S.B. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, June 1984.

Theses in Progress

1. Coan, B. "Some Fundamental Problems in Distributed Systems," Ph.D. dissertation, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected May 1986.

2. Lundelius, J. "Synchronizing Clocks in a Distributed System," S.M. thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA, expected August 1984.

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, expected August 1984.

Talks

1. Dwork, C. "The Inherent Cost of Nonblocking Commitment," *2nd Symposium on the Principles of Distributed Computing*, August 1983.

2. Dwork, C. "On the Minimal Synchronism Needed for Distributed Consensus," *23rd Symposium on the Foundations of Computer Science*, November 1983.

3. Dwork, C. "The Consensus Game,"
   University of California at Berkeley, Fall 1983;
   University of Washington, Seattle, Fall 1983;
   IBM Research, San Jose, CA, Fall 1983;
   New York University, Fall 1983;

4. Dwork, C. "Consensus in the Presence of Partial Synchrony,"
   AT&T Bell Laboratories, Murray Hill, NJ.
   IBM Research, Yorktown Heights, NY.

5. Lynch, N. A. "The Impact of Synchronous Communication on the Problem of Electing a Leader in a Ring,"

6. Brown University, Fall, 1983;
   Georgia Institute of Technology, Winter, 1984;
   Computer Corporation of America, Spring, 1984;
   Symposium on Theory of Computing, Spring, 1984;

7. Stark, E.W. "Reaching Approximate Agreement in the Presence of Faults," *IEEE Symposium on Reliability in Distributed Software and Database Systems*, October 1983.

8. Stark, E.W. "Assigning Meaning to Specifications,"
   University of Massachussets, Amherst, MA, Spring 1984;
   AT&T Bell Laboratories, Murray Hill, NJ, Spring 1984;
   Tufts University, Medford, MA, Spring 1984;
   State University of New York, Stony Brook, NY,
   Spring 1984;
   Cornell University Spring 1984;
   University of Michigan Spring 1984;
   University of Texas, Austin, TX, Spring 1984;

9. Zaks, S. "Tight Lower and Upper Bounds for Distributed Algorithms for a Complete Network of Processors,"

Harvard University, Cambridge, MA, Fall, 1983
University of Illinois at Urbana-Champaign, Fall, 1983
Brandeis University, Fall, 1983
MIT, Cambridge, MA, Spring, 1984
IBM - Thomas J. Watson Research Center, Spring, 1984
G.E. Schenectady Research Center, Spring, 1984
University of Toronto, Spring, 1984
Carleton University, Spring, 1984

10. Zaks, S. "The Cycle Lemma and Ordered Trees," MIT, Cambridge, MA, Fall, 1983.

PUBLICATIONS

Technical Memoranda

TM-10[4] 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

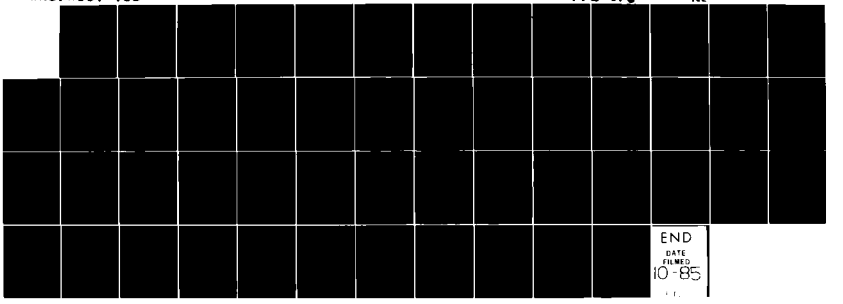
TM-19 Fenichel, R.R.
A New List-Tracing Algorithm, October 1970, AD 714-522

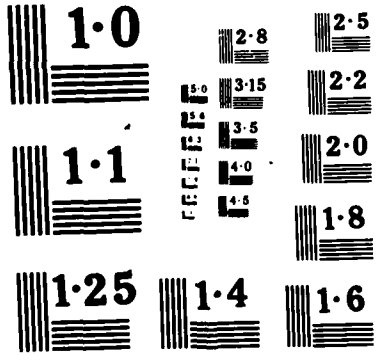
---

[4] TMs 1-9 were never issued.

AD-A158 299 LABORATORY FOR COMPUTER SCIENCE PROGRESS REPORT 21 JULY 83-JUNE 84(U) MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE M DERTOUZOS JUN 84 4/4

UNCLASSIFIED F/G 9/2 NL




1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0

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

PUBLICATIONS

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 $T = O(2^{n/2})$, $S = O(2^{n/4})$ 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

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

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

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.

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

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

Technical Reports

TR-1[5]	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

[5]TRs 5, 9, 10, 15 were never issued

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

TM-262 Leighton, T. and Rosenberg, A. Three-dimensional Circuit Layouts, June 1984

PUBLICATIONS

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-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

PUBLICATIONS

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-194 Barendregt, Henk and Giuseppe Longo
Recursion Theoretic Operators and Morphisms on Numbered Sets, February 1981

TM-195 Barber, Gerald R.
Record of the Workshop on Research in Office Semantics, February 1981

TM-196 Bhatt, Sandeep N.
On Concentration and Connection Networks, S.M. Thesis, EE & CS Dept., March 1981

TM-197 Fredkin, Edward and Toffoli Thomaso
Conservative Logic, May 1981

TM-198 Halpern, Joseph 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

PUBLICATIONS

TM-182 Itai, Alon, Christos H. Papadimitriou and Jayme Luiz Szwarefiter
Hamilton Paths in Grid Graphs, October 1980

TM-183 Meyer, Albert R.
A Note on the Length of Craig's Interpolants, October 1980

TM-184 Lieberman, Henry and Carl Hewitt
A Real Time Garbage Collector that can Recover Temporary Storage Quickly, October 1980

TM-185 Kung, Hsing-Tsung and Christos H. Papadimitriou
An Optimality Theory of Concurrency Control for Databases, November 1980, AD A092-625

TM-186 Szolovits, Peter and William A. Martin
BRAND X Manual, November 1980, AD A093-041

TM-187 Fischer, Michael J., Albert R. Meyer and Michael S. Paterson
CapOmega()(n log n) Lower Bounds on Length of Boolean Formulas, November 1980

TM-188 Mayr, Ernst
An Effective Representation of the Reachability Set of Persistent Petri Nets, January 1981

TM-189 Mayr, Ernst
Persistence of Vector Replacement Systems is Decidable, January 1981

TM-190 Ben-Ari, Mordechai, Joseph Y. Halpern and Amir Pnueli
Deterministic Propositional Dynamic Logic: Finite Models, Complexity, and Completeness, January 1981.

TM-191 Parikh, Rohit
Propositional Dynamic Logics of Programs: A Survey, January 198l.

TM-192 Meyer, Albert R., Robert S. Streett and Grazina Mirkowska
The Deducibility Problem in Propositional Dynamic Logic, February 1981

TM-193 Yannakakis, Mihalis and Christos H. Papadimitriou
Algebraic Dependencies, February 1981

TM-169 Burke, Glenn and David Moon
LOOP Iteration Macro, (revised January 1981) July 1980, AD A087-372

TM-170 Ehrenfeucht, Andrzej, Rohit Parikh and Gregorz Rozenberg
Pumping Lemmas for Regular Sets, August 1980

TM-171 Meyer, Albert R.
What is a Model of the Lambda Calculus?, August 1980

TM-172 Paseman, William G.
Some New Methods of Music Synthesis, S.M. Thesis, EE & CS Dept., August 1980, AD A090-130

TM-173 Hawkinson, Lowell B.
XLMS: A Linguistic Memory System, September 1980, AD A090-033

TM-174 Arvind, Vinod Kathail and Keshav Pingali
A Dataflow Architecture with Tagged Tokens, September 1980

TM-175 Meyer, Albert R., Daniel Weise and Michael C. Loui
On Time Versus Space III, September 1980

TM-176 Seaquist, Carl R.
A Semantics of Synchronization, S.M. Thesis, EE & CS Dept., September 1980, AD A091-015

TM-177 Sinha, Mukul K.
TIMEPAD A Performance Improving Synchronization Mechanism for Distributed Systems, September 1980

TM-178 Arvind and Robert E. Thomas
I-Structures: An Efficient Data Type for Functional Languages, September 1980

TM-179 Halpern Joseph Y. and Albert R. Meyer
Axiomatic Definitions of Programming Languages, II, October 1980

TM-180 Papadimitriou, Christos H.
A Theorem in Database Concurrency Control, October 1980

TM-181 Lipski, Witold Jr. and Christos H. Papadimitriou
A Fast Algorithm for Testing for Safety and Detecting Deadlocks in Locked Transaction Systems, October 1980

PUBLICATIONS

TM-157 Meyer, Albert R. and Karl Winklmann
On the Expressive Power of Dynamic Logic, February 1980

TM-158 Stark, Eugene W.
Semaphore Primitives and Starvation-Free Mutual Exclusion, S.M. Thesis, EE & CS Dept., March 1980

TM-159 Pratt, Vaughan R.
Dynamic Algebras and the Nature of Induction, March 1980

TM-160 Kanellakis, Paris C.
On the Computational Complexity of Cardinality Constraints in Relational Databases, March 1980

TM-161 Lloyd, Errol L.
Critical Path Scheduling of Task Systems with Resource and Processor Constraints, March 1980

TM-162 Marcum, Alan M.
A Manager for Named, Permanent Objects, S.B. & S.M. Thesis, EE & CS Dept., April 1980, AD A083-491

TM-163 Meyer, Albert R. and Joseph Y. Halpern
Axiomatic Definitions of Programming Languages: A Theoretical Assessment, April 1980

TM-164 Shamir, Adi
The Cryptographic Security of Compact Knapsacks (Preliminary Report), April 1980, AD A084-456

TM-165 Finseth, Craig A.
Theory and Practice of Text Editors or A Cookbook for an Emacs, S.B. Thesis, EE & CS Dept., May 1980

TM-166 Bryant, Randal E.
Report on the Workshop on Self-Timed Systems, May 1980

TM-167 Pavelle, Richard and Michael Wester
Computer Programs for Research in Gravitation and Differential Geometry, June 1980

TM-168 Greif, Irene
Programs for Distributed Computing: The Calendar Application, July 1980, AD A087-357

PUBLICATIONS

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

PUBLICATIONS

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
Furtner 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.

PUBLICATIONS

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.

PUBLICATIONS

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.

PUBLICATIONS

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

PUBLICATIONS

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

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

PUBLICATIONS

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

PUBLICATIONS

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

PUBLICATIONS

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-262 Fischer, M.J., Lynch, N.A., Paterson, M.S.
Impossibility of Distributed Consensus with One Faulty Process, EE & CS Dept., September 1982

PUBLICATIONS

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

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

PUBLICATIONS

16. Laboratory for Computer Science Progress Report 16, July 1978-July 1979, AD A088-355

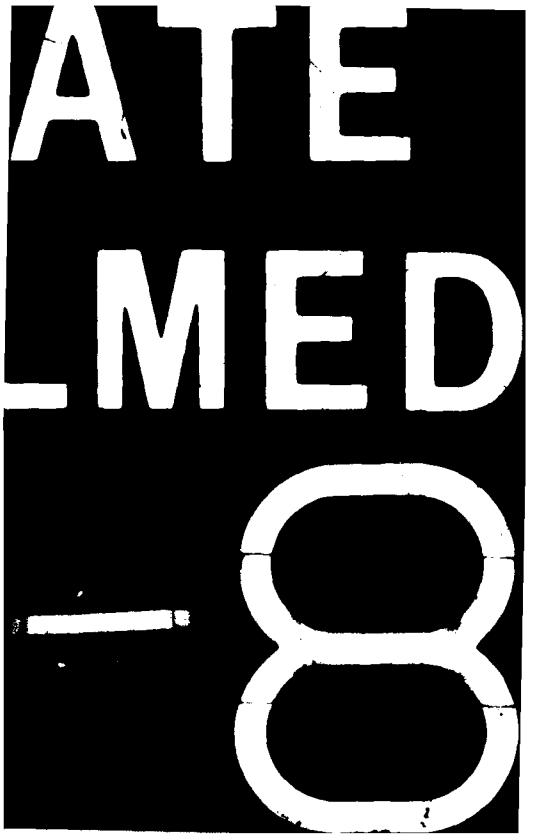
17. Laboratory for Computer Science Progress Report 17, July 1979-July 1980, AD A093-384

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

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. Prices vary. The reference number must be supplied with the request.


ATE
LMED
8