Final Report on Information Processing Research

January 1985 to September 1987

Edited by Rachel T. Levine
and the Research Documents Group

May 1988
CMU-CS-88-110

This research is sponsored by the Defense Advanced Research Projects Agency, DoD, through DARPA order 4976, and monitored by the Air Force Avionics Laboratory under contract F33615-84-K-1520. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the United States Government.

Table of Contents

1. INTRODUCTION

This report documents a broad program of basic and applied information processing research conducted by Carnegie Mellon University's Computer Science Department (CMU-CSD). The Information Processing Techniques Office of the Defense Advanced Research Projects Agency (DARPA) supported this work during the period 2 January 1985 through 31 May 1987, and extended to 30 September 1987.

The remainder of this chapter describes our research scope and the CMU-CSD research environment. Chapters 2 through 8 then present in detail our seven major research areas: Distributed Processing, Image Understanding, Machine Intelligence, Programming Technology, Distributed Sensor Networks, Graceful Interaction, and VLSI Systems Integration. Sections in each chapter present the area's general research context, the specific problems we addressed, our contributions and their significance, and an annotated bibliography.

The bibliographies present selected references that reflect the scope and significance of CMU's contributions to basic and applied computer science. Wherever possible, particularly for key reports, we have included abstracts. Finally, though basic research does not proceed with the mechanical regularity of industrial production, publication dates do indicate progress in the various problem areas. CSD Technical Report dates exhibit the closest correlation with temporal progress and the report text frequently reappears later in the more accessible archival literature.

1.1. Research scope

We organize the research reported here under seven major headings. These interrelated categories and their major objectives are:

- *Distributed Processing* (DP): Develop techniques and systems for effective use of numerous separate computers interconnected by high-bandwidth, local area networks. This effort involves developing a methodology for efficient utilization of distributed (loosely connected) personal computers. Research on a concept demonstration system will proceed in several areas:
  - Integration of subsystems and services at two levels—the user interface and the underlying system architecture—in order to provide significant improvement in the productivity of computer science researchers.
  - Design and implementation of two programming systems to support a variety of applications.
  - Development of a distributed file system offering automatic archiving facilities and transparent access to remote files.
  - Building an interactive document preparation system by merging existing packages into an integrated environment.

- Extension of current message systems to handle multi-media formats by exploiting the technology of personal computers and their interconnecting network.

- *Image Understanding* (IU): Apply knowledge effectively in assisting the image interpretation process. Research in this area has several aims:
  - Develop basic theories and construct systems for comprehending the three-dimensional structure of the environment from two-dimensional visual images.
  - Discovering the representations, algorithms, and control structures required to exploit pre-existing knowledge about the environment for image understanding.
  - Investigate special architectures and programming structures to realize vision algorithms efficiently.
- *Machine Intelligence* (MI): Explore ways to utilize knowledge in obtaining intelligent action by computers. Long range goals of this effort include the discovery of principles that enable intelligent action and the construction of computer systems that can perform tasks requiring intelligence. Research in machine intelligence covers a wide range of issues:
  - Discovering and analyzing methods of problem solving, the ways problems may be represented and how such representations affect the difficulty of solving the problems.
  - Discovering and analyzing processes that produce appropriate internal representations through recognition and description of external task situations.
  - Discovering and understanding control structures and system organizations that can combine a collection of problem-solving methods and problem representations into an effective total system.
- *Programming Technology* (PT): Develop new techniques and methods for generating better software through work on:
  - Developing advanced, highly interactive programming environments that facilitate the development and maintenance of large software projects.
  - Advanced development tools that help programmers reason about and modify programs.
  - Parallel programming methodologies.
  - Program organization and development methodologies.
- *Distributed Sensor Nets* (DSN): Construct a demonstration system of physically and logically distributed computers interacting through a communication network to identify, track, and display the situation of multiple objects in a laboratory environment. This project will involve the following tasks:
  - Evaluate and enhance the design and performance of our current testbed system.

- Extend the testbed by integrating multiprocessor hosts.
- Investigate design and implementation issues basic to distributed computing: architecture, language primitives, descriptive representation, and reliable distributed computation.

- *Graceful Interaction* (GI): Design, construct, and evaluate user interface systems that both appear cooperative and supportive to their users and can provide interfaces to a wide variety of application systems. We plan to emphasize:
  - Techniques for interactive error correction of command interaction and the provision of contextually sensitive on-line help.
  - Decoupling of application systems from direct interaction with the user through form-based communication.
  - Support for user tasks that require the coordinated use of more than one application systems.
- *VLSI Systems Integration* (VLSI): Focus on research into VLSI systems and computation, with some effort in design tools to assure that we maintain a minimal set of tools necessary to carry out our design tasks.
  - Integrate a capability for custom VLSI design into the design of novel computer architectures. Design and implement several systems that can benefit greatly from custom chips of our own design.
  - Eventually establish an environment where researchers in a wide variety of areas will have direct and convenient means of using custom VLSI for their own systems.

1.2. The Research environment

Research in the CMU Computer Science environment tends to be organized around specific experimental systems aimed at particular objectives, e.g. the demonstration of an image understanding system or the design and fabrication of an advanced VLSI systems. This report describes several such activities. Sometimes the creation and demonstration of a system is itself an appropriate scientific objective. At other times, some level of system performance constitutes the scientific goal. Thus our work tends to emphasize concept demonstration rather than system engineering. These research systems provide a convenient way to discuss and even to organize the projects at CMU-CSD. They are not always, however, ends in themselves.

A major strength of the Carnegie Mellon University environment lies in the synergy resulting from close cooperation and interdependence among varied research efforts, despite their diverse foci. For example, the Image Understanding project typically requires extensive computational resources that can profitably employ novel machine architectures and software techniques. Research in VLSI techniques, on the other hand, often provides powerful, specialized hardware in need of an application to focus continuing development efforts. Close interaction and cooperation among our various

research efforts has led to innovative approaches and solutions, and has significantly contributed to the intellectual ferment that makes Carnegie Mellon University unique in the computer science area.

We have no administrative structure that corresponds to our organization of effort. We consist simply of faculty, research scientists, and graduate students of the Computer Science Department, with the facilities support divided into an Engineering Laboratory and a Facilities Software Group. The rest of the organization is informal. This organizational style keeps the barriers between efforts to a minimum and promotes the kind of interactions and synergy reflected in the work distribution shown in Table 1-1.

	Number of Areas	DP	IUS	MI	PT	DSN	GI	VLSI
Mario Barbacci	1	x						
Hans Berliner	1			x				
Roberto Bisiani	3			x		x		x
Stephen Brookes	2				x			x
Jaime Carbonell	1			x				
Edmund Clarke	1				x			
Roger Dannenberg	1		x				x	
Scott Fahlman	2	x		x				
Merrick Furst	1				x			
Nico Habermann	1				♦			
Phil Hayes	1						♦	
Peter Hibbard	2	♦					x	
Douglas Jensen	1					x		
Takeo Kanade	1		♦					
Elaine Kant	1			x				
H.T. Kung	1							♦
John McDermott	1			♦				
Allen Newell	1			x				
Rick Rashid	1					♦		
Raj Reddy	3		x			x	x	
Bill Scherlis	1				x			
Dana Scott	1				♦			
Mary Shaw	1				x			
Herb Simon	1			x				
Alfred Spector	1					x		

x = Active research in this area
♦ = Responsible for area
Faculty participating, total = 25

Table 1-1: Distribution of faculty effort

2. DISTRIBUTED PROCESSING

The basic goal of our Distributed Processing research is to understand and evaluate, as a possible replacement for timeshared computing facilities, the use of high performance personal computers interconnected on a high-speed network. We felt from the outset that success depended on building real systems and subjecting them to use by a large number of faculty and students within the Department. To achieve this goal we built a large-scale distributed computing environment called Spice.

We developed the original Spice system on Perq computers from the PERQ Systems Corporation. At the peak of Perq use within the Department, the Spice environment consisted of over 150 workstations connected by 10 MHz and 3 MHz Ethernet LANs. Spice included the Accent network operating system, Spice Common Lisp, a research-oriented Ada+ programming environment, and numerous user interface tools. (For a discussion of our work on Accent see Chapter 6, section 3.)

Early in 1985 we began porting the work done under the Spice project onto new hardware and onto a new operating system base called Mach. Mach is a portable multiprocessor operating system patterned in many respects after Accent and built as part of the DARPA Strategic Computing Initiative. One of the major issues addressed during this porting effort was dealing with heterogeneity in both the software and hardware bases. During the transition period many of the system's components had to function under both Accent and Mach. In addition, the significant difference in data types and data packing conventions between VAX, IBM RT PC, and PERQ computers placed significant constraints on facilities, such as Matchmaker, which define and compile interfaces between major system modules.

However, by the middle of 1986, we had ported many of the basic components of Spice to run under Mach, including:

- Network message services
- Spice Common Lisp (for the RT PC)
- Sesame Authentication and Authorization servers
- Matchmaker Interprocess Interface Generator
- Flamingo Window Manager.

Overall, our research led to development efforts in operating systems, distributed file systems, languages and language support tools, and user interfaces.

2.1. System support

2.1.1. Distributed file systems

One of the challenges in the Spice project was the management of a large, distributed file system. Our answer to this challenge was to design and implement the Sesame file

system. With Sesame we demonstrated the feasibility of using Accent ports as tokens of identity. We also demonstrated the feasibility of using file caching to reduce network communication traffic and to improve performance. Sesame provides most of the interrelated services needed to allow protected sharing of data and services in a network of personal and central computers. Each service is independently implementable on other hosts on the local net. Sesame handles user verification issues both locally and between machines, name look-up services for various typed objects, migrating files to more stable media, as well as the fundamental functions of reading and writing files. In 1985 Sesame was running as an alternate file system in the Spice environment [Thompson et al. 85].

A file system similar in many respects to Sesame is the Vice/Virtue distributed file system developed by the CMU Information Technology Center (ITC). The similarities between Vice/Virtue and Sesame arose out of the fact that many members of the Sesame design team participated in the development of Vice/Virtue as a way to transfer that technology to the rest of the University and to industry. During the later part of 1986, we incorporated Vice/Virtue back into Mach and worked with members of the ITC to compare the performance of Vice/Virtue to remote file access facilities such as Sun's network file system (NFS). Using Vice/Virtue, Mach can now provide a large shared file system for hundreds of workstations at a time with as many as 50-75 workstations per server.

In addition to Vice/Virtue support, we began distribution of a compatible remote file access facility, CMU RFS, that allows workstations to cooperate and share files without merging them into a common administrative domain. We tested CMU RFS over 10 MHz Ethernet, 4 MHz IBM token ring, and even 56 KB and 9.6 KB serial line connections to the CMU SEI and to a research laboratory in Princeton, NJ. During the first half of 1987, the Vice/Virtue vs. NFS benchmarks were performed and a paper on the results was accepted by the ACM Symposium on Operating Systems Principles and recommended for publication in ACM Transactions on Computing. The key result was the demonstration of Vice/Virtue's ability to support significantly larger workstation loads than NFS. NFS server performance fell off drastically with increasing numbers of clients while Vice/Virtue allowed a significant number of workstation clients per server with only slowly growing response time.

2.1.2. Access to remote systems

Early in 1985 we completed a much-improved Common Lisp version of our interface specification language Matchmaker and its compiler. Most notably, Matchmaker can now generate type definition files for all the target languages; this previously had to be done by hand for each language. As a result, the entire interface definition is contained in the Matchmaker input file rather than in a variety of files that had to be kept consistent by the programmer [Jones et al. 85].

As we made the change from Accent to Mach, we had to refine Matchmaker because the old Spice style interfaces had become cumbersome, and in some cases incorrect.

Almost all the syntax and some of the semantics of the interfaces were changed to conform to Mach programming conventions. The Matchmaker code provides communication, run-time support for type-checking, type conversions, synchronization and exception handling. Matchmaker in combination with Mach provides a heterogeneous distributed object-oriented programming environment. One refinement to Matchmaker is MIG [Jones and Rashid 87]. MIG allows a programmer to create a distributed program without worrying about the details of sending messages or type conversion between different machines. MIG is an implementation of a subset of Matchmaker that generates C and C++ remote procedure call interfaces for interprocess communication between Mach tasks.

2.1.3. A distributed object system

We have developed a distributed object system for Mach. This object system, named FOG, provides machine- and language-independent descriptions of structured data, language-independent message passing of structured data, and a distributed object reference/method invocation system. The design of FOG was largely influenced by the Matchmaker remote procedure call system and by the Flamingo User Interface Management System. Unlike those systems, however, FOG provides the ability to dynamically add the definitions of new structured types to the run-time environment, and a transparent method invocation system.

2.1.4. Communication security

A major area of concern in managing large numbers of workstations in a local area network is communication security. During the last part of 1986, we designed a new network communication manager for Mach which included a collection of encryption algorithms and protocols to securely extend the capability protection of a single node into the network environment [Sansom et al. 86]. To allow one communicating process to verify the identity of another, we designed a new network authentication and authorization protocol. The underlying Mach interprocess communication facility is protected on a single node by using capabilities managed by the operating system kernel to represent communication channels. During the first half of 1987 this work was brought to conclusion in the implementation of that manager and its initial testing within Mach.

2.1.5. Large scale software distribution

We developed and implemented facilities for maintaining software consistency in a distributed environment. The key component of this work is a program called "SUP" which performs automatic, network software distribution and update. In addition to distribution and the ability to carry out procedures for software installation, SUP provides a level of security by allowing software collections to be protected by individual encryption keys.

During the latter part of 1986, we tested these facilities both within our Department

and jointly with other research laboratories, including the CMU Software Engineering Institute, the University of Maryland, and Berkeley. This was done using the ARPAnet as well as private leased line networks.

The use of SUP grew substantially during the first half of 1987. For example, during this period SUP began to be used for routine software distribution of Mach and related source code between CMU and the DARPA-funded UltraMax project at Encore, and between CMU and the DARPA-funded BBN Monarch project. Our experiences with SUP led to the development of an automated software distribution project proposal.

2.2. Programming environments

One of our aims while developing the Spice environment was to make it language independent and to provide it with interlanguage communication facilities. We achieved this by developing comprehensive programming environments for Ada and Lisp, each with its own microcode interpreter. In the beginning of 1985 our work on the Spice Ada environment reached maturity. We distributed the compiler and utilities to the CMU community, and individual users continue to enhance it according to their needs.

While our work on Ada ended, our work on Lisp continued to grow. In 1984 we had designed Common Lisp and implemented Spice Lisp. In 1985 our efforts were directed toward debugging, modifying, and improving the Lisp programming environment, building up a library of application programs, and porting the Spice Lisp system from Perqs to IBM RT PCs. We extended the Hemlock text editor, reduced Spice Lisp's core size, and organized an initial library of portable and semi-portable public-domain programs for Common Lisp, including OPS5.

We extended the Hemlock text editor (written in mostly-portable Common Lisp) with facilities such as a real-time spelling corrector and a "shell" mode that allows the user to control many jobs while residing in a convenient, easily extensible, Lisp-based environment. Our extensions lay the groundwork for work on user interfaces in the new workstation. (See section 2.3 and Chapter 7.)

In order to port Spice Common Lisp we first reduced its core size by 30%, with noticeable reductions in the working sets of programs. We achieved this by eliminating redundant structures and improving paging performance by automatically allocating related procedures in adjacent virtual memory addresses. These techniques allowed us to pack a full Common Lisp system onto a 4 Mbyte IBM RT PC. Before the compression, we thrashed on that configuration; after, the Lisp didn't page too much except when running large AI programs. By the end of 1985 we had finished porting Common Lisp and the Hemlock text editor to the new IBM workstation.

In 1986 we began studying RISC-like architectures with features designed especially for Lisp and object-oriented programming. We began designing a new, more portable, public-domain Common Lisp compiler. The compiler's design incorporated significant optimizations and improved compatibility for today's conventional processors, par-

ticularly RISC machines. We continue to work on this highly-optimizing compiler and anticipate it will be complete by the end of 1987. Its design will make it easy to convert it to produce code for new instructions sets — a valuable attribute at a time when many new RISC architectures are appearing.

While working on the Spice Lisp environment, it became clear to us that the Common Lisp community needed support and that an informal group was not going to suffice. At the December 1985 Common Lisp meeting the attendees decided to seek formal ANSI and ISO standardization for the language. The new committee, X3J13, began work by the year's end and is the U.S. participant in the ISO's international standards effort. Much time and effort went into the activities that support the Common Lisp community: answering questions about the language's design and possible extensions, discussing various formal organizations that might be adopted, and supporting manufacturers in their efforts to bring Common Lisp systems to market. At the IJCAI conference in August, Xerox announced plans to introduce a Common Lisp for their machines during 1986. Among US manufacturers affiliated with the DARPA research community, Xerox was the last major corporation to accept Common Lisp.

In the beginning of 1987 we incorporated the Xerox Common Loops object-oriented extension into our Common Lisp and completed and released our Common Lisp implementation for the IBM RT PC in source form. This enables us to gain real experience with this facility before it [the standard] becomes final and unchangeable. We also integrated the CMU Common Lisp system with the popular X window system, and participated in the design of CLX, a proposed standard interface between Common Lisp and X, version 11. The Hemlock text editor was made to work under X windows and also was made to work with standard ASCII terminals. We have been cooperating with members of the Dante project to develop higher-level graphics facilities. As part of the effort to interface to X, we developed a facility for interfacing Lisp programs to C subroutines.

In 1987 CMU remains a center of Common Lisp development and innovation in part due to our role in Common Lisp standards efforts at both the national and international level (ANSI and ISO), and in part because CMU Common Lisp is the most complete public domain version of the language and is a base for many other Common Lisp systems.

2.3. User interfaces

Along with researchers from the Dante project, we developed a user interface shell for Mach in Common Lisp which allows a Mach Common Lisp programmer to easily invoke and control the actions of his Mach programs and at the same time do so within a full Common Lisp programming environment. This tool, the Lisp Shell, provides considerably more flexibility and programmability than traditional UNIX shells because of its more complete programming environment. It also makes possible the extension of AI techniques to traditional user interface tasks of UNIX-like systems. (See Chapter 7, section 1.1.)

The work on the Lisp Shell is closely linked to the Warp Programming Environment. The main interface mechanism for the Warp Programming Environment is a Lisp-based command interpreter. The Warp Programming Environment is a distributed software environment that allows users of the Warp multiprocessor to develop and debug Warp applications. The environment includes local user workstations, a set of central server workstations each acting as a host for a Warp machine, and a few Warp multiprocessors. The command interpreter is executed in Lisp on the user's workstation, while the more communication-intensive tasks are executed on the host workstations. The system allows the user to choose the ideal load distribution for the different machines (local, host, and Warp) and can provide extensive help and information about the current status of the environment. A first version of the Warp Programming Environment has been released to the Warp users community outside CMU.

In the area of window managers, we made the transition to X as a window manager base to conform with emerging window system standards. Our work on object-oriented graphics and window management was transferred to this new environment. Before this transition we used Flamingo.

Flamingo

In February 1985, Spice researchers met to consider the current and future requirements of the user interface systems available on Spice systems. Our primary goal was to create a flexible, distributed interface system that could manage input and output resources for application programs. The challenge was to develop a system that could span heterogeneous hardware architectures (so users could, for example, process on a Perq and display on a MicroVAX) while providing upward compatibility for our older software.

This group designed Flamingo, an object-oriented interface manager for programs running within the Spice environment. Our design for the Flamingo system addressed these key objectives through an object-oriented strategy that can associate data objects with operations, or "methods", implemented in remote processes. The "remote method" mechanism differs from the traditional user/server structure found in many distributed systems. In Flamingo, the system is a central "object manager," while remote client programs provide the methods that Flamingo and other clients may call. Flamingo and its clients both serve and use each other.

On June 5, 1985, the first Flamingo prototype was executing on a MicroVAX I. This prototype had the simple ability to create rectangular regions on the screen using a specially constructed raster operation, with pixel array objects representing the screen objects being managed by Flamingo. In October 1985, we released an initial version to MicroVAX users running the Mach operating system [Smith and Anderson 86a], and in 1986 Flamingo was ported to the IBM RT PC and Sun. Flamingo usage in the Department is limited since the switch to X was made.

2.4. Bibliography

[Back and Kurki-Suonio 85]
Back, R.J. and R. Kurki-Suonio.
Serializability in distributed systems with handshaking.
Technical Report CMU-CS-85-109, Carnegie Mellon University, Computer Science Department,
February, 1985.

Two interleaving models, a concurrent model and a serial model, are given for distributed systems in which two or more processes can be synchronized for communication by a handshake mechanism. The equivalence of the two models is shown, up to fairness and justice properties. The relationships between the natural fairness and justice notions in the models are analyzed, and sufficient conditions are derived for the validity of serial reasoning in the concurrent model. Proving that these conditions hold for a particular system can be carried out totally within the simpler serial model.

[Barbacci and Wing 87a]
Barbacci, M.R. and J.M. Wing.
Durra: a task-level description language.
In *Proceedings of the International Conference on Parallel Processing*, 1987.
Also available as Technical Report CMU-CS-86-176.

Computation-intensive, real-time applications such as vision, robotics, and vehicular control require efficient concurrent execution of multiple *tasks*, e.g., sensor data collection, obstacle recognition, and global path planning, devoted to specific pieces of the application. At CMU we are developing some of these applications and the hardware and software environments to support them, and in this paper we present a new language, Durra, to write what we call *task-level application descriptions*. Although the language was developed with a concrete set of needs, we aim at a broader class of applications and hardware implementations. After a brief description of the nature of these applications and a scenario for the development process, we concentrate on the language and its main features.

[Barbacci and Wing 87b]
Barbacci, M.R. and J.M. Wing.
Lecture Notes in Computer Science. Volume Volume II, Parallel Languages: Specifying functional and timing behavior for real-time applications,
Proceedings of PARLE (Parallel Architectures and Languages Europe) 259. Springer-Verlag Publishers, 1987.
Also available as Technical Report CMU-CS-86-177.

We present a notation and a methodology for specifying the functional and timing behavior of real-time applications for a heterogeneous machine. In our methodology we build upon well-defined, though isolated, pieces of previous work: Larch and Real Time Logic. In our notation, we strive to keep separate the functional specification from the timing specification so that a task's functionality can be understood independently of its timing behavior. We show that while there is a clean separation of concerns between these two specifications, the semantics of both pieces as well as their combination are simple.

[Barbacci et al. 85a]
Barbacci, M.R., S. Grout, G. Lindstrom, M.P. Maloney, E.T. Organick, and D. Rudisill.
Ada as a hardware description language: an initial report.
In *Proceedings of the 7th International Conference on Computer Hardware Description Languages*, CHDL, August, 1985.

This paper reports on our initial results in using Ada as a Hardware Description Language. Ada provides abstraction mechanisms to support the development of large software systems. Separate compilation as well as nesting of packages, tasks, and subprograms allow the construction of modular systems communicating through well defined interfaces. The complexity of modern chips (e.g. those proposed in the VHSIC program) will require the use of those features that make Ada a good language for programming-in-the-large.

The key to our approach is establishing a writing style appropriate to the objective of describing both the behavior and the structure of hardware components. We model a hardware system as an ensemble of typed objects, where each object is an instance of an abstract data type. The type definition and the associated operations are encapsulated by a corresponding package. In this paper we illustrate out approach through a series of examples, building up a hypothetical hierarchy of hardware components. We conclude by discussing ways to describe arbitrarily complex simulation models and synthesis styles.

[Barbacci et al. 85b]
Barbacci, M.R., W.H. Maddox, T.D. Newton, and R.G. Stockton.
The Ada+ front end and code generator.
In *Proceedings of the Ada International Conference*, ACM, September, 1985.

It will be shown that the general design of the Ada+ system has been kept as simple and straightforward as possible. This has proven immensely useful in that it provides a simple framework for dealing with many of the details involved in

Ada semantics. The semantic areas mentioned would have been much harder to implement if the compiler had attempted to use a more efficient or restrictive scheme. In addition, the separation of facilities tended to minimize interaction between various features and make it easier to produce correct code.

We will describe a simplified runtime representation for Ada programs, as implemented by the Ada+ compiler. We believe that our experience may be of assistance to others undertaking the development of an Ada compiler with limited resources.

[Black 86] Black, D.L.
Measure theory and fair arbiters.
Technical Report CMU-CS-86-116, Carnegie Mellon University, Computer Science Department,
April, 1986.

The existence of fair arbiters and formal specifications for them was a major topic of discussion at the Workshop. One of the many results discussed is that it is possible to create a fair arbiter by adding output delays to a mutual exclusion element. This work builds on that result by investigating the basic fairness properties of mutual exclusion elements and combinations thereof. Rather than working with a particular mutual exclusion element, we abstract the behavior of a class of such elements using a choice set model and a probabilistic specification of the choice inherent in mutual exclusion. This allows us to capture the choice behavior of a mutual exclusion element in a probabilistic structure containing finite and infinite traces. To analyze such structures we employ techniques from the mathematical discipline of measure theory, and in particular the measure theoretical treatment of probability. The major result from this analysis is that the mutual exclusion elements are fair under a strong probabilistic notion of fairness.

[Cohen et al. 85] Cohen, E.S., E.T. Smith, and L.A. Iverson.
Constraint-based tiled windows.
Technical Report CMU-CS-85-165, Carnegie Mellon University, Computer Science Department,
October, 1985.

Typical computer workstations feature large graphical display screens filled with windows that each show information about a user's processes and data. Window managers implemented on these systems provide mechanisms for creating, destroying, and arranging windows on the screen. Window managers generally follow either a 'desktop' metaphor, allowing windows to overlap each other like sheets of paper piled up on a physical desk, or they use a

'tiling' model, arranging each window with a specific size and location on the screen such that no overlap occurs between windows.

Desktop models allow for the most freedom in arranging window, but can become quite frustrating to use when faced with a large number of windows 'coming and going' over a short period of time that must be visible on the screen simultaneously. Tiling models save the user from having to specify every window location and guarantee that each window will be completely visible on the screen, but thus far, such systems have provided relatively poor mechanisms for the user to control layout decisions.

This paper describes work in progress on tiled window management featuring a constraint-based layout mechanism. With this mechanism the user can specify the appearance of individual windows and constrain relationships between windows, thus providing necessary control over the tiling process. We discuss our constraint model as well as detail an implementation approach that would make use of those constraints to arrange windows on a screen.

[Fahlman 87] Fahlman, S.E.
Common Lisp.
Annual Review of Computer Science 2:1-18, 1987.

The emergence of Common Lisp as a standard, well-supported dialect of Lisp that is available on many different computers has made it much easier to move artificial intelligence concepts and software tools out of the laboratory and into the commercial marketplace. This development has also stimulated interest in Lisp as a language for applications outside of AI. This paper examines some of the ways in which Common Lisp differs from other programming languages and discusses some of the implications of these differences for the Lisp programmer, the Lisp implementor, and the designer of hardware intended to run Lisp.

[Fitzgerald and Rashid 86]
Fitzgerald, R. and R. Rashid.
The integration of virtual memory management and interprocess communication in Accent.
In *ACM Transactions on Computing Systems*, ACM, May, 1986.

The integration of virtual memory management and interprocess communication in the Accent network operating system kernel is examined. The design and implementation of the Accent memory management system is discussed and its performance, both on a series of message-oriented benchmarks and in normal operation, is analyzed in detail.

[Herlihy 85] Herlihy, M.
Comparing how atomicity mechanisms support replication.
In *Proceedings of the 4th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, ACM SIGACT-SIGOPS, August, 1985.

Most pessimistic mechanisms for implementing atomicity in distributed systems fall into three broad categories: two-phase locking schemes, timestamping schemes, and hybrid schemes employing both locking and timestamps. This paper proposes a new criterion for evaluating these mechanisms: the constraints they impose on the availability of replicated data.

A replicated data item is a typed object that provides a set of operations to its clients. A quorum for an operation is any set of sites whose co-operation suffices to execute that operation, and a quorum assignment associates a set of quorums with each operation. Constraints on quorum assignment determine the range of availability properties realizable by a replication method.

This paper compares the constraints on quorum assignment necessary to maximize concurrency under generalized locking, timestamping, and hybrid concurrency control mechanisms. This comparison shows that hybrid schemes impose weaker constraints on availability than timestamping schemes, and locking schemes impose constraints incomparable to those of the others. Because hybrid schemes permit more concurrency than locking schemes, these results suggest that hybrid schemes are preferable to the others for ensuring atomicity in highly available and highly concurrent distributed systems.

[Herlihy 86] Herlihy, M.
A quorum-consensus replication method for abstract data types.
ACM Transactions on Computer Systems 4(1):32-53, February, 1986.
Also available as Technical Report CMU-CS-85-164.

Replication can enhance the availability of data in distributed systems. This paper introduces a new method for managing replicated data. Unlike many methods that support replication only for uninterpreted files, this method systematically exploits type-specific properties of objects such as sets, queues, or directories to provide more effective replication. Each operation requires the cooperation of a certain number of sites for its successful completion. A quorum for an operation is any such set of sites. Necessary and sufficient constraints on quorum intersections are derived from an analysis of the data type's algebraic structure. A reconfiguration method is proposed that permits quorums to be changed dynamically. By taking advantage

of type-specific properties in a general and systematic way, this method can realize a wider range of availability properties in a more flexible reconfiguration than comparable replication methods.

[Jones and Rashid 87]
Jones, M.B. and R.F. Rashid.
Mach and Matchmaker: Kernel and language support for object-oriented distributed systems.
Technical Report CMU-CS-87-150, Carnegie Mellon University, Computer Science Department,
September, 1987.
Also available in the *Proceedings of the First Annual ACM Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA*, September 1986.
Mach, a multiprocessor operating system kernel providing capability-based interprocess communication, and Matchmaker, a language for specifying and automating the generation of multi-lingual interprocess communication interfaces, are presented. Their usage together providing a heterogeneous, distributed, object-oriented programming environment is described. Performance and usage statistics are presented. Comparisons are made between the Mach/Matchmaker environment and other related systems. Possible future directions are examined.

[Jones et al. 85] Jones, M.B., R.F. Rashid, and M.R. Thompson.
Matchmaker: an interface specification language for distributed processing.
In *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1985.
Matchmaker, a language used to specify and automate the generation of interprocess communication interfaces, is presented. The process of and reasons for the evolution of Matchmaker are described. Performance and usage statistics are presented. Comparisons are made between Matchmaker and other related systems. Possible future directions are examined.

[Kurki-Suonio 85] Kurki-Suonio, R.
Towards programming with knowledge expressions.
Technical Report CMU-CS-85-153, Carnegie Mellon University, Computer Science Department,
August, 1985.
Explicit use of knowledge expressions in the design of distributed algorithms is explored. A non-trivial case study is carried through, illustrating the facilities that a design language could have for setting and deleting the knowledge that the processes possess about the global state and about the knowledge of other processes. No implicit

capabilities for logical reasoning are assumed. A language basis is used that allows not only eventual but also true common knowledge between processes. The observation is made that the distinction between these two kinds of common knowledge can be associated with the level of abstraction: true common knowledge of higher levels can be implemented as eventual common knowledge on lower levels. A knowledge-motivated abstraction tool is therefore suggested to be useful in supporting stepwise refinement of distributed algorithms.

[Liskov et al. 85] Liskov, B., M. Herlihy, and L. Gilbert.
Limitations of synchronous communication with static process structure in languages for distributed computing.
Technical Report CMU-CS-85-168, Carnegie Mellon University, Computer Science Department,
October, 1985.

Modules in a distributed program are active, communicating entities. A language for distributed programs must choose a set of communication primitives and a structure for processes. This paper examines one possible choice: synchronous communication primitives (such as rendezvous or remote procedure call) in combination with modules that encompass a fixed number of processes (such as Ada tasks or UNIX processes). An analysis of the concurrency requirements of distributed programs suggests that this combination imposes complex and indirect solutions to common problems and thus is poorly suited for applications such as distributed programs in which concurrency is important. To provide adequate expressive power, a language for distributed programs should abandon either synchronous communication primitives of the static process structure.

[McDonald et al. 87]
McDonald, D.B., S.E. Fahlman, and A.Z. Spector.
An efficient Common Lisp for the IBM RT PC.
Technical Report CMU-CS-87-134, Carnegie Mellon University, Computer Science Department,
July, 1987.

CMU Common Lisp is a full implementation of Common Lisp developed within the Computer Science Department of Carnegie Mellon University. It runs on the IBM RT PC under CMU's Mach operating system, which is compatible with Berkeley Unix 4.3. An important consideration in the design of CMU Common Lisp was our desire to make the best possible use of the IBM RT PC's RISC instruction set and flexible memory architecture.

CMU Common Lisp provides a comprehensive Lisp program-

ming environment and is now heavily used within the CMU Computer Science Department, both in Lisp's traditional role as the language of AI research, and in many other areas where 'mainstream' languages were formerly used.
In this paper we focus on the design and implementation strategy used in CMU Common Lisp. We also briefly describe the improvements we plan for the future.

[Rashid 86a] Rashid, R.F.
From RIG to Accent to Mach: the evolution of a network operating system.
In *Proceedings of the ACM/IEEE Computer Society 1986 Fall Joint Computer Conference*, November, 1986.
This paper describes experiences gained during the design, implementation and use of the CMU Accent Network Operating System, its predecessor, the University of Rochester RIG system and its successor CMU's Mach multiprocessor operating system. It outlines the major design decisions on which the Accent kernel was based, how those decisions evolved from the RIG experiences and how they had to be modified to properly handle general purpose multiprocessors in Mach. Also discussed are some of the major issues in the implementation of message-based systems, the usage patterns observed with Accent over a three year period of extensive use at CMU and a timing analysis of various Accent functions.

[Rashid 86b] Rashid, R.F.
Threads of a new system.
Unix Review 4(8):37-49, 1986.
The Department of Defense, anxious for better multithreaded application support, has funded the development of Mach, a multiprocessor operating system for UNIX applications.

[Sansom et al. 86] Sansom, R.D., D.P. Julin, and R.F. Rashid.
Extending a capability based system into a network environment.
Technical Report CMU-CS-86-115, Carnegie Mellon University, Computer Science Department,
April, 1986.
The Mach operating system supports secure local communication within one node of a distributed system by providing protected communication capabilities called Ports. The local port-based communication abstraction can be extended over a network by Network Server tasks. The network servers efficiently act as local representatives for remote tasks by implementing an abstraction of Network Ports. To extend the security of the port-based communication abstraction into the network environment, the network servers must protect both the messages sent over the network to network ports and the access rights to network ports. This

paper describes in detail the protocols used by the network servers to support protection.

[Scherlis and Jorring 86]
Scherlis, W.L. and U. Jorring.
Deriving and using destructive data types.
In *IFIP TC2 Working Conference on Programs Specification and Transformation*, IFIP, March, 1986.
Techniques are presented that support the derivation and reuse of high performance implementations of programs specified using applicative abstract data types. There are two facets to this work.
First, we show how the performance of programs specified using general-purpose types such as lists or trees can be improved by specializing the types to their contexts of use. These techniques provide a means for using mechanical tools to build, given a small set of general types, a diverse collection of derived specialized types suitable for a broad variety of applications.
Second, we sketch techniques for deriving, from simple applicative programs, efficient implementations that use destructive data operations and that can reuse heap-allocated storage. These techniques rely on simple propagation of non-interference assertions; reasoning about the global state of storage is not required for any of the examples presented.

[Sha 85]
Sha, L.
Modular concurrency control and failure recovery--- consistency, correctness and optimality.
Technical Report CMU-CS-85-114, Carnegie Mellon University, Computer Science Department,
March, 1985.
A distributed computer system offers the potential for a degree of concurrency, modularity and reliability higher than that which can be achieved in a centralized system. To realize this potential, we must develop provably consistent and correct scheduling rules to control the concurrent execution of transactions. Furthermore, we must develop failure recovery rules that ensure the consistency and correctness of concurrency control in the face of system failures. Finally, these scheduling and recovery rules should support the modular development of system and application software so that a transaction can be written, modified, scheduled and recovered from system failures independently of others.
To realize these objects, we have developed a formal theory of modular scheduling rules and modular failure recovery rules. This theory is a generalization of the classical works

of serializability theory, nested transactions and failure atomicity. In addition, this theory addresses the concepts of consistency, correctness, modularity and optimality in concurrency control and failure recovery. This theory also provides us with provably consistent, correct and optimal modular concurrency control and failure recovery rules.

[Smith and Anderson 86a]
Smith,E.T. and D.B. Anderson.
Flamingo: object-oriented window management for distributed, heterogeneous systems.
Technical Report CMU-CS-86-118, Carnegie Mellon University, Computer Science Department,
April, 1986.
This report describes the Flamingo User Interface System (Version 15). Flamingo is a system for managing the interface between users and programs that run in large, distributed, heterogeneous computing environments. Using the mechanisms described herein, Flamingo provides a set of user interface features associated with traditional window management.
Flamingo uses an object-oriented structure whose objects can have methods (or 'operations') implemented in remote processes. This mechanism differs from the traditional 'user/server' relationship that is used to structure many distributed systems. In Flamingo, the system is a central 'object manager', while client programs running as remote processes provide the implementations for methods called upon by Flamingo and other clients. Both the clients and Flamingo act as servers and users of each other.
Flamingo is built on the Mach operating system, which provides a UNIX environment plus a message-based Interprocess Communication (IPC) mechanism. Flamingo uses a machine-generated Remote Method Invocation (RMI) mechanism to provide a symmetric interface between it and client programs that wish to call on method implementations located in each. The Remote method Invocation system itself uses a machine-generated Remote Procedure Call mechanism as a message transport layer.

[Smith and Anderson 86b]
Smith, E.T. and D.B. Anderson.
Flamingo: object-oriented abstractions for user interface management.
In *Proceedings of the Winter 1986 USENIX Conference*, USENIX, January, 1986.
This paper describes the Flamingo User Interface System designed for use by programs running on Spice machines. Flamingo is designed to use the remote procedure call

mechanism available through the various operating systems running on Spice machines to provide a flexible, robust, machine-independent interface to a variety of different machines communicating over local area networks.

Flamingo separates the abstractions of the objects used by the program to communicate with the user from the actual devices used to read or write information. A window manager is provided that makes a suitable mapping from output provided to map input events from real devices to either window management routines or to a form suitable for input by a program.

Flamingo itself can be divided into different processes running on different machines each implementing different parts of the system. All exported objects used for communicating between users and programs are implemented with specific methods defining the operations available for an instance of a particular object or for all objects of a class in a given running Flamingo system. These mechanisms provide a flexible framework within which a variety of window managers and user interfaces can be realized and evaluated.

[Thompson et al. 85]
Thompson, M.R., R.D. Sansom, M.B. Jones, and R.F. Rashid.
Sesame: the Spice file system.
Technical Report CMU-CS-85-172, Carnegie Mellon University,
Computer Science Department,
November, 1985.

Sesame provides several distinct but interrelated services needed to allow protected sharing of data and services in an environment of personal and central computers connected by a network. It provides a smooth memory hierarchy between the local secondary storage and central file system. It provides a global name space and a global user authentication protocol.

[Wing 87]
Wing,J.M.
Writing Larch interface language specifications.
ACM Transactions on Programming Languages and Systems
9(1):1-24, 1987.

Current research in specifications is emphasizing the practical use of formal specifications in program design. One way to encourage their use in practice is to provide specification languages that are accessible to both designers and programmers. With this goal in mind, the Larch family of formal specification languages has evolved to support a two-tiered approach to writing specifications. This approach separates the specification of state transformations and programming language dependencies from the

specification of underlying abstractions. Thus, each member of the Larch family has a subset derived from a programming language and another subset independent of any programming languages. We call the former interface languages, and the latter the Larch Shared Language.

This paper focuses on Larch interface language specifications. Through examples, we illustrate some salient features of Larch/CLU, a Larch interface language for the programming language CLU. We give an example of writing an interface specification following the two-tiered approach and discuss in detail issues involved in writing interface specifications and their interaction with their Shared Language component.

[Wing and Nixon 85]
Wing, J.M. and M.R. Nixon.
Adding temporal logic to Ina Jo.
Technical Report CMU-CS-85-146, Carnegie Mellon University, Computer Science Department,
July, 1985.

Toward the overall goal of putting formal specifications to practical use in the design of large systems, we explore the combination of two specification methods: using temporal logic to specify concurrency properties and using an existing specification language, Ina Jo, to specify functional behavior of nondeterministic systems. In this paper, we give both informal and formal descriptions of both current Ina Jo and Ina Jo enhanced with temporal logic. We include details of a simple example to demonstrate the expressiveness of the enhanced language. We discuss at length our language design goals, decisions, and their implications. The appendices contain complete proofs of derived rules and theorem schemata for the enhanced formal system.

[Wing et al. 85] Wing, J.M., J. Guttag, and J. Horning.
The Larch family of specification languages.
In *IEEE Software*, IEEE, September, 1985.

The use of suitable formalisms in the specification of computer programs offers significant advantages. Although there is considerable theoretical interest in this area, practical experience is rather limited. The Larch Project, a research project intended to have practical applications within the next few years, is developing tools and techniques to aid in the productive application of formal specifications. A major part of the project is a family of specification languages. Each specification language has components written in two languages. The Larch interface languages are particular to specific programming languages, while the Larch Shared Language is common to all languages.

3. IMAGE UNDERSTANDING

Image understanding (IU) research aims at developing adequate and versatile techniques to facilitate the construction of IU systems. Such techniques include processing methods for extracting useful information from images; representation and control structures for exploiting relevant knowledge sources; and special architectures and programming structures to realize the algorithms efficiently. The CMU Image Understanding program covers a variety of topics in vision ranging from the theory of color and texture to the system issues in building demonstrable vision systems. Our research focused on:

- *System Framework for Knowledge-Based Vision:* Developing systems that can combine generic reasoning with task-specific, constraints to solve complex vision problems.
- *Algorithms for Three-Dimensional Object Recognition:* Investigating special architectures and programming structures to realize vision algorithms efficiently. Extending current techniques in image analysis, representation, and geometrical reasoning.
- *Inferring Shape and Surface Properties:* Developing new methods for inferring basic surface and shape information from images using color, texture, and motion.

3.1. System framework for knowledge-based vision

To broaden the applicability of knowledge-based vision systems, we must have a general framework in which we can represent the 3-D object models, control the analysis using both generic and domain-specific knowledge, and access image features flexibly and efficiently. Toward this goal, we have developed image processing systems that combine general vision techniques with task-specific application-oriented knowledge. During the contract period, we have:

- Created the Digital Mapping Laboratory, a facility that uses the MAPS (Map Assisted Photointerpretation System) system to explore the generation and maintenance of a large-scale domain knowledge base.
- Improved the 3D Mosaic system to integrate techniques that allow us to build and verify 3-D scene models and demonstrate acquisition of scene models.
- Developed and tested SPAM (System for the Photointerpretation of Airports using MAPS), a rule-based system that uses map and domain-specific knowledge to interpret airport scenes.
- Worked on 3DFORM, a 3-D vision system based on a frame language (Framekit) defined on Common Lisp.

3.1.1. Digital mapping laboratory

A key issue in building systems for cartography and aerial photointerpretation is the generation and maintenance of a domain knowledge base. Loosely speaking, this "knowledge base" should contain known facts and spatial relations between objects in an area of interest, access to historical or normalcy reports, and methods that relate earth coordinates to pixel locations in digital imagery. Unfortunately, these spatial database capabilities are somewhat different from those found in traditional geographic information systems. Another issue is including methods for utilizing and representing spatial knowledge. Simply having access to cartographic descriptions does not address the problem of how to operationalize iconic descriptions for image analysis and interpretation.

In 1985, we created the Digital Mapping Laboratory (DML) for aerial photointerpretation, cartography, and computer vision [McKeown et al. 85a]. The DML uses MAPS(Map Assisted Photointerpretation System) to explore the generation and maintenance of a large-scale domain knowledge base. MAPS is an image/map database system that contains approximately 100 high-resolution aerial images, a digital terrain database, and a variety of map databases from the Defense Mapping Agency. We have continued work on the MAPS image/map database system primarily in the area of integrating map data to support our work in rule-based analyses of airport scenes.

3.1.2. Rule-based system for image processing

Interpreting aerial photographs requires extensive knowledge about the scene under consideration. Knowledge about the type of scene aids in low-level and intermediate-level image analysis, and drives high-level interpretation by constraining search for plausible consistent scene models.

We investigated using a rule-based system to control image processing and interpret the results with respect to a world model, as well as representing the world model within an image/map database. SPAM (System for the Photointerpretation of Airports using MAPS) uses domain-specific knowledge to interpret airport scenes [McKeown et al. 85b]. We have developed a set of specialized tools to aid this task. These tools include:

- A user interface for interactive knowledge acquisition
- Automated compilation of that knowledge from a schema-based representation into productions that are executable by our interpretation system
- A performance analysis tool that generates a critique of the final interpretation.

We demonstrated the generality of these tools by generating rules for a new task—suburban home scenes—and analyzing a set of images with our interpretation system.

(For more information, see section 4.1.2 in the Machine Intelligence Chapter.)

3.1.3. 3D Mosaic system

Most vision systems are based on many implicitly assumed relationships that restrict their usefulness to a single domain. By making all of these assumptions explicit, they can be used for reasoning about vision in a wide variety of domains. The 3D Mosaic system is a photointerpretation system built on the principle of explicit reasoning at all levels.

The 3D Mosaic system is based on both low-level and high-level image-processing tools, along with procedures that determine each tool's applicability. We use these tools for bottom-up verification of hypotheses developed by the top-down component of the system. For example, when a building roof is found in an image, the system hypothesizes edges from each roof vertex to the ground. We are studying operators that will verify these hypothesized edges to determine which ones to use under what conditions. We are testing the system on several aerial images of Washington, D.C. We have worked on a scheme for representing and reasoning about geometric objects, such as projections between 2-D images and 3-D scenes, shape and surface properties of objects, and geometrical and topological relationships between objects [Walker and Herman 87]. These capabilities are essential for knowledge-based, 3-D photointerpretation systems.

We adopted a frame-based representation using the CMU-built Framekit tool in Common Lisp. Each object type, such as a point, line, or plane, is represented as a frame. Specific objects are created by instantiating the generic frame. Instantiating an object consists of creating a new frame with a unique name and filling in slots specific to the new object. Empty slots are inherited from the generic object by means of an IS-A hierarchy.

Frames also represent geometric relationships between objects, such as parallel or perpendicular for lines. Each geometric relationship has slots for two or more geometric objects plus one or more numeric ranges. For example, the LINES-IN-PLANE relationship adds each line to the plane's list of contained lines and adds the plane's normal to the list of vectors perpendicular to each line. Finally, computations are done to ensure that the true numeric values (such as the angle between two lines) fall within the specified ranges. If the values fall outside the ranges, then the evaluation function returns FALSE, indicating an inconsistency in the data. The primitive geometric relationships are combined into conjunctions to describe the complex geometric relationships between objects.

Successful evaluation of the conjunction results in hypotheses for the remaining slots of the concepts. We have used this model to define such concepts as a roof and a building wall for understanding city scenes, a similar task domain to aerial photointerpretation.

3.1.4. 3DFORM

Three-dimensional representation of objects is necessary for many applications of vision such as robot navigation and 3-D change detection. Geometric reasoning is also an important capability, since geometric relationships among object parts are a rich source of knowledge and constraint in image analysis. Unfortunately, past systems for geometric representation and reasoning have not had sufficient flexibility and efficiency to be generally applied for computer vision.

Motivated by our work with the 3D Mosaic system, we began work on a system, called 3DFORM (Frame-based Object Recognition and Modeling), that has the desirable properties of generality, efficiency, and extensibility for computer vision applications [Walker et. al. 87]. This system includes a number of features that make it an improvement over past systems. 3DFORM uses frames to model object parts such as buildings and walls, geometric features such as lines and planes, and geometric relationships such as parallel or perpendicular. The system includes explicit modeling of the projections from the 3-D scene to the 2-D image and back, which allows a program to reason back and forth as needed. Active procedures can be attached to the frames to dynamically compute values as needed. For example, a line has an active procedure to compute its direction vector from known points on the line; this procedure would be invoked only when the direction vector is needed for other computations.

Since the order of computation is controlled largely by accessing objects' attribute values, the system can perform top-down and bottom-up reasoning as needed. This allows an efficient system that can perform the most reliable computations first, using the resulting constraints to guide the interpretation of more questionable or ambiguous data. There is no need for an external "focus of attention" mechanism, which in past systems has sometimes been a complex and problematic item to construct.

In 3DFORM, both objects and relationships are explicitly represented. Thus, extending the system to handle additional kinds of objects and relationships involves adding new frames but does not require modification of the existing system. This makes 3DFORM a relatively easy system to extend or tailor for a specific application domain. We have applied it to aerial photointerpretation, finding buildings from very sparse initial information with good success.

3.1.5. Parallel vision on Warp

The prototype Warp machine is an important research tool in our Image Understanding effort. Warp is CMU's Systolic Array Machine providing 100 MFLOP. The Warp group has been developing vision software for us to use in our vision research. This is one of the first examples of a parallel computer being used to advance research in an application area. To date, we have achieved the following:

- Several demonstrations of Warp's use for stereo vision, ERIM laser range scanner data, NMR image processing, signal processing, and other vision algorithms.

- A library based on the Spider Fortran subroutine library, all written in the Warp programming language W2. The current library includes about 80 different Warp programs, covering edge detection, smoothing, image operations, Fourier transforms, and so on. The actual number of routines in the Spider library covered by these Warp programs is about 100.
- We have developed a special-purpose programming language, Apply, in which low-level vision (local operation) programs can be written quickly and efficiently. By simply describing the local operations on a local window, the Apply compiler can generate code for the Warp machine (in W2) which executes the operations on the whole image efficiently. The compiler can also generate code in C under UNIX, which allows debugging algorithms off Warp [Hamey et al. 87].

As the software environment for Warp improves, it is becoming a tool for vision research, not merely for demonstrations of architectural concepts. In his research on analyzing repetitive textures (see section 3.3.3), Hamey needed to detect local point symmetry to locate the texture elements. Point symmetry is detected by an analysis of variance (ANOVA) statistical test which is applied to a window surrounding each pixel location. The ANOVA method consists of partitioning the variance of the data into two portions: that which is explained by the model and that which remains unexplained. The method is to be applied at each pixel location to measure point symmetry. Local peaks in an image of a symmetry measure values representing points of local symmetry. This analysis requires a large amount of computation.

The Warp implementation of this algorithm performs 346 million multiplications and 519 million additions. The prototype Warp processes a 512×512 image in 30 seconds. The same processing would take more than an hour on a Sun-3. This speedup was critical to Hamey's research progress.

3.2. Algorithms for three-dimensional scene analysis

Efficiently matching a model description with visual sensory input (images or ranges) forms the central part of recognition procedures. We have studied algorithms for generating 3-D data by stereo analysis, segmenting 3-D data, and automatically generating object recognition algorithms.

3.2.1. Dynamic programming in stereo vision

One of the most useful methods for obtaining 3-D depth information from images is stereo vision. The key problem in stereo is finding corresponding points in the left and right images so that the depth can be computed by triangulation using a known camera model. This stereo matching problem can be cast as a search problem.

Our dynamic programming stereo algorithm uses both intra- and inter-scanline search to obtain a disparity map, beginning with gray-scale real world images [Ohta and Kanade 85]. In 1985, we developed a faster version of the algorithm using a coarse-to-fine multi-resolution search strategy [Szeliski 85].

The images are first preprocessed using the DOLP Transform to build an image pyramid. Low-pass (blurred) images are used to calculate the cost function used by the stereo matcher, while band-pass images are used to extract the edges. The stereo matching algorithm is then applied to the coarsest (smallest) image, resulting in a list of matched edges that is used to constrain the stereo matching of the next finer (larger) level.

The matching proceeds until the solution for the finest level is obtained. The combined processing time of pyramid creation and multi-resolution search is much lower than that of single-resolution processing, since the constraints from the previous level greatly reduce the search space of the current level. In practice, the multi-resolution method was 2.5 times faster than the single-resolution method. The results for the single- and multi-resolution versions are similar in quality.

3.2.2. Trinocular stereo vision

In 1985, we developed a trinocular stereo algorithm, a variation of edge-based stereo algorithms using three cameras instead of two to reduce erroneous matches [Milenkovic and Kanade 85]. The trinocular matching algorithm performs better even though it uses the same order of computing resources as the binocular method.

The third view provided by trinocular stereo vision aids in selecting matching pairs of edge points from the first two views by providing a positional constraint. In addition to this constraint, trinocular stereo provides two constraint principles for use in determining correct matches. The first principle constrains the orientations of the matched edge pixels, and the second principle constrains the image intensity values in the regions surrounding the edge pixels.

We have applied the trinocular stereo algorithm to both real and synthetic images. In general, our trinocular method can match better than the best binocular method. The key features of the trinocular stereo vision include that it does not require the continuity assumption (i.e. neighboring edge pixels have similar disparities), and that it can handle reversals (i.e. the case where the matching pixels' order of appearance reverses between images).

3.2.3. Range data segmentation

One area we are pursuing is developing a 3-D vision system that provides a description of an unknown environment to a mobile robot. This description, a three-dimensional map of the observed scene in which regions are labeled as accessible terrain, objects, etc., can provide the necessary information for path planning and landmark recognition. As part of our efforts, Hebert developed several range data segmentation algorithms [Hebert and Kanade 86]. We have used the segmentation programs to produce input for path planning programs of the Terregator mobile robot.

We use a state-of-the-art sensing device, the ERIM scanner, that can produce 64×256

range images with an accuracy of 0.4 feet at a frame rate of two images per second. This sensor combines a large field of view (30 degrees horizontal and 40 degrees vertical) and a fast acquisition rate, making it suitable for outdoor imagery analysis.

Our algorithms produce three types of features: 3-D edges, accessible and non-accessible terrain regions, and obstacles divided into planar regions. The final segmentation product is a graph of edges, regions, and objects. The segmentation algorithms proceed by first extracting low-level attributes such as edge points, surface normals, and surface curvature. Then each attribute is used to derive an intermediate segmentation. Finally, the intermediate segmentations are merged together to form a consistent scene description. The complete segmentation takes about one minute on a VAX-785. We plan to reduce this computation time by using the Warp systolic array processor.

The range data segmentation techniques described so far proceed by independently processing one image at a time. In addition to this independent processing, the system can develop a global map by accumulating information from consecutive images. As the vehicle moves, the robot obtains an image every one to ten meters and consecutive images are registered with respect to the previous ones. Matching proceeds by finding the best match between the features produced by the segmentation program. This matching, in turn, provides an estimate of the current image's 3-D position with respect to the global map. The sequence of merged images then forms a global map of all the terrain the robot has seen.

We have tested the matching algorithms in a real outdoor navigation environment using a sensor mounted on a mobile testbed robot. The results indicate that active range data processing is suitable for navigation through an unknown environment.

3.2.4. Automatic generation of object recognition algorithms

Historically, computer vision programs have been "hand" written by a vision programmer. An alternative approach is to develop a general model-based vision program that takes a model of the object and recognizes the scene by reasoning about various properties and relationships based on the model. We developed an approach that provides a third alternative: develop a general program that takes a model of an object and generates a specialized run-time program tailored for the object [Ikeuchi 87]. We have applied this method in a task for bin-picking objects which include both planar and cylindrical surfaces.

The program analyzes the apparent object shapes from all possible viewing directions. These shapes are classified into groups based on dominant visible faces and other features. Based on the grouping, recognition algorithms are generated in the form of an interpretation tree. The interpretation tree classifies a target region into a representative shape group, and then determines the precise attitude of the object within that group. We have developed a set of rules for determining the appropriate features and the order in which the geometric modeler will use them to generate an efficient and reliable interpretation tree.

This approach will also have an impact on automatic learning in vision. During the course of the research, Ikeuchi developed a set of rules (mostly heuristic) which guide the decisions about what features to use in what order to generate an efficient and reliable interpretation tree. Currently, the interpretation trees are represented by semi-automatically written Lisp programs. We are developing a new approach that generates interpretation trees represented by object-oriented programming.

3.2.5. Recognition of 3-D objects based on solid models

Different sensors such as a video camera, light-stripe projector, or synthetic aperture radar have very different properties and respond to different types of object features under different circumstances. Since past research in computer vision assumed a single type of sensor, existing programs are restricted to the sensor for which they were designed. Our research in automatically generating object recognition algorithms has the potential to break out of this restriction by using an explicit sensor model in addition to the solid object model. In this way, we will be able to automatically generate appropriate object recognition algorithms from several different sensors. This capability will be important for sensor fusion or integration tasks that involve the use of many sensors to recognize a single object, and will also be important in robot system design as an automated aid to sensor selection for specific applications.

We have developed a model for sensor properties that can specify two important characteristics: *detectability* and *reliability*. Detectability refers to the kind of features that can be detected, such as faces, edges, and vertices. For example, an edge detector is sensitive to edges; a laser range scanner is sensitive to faces; synthetic aperture radar is most sensitive to vertices. We have developed a uniform representation for such detectability properties that allows many different sensor modalities to be described in a single framework. Reliability specifies how reliable the detection process is, and how errors are propagated from the measured data to the inferred geometric features.

We have used this sensor modeling methodology to construct a survey of commonly available sensors, and have produced detailed descriptions of photometric stereo and light-stripe range finders as examples. We plan to include the use of these sensor models in conjunction with our previous methodology for recognizing objects from solid models, to produce a system that can operate with diverse sensors.

3.3. Inferring shape and surface properties

Developing computational techniques for recovering scene and shape information from images remains a basic computer vision research area. Working vision theories must build on sound models of geometric, optical, and statistical processes. Research must be undertaken to evaluate the theories in controlled environments to understand their scope and limitations, such as requisite measurement precision, illumination conditions, surface shape, etc. Finally, we can apply the theories to real situations with the

appropriate instrumentation and assumptions identified. Toward these goals, we have established the Calibrated Imaging Laboratory. Using our new facility, we have been investigating three basic vision areas: color, texture, and motion.

3.3.1. Calibrated imaging laboratory

In 1986, we established the Calibrated Imaging Laboratory (CIL), a facility for high-precision imaging with accurate ground truth data [Shafer 85]. The CIL bridges the gap between vision theories, which typically depend on idealized models about the world, and applications, which must function on real images. Real images are provided in a controlled environment, with the ability to incrementally add more complexity to the imaging situation and the scene. In all cases, accurate ground truth data make it possible to quantitatively evaluate the performance of methods used for image analysis.

The facilities of the CIL include:

- A *variety of cameras* including sets of color and other filters, R-G-B color cameras, and a high-precision camera yielding 512x512x8-bit images that are nearly noise-free (repeatable) and linearly related to scene radiance, using color filters in a filter wheel.
- *Calibration Data* provided by appropriate tools including photometers, precision targets, and calibration camera filters.
- *Accurate Ground Truth Data* given by an optical table with precision position control devices and surveyors' transits for position measurement.
- *Flexible Lighting Control* with a near-point light source (arc lamp) for precision shadow analysis, and a complete track lighting system for flexible general illumination.
- *Background Reflection Control* in a room with a black ceiling, black carpet, and black or white curtains, with other colored backdrops as needed.
- *Geometric control of the camera and object positions*, with a mobile platform to allow for controlled position, motion, and stereo configurations.
- A *variety of test objects* including calibration materials, simple objects for color and texture studies, and a highly detailed landscape model for studying images of a complex environment within the laboratory.

Our goal is to provide images with every bit noise-free and with ground truth data that allows any pixel value to be exactly calculated from direct measurements of the scene. The CIL has already provided data for several vision projects, including our studies of color and highlights, color edges, motion, and image segmentation. We have also provided tapes of images for other universities, and we have provided assistance for other labs in deciding what equipment to obtain, such as cameras and color filters.

3.3.2. Color

Using the facilities of the Calibrated Imaging Laboratory, we have been developing techniques for using color information in a manner which is sound computationally and physically. Current work includes measurement of gloss components from color images and extraction of color edges.

Gloss from color

Image segmentation methods that are widely used today are confused by artifacts such as highlights, because they are not based on any physical model of these phenomena [Klinker et al. 87]. We have developed and implemented a method for automatically separating highlight reflection from matte object reflection. By exploiting the color difference between object color and highlight color, our algorithm generates two intrinsic images from one color image of a scene, one showing the scene without highlights and the other showing only the highlights. The successful modeling of highlight reflection can provide a useful preprocessor for: stereo and motion analysis, direct geometric shape inference, color image segmentation, and material type classification.

Our work is based on a spectral theory of light reflection from dielectric materials. The theory describes the color at each point as a linear combination of the object color and the highlight color. According to this model of light reflection, the color data of all points from one object forms a planar cluster in the color space. The cluster shape is determined by the object and highlight colors and by the object shape and illumination geometry. We use the shape of such clusters to determine the amount of highlight reflection and matte object reflection at each image point. We have successfully run this method on several real images.

Color edge detection

Edge detection is one of the most useful steps in early vision. We have been studying how to improve it by using color data instead of black-and-white images. As our test case, we selected Canny's algorithm, which finds high-contrast edges in smoothed images.

In general, the color edges are noticeably better than edges from intensity images as evaluated by human judgment. The color version of the Canny operator differs in that the pixel value is a color vector $\boldsymbol{C}=[R\ G\ B]$. The color version of the Canny operator calculates the x- and y-derivatives of each color band independently after smoothing the images, then calculates the magnitude and direction of each edge using the eigenvectors of the local covariance matrix at each point. This method is theoretically sound, but as a practical method it is computationally expensive. We have also developed a less expensive, general method for creating color operators from multi-stage intensity operators, such as edge detectors, using a color distance metric.

We have evaluated the theoretical color Canny operator and all of the non-trivial multi-stage color operators on a set of images of our landscape model.

By visually comparing these operators' output with each other and with the output of the Canny operator applied to intensity images of these scenes, we observed the following:

- The color edges are consistently better than edges from intensity images, though most (over 90%) of the edges are about the same.
- The best multi-stage operator seems to be calculating the magnitude and direction independently for each color band and then selecting the edge with the maximum magnitude. This operator produced almost exactly the same edges as the theoretical operator based on Jacobian analysis.
- The quality of the input image was very important; producing marginally better quality input images yielded substantially better results.

We developed similar color operators for stereo feature point detection and matching, and have greatly reduced the matching error rate using color.

Supervised color constancy

Color constancy—the recognition of a color independent ot the illumination—is one of the fundamental problems in color vision. Traditional color constancy methods attempt to compensate for the color of illumination by assuming some heuristic such as that the average color in the image is gray. This provides three measurements (red-green-blue) to use in the color correction. In our method, a standard color chart is placed in the scene to provide more comprehensive reference data. The color chart has 24 blocks with three color measurements on each (red-green-blue), yielding 72 measurements. These measurements are used in a system of linear equations to solve for coefficients of the illumination, yielding a tremendous improvement in the ability to model the details of the illumination color. Based on this model, the color of an object in colored illumination can then be calculated. This method is far more accurate than standard color constancy approaches, but requires the use of the color chart as a reference standard. For this reason, we call the method "Supervised Color Constancy." Our future work will include experimental verification of the method, which will take place when the necessary equipment (spectroradiometer) arrives in our lab.

3.3.3. Texture

In addition to color, texture is a vital clue to object properties for low-level vision. To understand texture, we are studying the perception of regular texture repetitions. The central problem in texture analysis is a chicken-and-egg problem: the texture element is difficult to define until the repetition has been detected, but at the same time the repetition cannot be found until the texture element is defined. We have developed a way to address this problem by identifying several potential features to define the texture element, then looking for all possible repetitions in these features. Some of the features we use include constant intensity regions and corners.

After identifying the potential features, we look for local repetitions in them. This is done by forming a histogram around each feature point of the vectors leading to other nearby feature points. These histograms are analyzed to discover which vectors occur most frequently. The shortest of these vectors indicates the direction and distance of the repetition. If this repetition applies over a sufficient number of feature points, it indicates a region of the image containing a repetition. Because the method is local in

nature, it can deal with distortions such as a checkered pattern on fabric or a perspective texture gradient (foreshortening) on a tall building. This method has been applied to several images of textured objects with good results. We have now determined some new constraints that identify the dominant features within each repetition pattern. We plan to incorporate these constraints in our future work.

3.3.4. Motion

Vision is a vital source of information for mobile robots. Vision can be used to determine vehicle motion, detect obstacles, and recognize objects. Unfortunately, the inherent uncertainty in vehicle motion and 3-D vision has limited its usefulness for mobile robots in the past. We are addressing these limitations by studying and modeling this uncertainty directly, and by developing motion stereo methods to reduce the uncertainty.

Determining vehicle motion by stereo vision

In many situations it is desirable to estimate vehicle motion directly from information in the sequence of images provided by the cameras, rather than to obtain it from external sources. Prior performance on this task has been very poor due to inadequate modeling of the inherent uncertainty in imaging due to finite pixel size. In 1985, we developed a method for incorporating adequate models into motion estimates [Matthies and Shafer 86]. Our work applies directly to the visual mapping and navigation problems in autonomous vehicles.

Previous efforts based on this type of algorithm have performed poorly because the measurement error inherent in triangulation was not adequately modeled in the procedure that determined the 3-D transformations. Previous error models were based on scalar descriptions of the uncertainty in the position of each 3-D point. This treats the uncertainty in the position of a point as spherically distributed in space. However, with triangulation the actual uncertainty can be highly non-spherical. Treating each 3-D point as the mean of a 3-D normal distribution can obtain a much better error model. The uncertainty in the location of the point can then be modeled by the covariance of the distribution. We have developed a new method for estimating the motion of the cameras based on this model.

We demonstrated this method in a scenario in which the motion of a mobile robot was determined from image sequences provided by onboard stereo cameras. The algorithm has two main steps:

- Building simple 3-D scene models from each stereo pair
- Finding the 3-D transformation that best maps the model built at each robot location into the model built at the next location.

The transformations produced by the second step are the desired output of the algorithm. The 3-D models consist of points whose positions are computed by triangulation from corresponding features in the stereo pairs. We have tested the new algorithm in simulations and in live operation with real images. The results show reductions in es-

timation error by factors of three to ten or more, depending on the distance of the points from the cameras. The greater the distance, the greater the difference in performance. This shows that error modeling is important for obtaining high performance in algorithms for visual range and motion estimation.

Modeling uncertainty in representations for low-level vision

Most work in computer vision has ignored uncertainty in the past, leading to methods that are brittle and tend to fail when confronted with the noise that arises in real data. We are developing methods for explicitly modeling uncertainty and using the noise properties of the sensor, measured over multiple images, to produce improved results.

We have been working on modeling uncertainty in low-level dense representations, such as depth maps and optical flow (velocity) maps, by means of Bayesian models. This approach has the potential to provide more accurate surface descriptions from image sequences than currently available surface methods. Researchers have already used the Bayesian modeling in low-level vision processing. However, one of the distinguishing features of our approach is that it uses Bayesian modeling not only to recover optimal estimates (as is currently done), but also to calculate the uncertainty associated with these estimates.

Low-level representations are usually derived from the input image(s) using "shape-from-X" methods such as stereo or shape from shading. These methods usually yield data that is sparse, (e.g. stereo) or underconstrained (e.g. shape from shading). Two approaches to overcoming this problem are currently popular. The first—regularization—reformulates the problem in terms of the minimization of an energy functional. Smoothness constraints, in the form of added energy terms, are used to guarantee a unique and well-behaved solution. The second method, Bayesian estimation, assumes both a probabilistic prior model for the data being estimated and a probabilistic imaging model relating the data to the sensed image [Szeliski 86]. An optimal estimate (e.g. Maximum A Posteriori or Minimum Variance) can then be obtained.

The main emphasis of this research is to study how the uncertainty inherent in the Bayesian modeling approach can be estimated and used in further processing. Previous work, both in regularization and Bayesian estimation, has concentrated solely on obtaining a single optimal estimate of the underlying field. However, the Bayesian approach actually (implicitly) defines a whole distribution conditional on the sensed data. For example, when regularization is used, the resulting distribution is a multivariate correlated Gaussian image. Thus knowing both the mean (minimum variance estimate) and the covariance fully characterizes the distribution. The estimated uncertainty can then be used for further processing, such as integration with new data, or matching to a model. Current research is focusing on the former application (using Kalman filtering), as well as examining the use of alternate representations that better model the uncertainty.

3.4. Bibliography

[Gross et al. 85] Gross,T., H.T. Kung, M. Lam, and J. Webb.
Warp as a machine for low-level vision .
In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, IEEE, March, 1985.

Warp is a programmable systolic array processor. One of its objectives is to support computer vision research. This paper shows how the Warp architecture can be used to fulfill the computational needs of low-level vision.

We study the characteristics of low-level vision algorithms and show they lead to requirements for computer architecture. The requirements are met by Warp. We then describe how the Warp system can be used. Warp programs can be classified in two ways: chained versus severed, and heterogeneous versus homogeneous. Chained and severed characterize the degree of interprocessor dependency, while heterogeneous and homogeneous characterize the degree of similarity between programs on individual processors. Taken in combination, these classes give four user models. Sophisticated programming tools are needed to support these user models.

[Hamey et al. 87] Hamey, L.G.C., J.A. Webb, and I.C. Wu.
Low-level vision on Warp and the Apply programming module,
In J. Kowalik, *Parallel Computation and Computers for Artificial Intelligence*. Kluwer Academic Publishers, 1987.

In the course of implementing low-level (image to image) vision algorithms on Warp, we have understood the mapping of this important class of algorithms well enough so that the programming of these algorithms is now a straightforward and stereotypical task. The partitioning method used is input partitioning. This seems to consistently provide an efficient implementation of this class of algorithms, which is, moreover, quite natural for the programmer. We have developed a specialized programming language, called Apply, which reduces the problem of writing the algorithm for this class of programs to the task of writing the function to be applied to a window around a single pixel. Apply provides a method for programming Warp in these applications which is extremely easy, consistent, and efficient. Apply is application specific, but machine independent—it appears possible to implement versions of Apply which run efficiently on a wide variety of computers, including uniprocessors, bit-serial processor arrays, and distributed memory machines. Apply is therefore a significant aid to the programmer, which allows him to program efficiently and consistently in a well-defined application area, for a specialized type of machine, without restricting his code to be run just on that machine.

[Hebert and Kanade 86]
Hebert,M., and T. Kanade.
Outdoor scene analysis using range data.
In *IEEE International Conference on Robotics and Automation*, 1986.
Also available in *Proceedings of DARPA Image Understanding Workshop, DARPA*, December 1985.
This paper describes techniques for outdoor scene analysis using range data. The purpose of these techniques is to build a 3-D representation of the environment of a mobile robot equipped with a range sensor. Algorithms are presented for scene segmentation, object detection, map building, and object recognition.
We present results obtained in an outdoor navigation environment in which a laser range finder is mounted on a vehicle. These results have been successfully applied to the problem o path planning through obstacles.

[Herman 85]
Herman,M.
Representation and incremental construction of a three-dimensional scene model.
Technical Report CMU-CS-85-103, Carnegie Mellon University Computer Science Department,
January, 1985.
The representation, construction, and updating of the 3D scene model derived by the 3D Mosaic scene understanding system is described. The scene model is a surface-based description of an urban scene, and is incrementally acquired from a sequence of images obtained from multiple viewpoints. Each view of the scene undergoes analysis which results in a 3D wire-frame description that represents portions of edges and vertices of buildings. The initial model, constructed from the wire frames obtained from the first view, represents an initial approximation of the scene. As each successive view is processed, the model is incrementally updated and gradually becomes more accurate and complete. Task-specific knowledge is used to construct and update the model from the wire frames. At any point along its development, the model represents the current understanding of the scene and may be used for tasks such as matching, display generation, planning paths through the scene, and making other decisions dealing with the scene environment.
The model is represented as a graph in terms of symbolic primitives such as faces, edges, vertices, and their topology and geometry. This permits the representation of partially complete, planar-faced objects. Because incremental modifications to the model must be easy to perform, the model contains mechanisms to (1) add primitives in a manner such that constraints on geometry imposed by these

additions are propagated throughout the model, and (2) modify and delete primitives if discrepancies arise between newly derived and current information. The model also contains mechanisms that permit the generation, addition, and deletion of hypotheses for parts of the scene for which there is little data.

[Ikeuchi 87] Ikeuchi,K.
Precompiling a geometrical model into an interpretation tree for object recognition in bin-picking tasks.
In *Image Understanding Workshop*, DARPA, February, 1987.
Given a 3D solid model of an object, we first generate apparent shapes of an object under various viewer directions. Those apparent shapes are then classified into groups (representative attitudes) based on dominant visible faces and other features. Based on the grouping, recognition algorithms are generated in the form of an interpretation tree. The interpretation tree consists of two parts: the first part for classifying a target region in an image into one of the shape groups, and the second part for determining the precise attitude of the object within that group. We have developed a set of rules to find out what appropriate features are to be used in what order to generate an efficient and reliable interpretation tree. Features used in the interpretation tree include inertia of region, relationship to the neighboring regions, position and orientation of edges, and extended Gaussian images. This method has been applied in a task for bin-picking objects which include both planar and cylindrical surfaces. As sensory data, we have used surface orientations from photometric stereo, depth from binocular stereo using oriented-region matching, and edges from an intensity image.

[Ikeuchi et al. 86] Ikeuchi,K., H.K. Nishihara, B.K.P. Horn, P. Sobalvarro, and S. Nagata.
Determining grasp configurations using photometric stereo and the PRISM binocular stereo system.
The International Journal of Robotics Research 5(1):46-65, 1986.
This paper describes a system which locates and grasps parts from a pile. The system uses photometric stereo and binocular stereo as vision input tools. Photometric stereo is used to make surface orientation measurements. With this information the camera field is segmented into isolated regions of a continuous smooth surface. One of these regions is then selected as the target region. The attitude of the physical object associated with the target region is determined by histograming surface orientations over that region and comparing them with stored histograms obtained from prototypical objects. Range information, not

available from photometric stereo, is obtained by the PRISM binocular stereo system. A collision-free grasp configuration is computed and executed using the attitude and range data.

[Kanade and Shafer 85]

Kanade,T., and S. Shafer.
Image understanding research at CMU.
In *Proceedings of the DARPA Image Understanding Workshop*,
DARPA, December, 1985.

In the CMU Image Understanding Program we have been working on both the basic issues in understanding vision processes that deal with images and shapes, and the system issues in developing demonstrable vision systems. This report reviews our progress since the October 1984 workshop proceedings. The highlights in our program include the following:

V. Milenkovic has developed and edge-based trinocular (three-camera) stereo method for computing depth from images.

R. Szeliski has extended Ohta and Kanade's dynamic programming stereo method to use a coarse-to-fine multi-resolution search strategy.

E. Walker is analyzing the object-independent geometric reasoning rules in the 3D Mosaic system.

S. Shafer is constructing the Calibrated Imaging Lab which will provide high-precision images for stereo, motion, shape analysis, and photometric analysis.

M. Hebert has developed several algorithms for analysis of outdoor range images to extract edges, planar faces of objects, and terrain patches.

L. Matties is analyzing motion stereo image sequences using a statistical analysis of uncertainty to yield high accuracy.

D. McKeown has started a Digital Mapping Laboratory as a focal point for work in aerial photo interpretation, cartography, and computer vision. Current projects include MAPS, a large-scale image/map database system, SPAM, a rule-based system for airport scene interpretation, and ARF, a system for finding and tracking roads in aerial imagery.

J. Webb is developing a high-performance vision system on a systolic machine, Warp, which will be actively used by the vision community at CMU. The Warp hardware is a reality, and almost a dozen implementation programs are now running.

G. Klinker has implemented the FIDO mobile robot vision and navigation system using the WARP.

C. Thorpe, R. Wallace, and A. Stentz are working on the Strategic Computing Vision project, building an intelligent mobile robot for outdoor operation.

[Klinker et al. 87] Klinker,G., S. Shafer, and T. Kanade.
Measurement of gloss from color images.
In *Proceedings of the Conference on Appearance*, ISCC, February, 1987.

It is the goal of computer vision to automatically recover the three-dimensional objects in the scene from images. Most current research in computer vision analyzes black-and-white images and assumes that the objects in the scene are matte. Brightness variation in the image is then attributed to variations of surface orientation on the objects and to material changes at object boundaries. However, real scenes generally contain glossy objects, as well as matte objects. Highlights on glossy objects provide additional brightness variations in the images and are commonly misinterpreted by current computer vision systems. Shafer has introduced a spectrally-based dichromatic reflection model that accounts for both diffuse and specular reflection. Along with the model, we describe a method that exploits the model to detect and remove highlights from color images. This approach thus provides a useful preprocessor for many areas of computer vision. We present the results of applying the technique to real images.

[Kung and Webb 86]
Kung,H.T., and J. Webb.
Mapping image processing operations onto a linear systolic machine,
In Gouda, M.C., *Distributed Computing.* Springer-Verlag, 1986.

A high-performance systolic machine, called Warp, is operational at Carnegie Mellon. The machine has a programmable systolic array of linearly connected cells, each capable of performing 10 million floating-point operations per second. Many image processing operations have been programmed on the machine. This programming experience has yielded new insights in the mapping of image processing operations onto a parallel computer. This paper identifies three major mapping methods that are particularly suited to a Warp-like parallel machine using a linear array of processing elements. These mapping methods correspond to partitioning of output dataset, and partitioning of computation along the time domain (pipelining). Parallel implementations of several important image processing operations are presented to illustrate the mapping methods. These operations include the Fast Fourier Transform (FFT), connected component labeling, Hough transform, image warping and relaxation.

[Lucas 84] Lucas,B.D.
Generalized image matching by the method of differences.
Technical Report CMU-CS-85-160, Carnegie Mellon University Computer Science Department,
July, 1984.

Image matching refers to aligning two similar images related by a transformation such as a translation, rotation, etc. In its general form image matching is a problem of estimating the parameters that determine that transformation. These parameters may be a few global parameters or a field of parameters describing local transformations.

This thesis explores in theory and by experiment image matching by the *method of differences*. The method uses intensity differences between the images together with the spatial intensity gradient to obtain from each image point a linear constraint on the match parameters; combining constraints from many points yields a parameter estimate. The method is particularly suitable where an initial estimate of the match parameters is available. In such cases it eliminates search which can be costly, particularly in multi-dimensional parameter spaces. Essential to the technique are smoothing, which increases the range of validity of the constraint provided by the gradient, and inteation, because the parameter estimate is an approximation. Smoothing increases the range of convergence but it decreases accuracy, so a coarse-fine approach is needed. A theoretical analysis supports these claims and provides a means for predicting the algorithm's behavior.

[Matthies and Shafer 86]
Matthies, L.H. and S. Shafer.
Error modeling in stereo navigation.
In *1986 Proceedings of the Fall Joint Computer Conference*,
ACM/IEEE, November, 1986.

In stereo navigation, a mobile robot estimates its position by tracking landmarks with onboard cameras. Previous systems for stereo navigation have suffered from poor accuracy, in part because they relied on scalar models of measurement error in triangulation. The authors show that using 3-D Gaussian distributions to model triangulation error leads to much better performance. They also show how to compute the error model from image correspondences, estimate robot motion between frames, and update the global positions of the robot and the landmarks over time. Simulations show that compared to scalar error models the 3-D Gaussian reduces the variance in robot position estimates and better distinguishes rotational from translational motion. A short indoor run with real images supported these conclusions and computed the final robot

position to within 2% of distance and one degree of orientation. These results illustrate the importance of error modelling in stereo vision for this and other applications.

[McKeown et al. 85a]
McKeown, D.M., C.A. McVay, and B.D. Lucas.
Stereo verification in aerial image analysis.
In *Proceedings of DARPA Image Understanding Workshop*, DARPA, December, 1985.
Also available as Technical Report CMU-CS-85-139.

Computer vision systems that attempt to extract cultural features from aerial imagery are often forced to interpret segmentations where the actual features are broken into numerous segments or fragments. For example, roads and road-like features are difficult to completely segment due to occlusions, poor contrast with their surroundings, and changes in surface material. Often the nature of the segmentation process is designed to err toward oversegmentation of the image, since the joining of feature descriptions is believed to be simpler than their decomposition. No matter what the cause, it is necessary to aggregate these incomplete segmentations, filling in missing information, in order to reason about the overall scene interpretation. This paper describes a method to select sets of such fragments as candidates for alignment into a single region, as well as a procedure to generate new linear regions that are linked composites of the original sets of fragments. Portions of the composite region that lie between pairs of the original fragments are approximated with a spline. The resulting composite region can be used to predict the areas in which to search for missing components of the cultural feature.

[McKeown et al. 85b]
McKeown,D.M. Jr., W.A. Harvey, and J. McDermott.
Rule-based interpretation of aerial imagery.
IEEE Transactions on Pattern Analysis and Machine Intelligence 7(5):570-585, 1985.

In this paper, we describe the organization of a rule-based system, SPAM, that uses map and domain-specific knowledge to interpret airport scenes. This research investigates the use of a rule-based system for the control of image processing and interpretation of results with respect to a world model, as well as the representation of the world model within an image/map database. We present results of a high resolution airport scene where the image segmentation has been performed by a human, and by a region-based image segmentation program. The results of the system's analysis is characterized by the labeling of individual regions in the image and the collection of these

regions into consistent interpretations of the major components of an airport model. These interpretations are ranked on the basis of their overall spatial and structural consistency. Some evaluations based on the results from three evolutionary versions of SPAM are presented.

[McVay et al. 85] McVay,C., B.D. Lucas, and D.M. McKeown.
Stereo verification in aerial image analysis.
Technical Report CMU-CS-85-139, Carnegie Mellon University Computer Science Department,
July, 1985.

This paper describes a flexible stereo verification system, STEREOSYS, and its application to the analysis of high resolution aerial photography. Stereo verification refers to the verification of hypotheses about a scene by stereo analysis of the scene. Unlike stereo interpretation, stereo verification requires only coarse indications of three-dimensional structure. In the case of aerial photography, this means coarse indications of the heights of objects above their surroundings. This requirement, together with requirements for robustness and for dense height measurements, shape the decision about the stereo system to use. This paper discusses these design issues and details the results of implementation.

[Milenkovic and Kanade 85]
Milenkovic, V. and T. Kanade.
Trinocular vision using photometric orientation constraints.
In *Proceedings of the Image Understanding Workshop*, Pages 163-175. DARPA, December, 1985.

Trinocular vision is stereo using three non-collinear views. It has been shown in the literature that a third view aids in the selection of matching pairs of edge points from the first two views by providing a constraint on the positions of the points. In addition to this positional constraint, this paper proposes two new constraint principles for use in determining the set of correct matches. The first principle constrains the orientation of the matched edge pixels, and the second principle constrains the image intensity values in the regions surrounding the edge pixels. Statistical confidence measures and rejection thresholds are derived from these constraint principles in order to maximize the correct matches in the presence of error. An trinocular stereo algorithm based on these principles is described and applied to synthetic and real images with good results.

[Ohta and Kanade 85]
Ohta,Y., and T. Kanade.
Stereo by intra- and inter-scanline search using dynamic programming.
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, March, 1985.

This paper presents a stereo matching algorithm using the dynamic programming technique. The stereo matching problem, that is, obtaining a correspondence between right and left images, can be cast as a search problem. When a pair of stereo images is rectified, pairs of corresponding points can be searched for within the same scanlines. We call this search intra-scanline search. This intra-scanline search can be treated as the problem of finding a matching path on a two-dimensional (2D) search plane whose axes are the right and left scanlines. Vertically connected edges in the images provide consistency constraints across the 2D search planes. Inter-scanline search in a three-dimensional (3D) search space, which is a stack of the 2D search planes, is needed to utilize this constraint.

Our stereo matching algorithm uses edge-delimited intervals as elements to be matched, and employs the above mentioned two searches: one is inter-scanline search for possible correspondences of connected edges in right and left images and the other is intra-scanline search for correspondences of edge-delimited intervals on each scanline pair.

Dynamic programming is used for both searches which proceed simultaneously: the former supplies the consistency constraint to the latter while the latter supplies the matching score to the former. An interval-based similarity metric is used to compute the score.

The algorithm has been tested with different types of images including urban aerial images, synthesized images, and block scenes, and its computational requirement has been discussed.

[Shafer 85]
Shafer, S.A.
The Calibrated Imaging Lab under construction at CMU.
In *Proceedings of DARPA Image Understanding Workshop*, DARPA, December, 1985.

This document describes the Calibrated Imaging Laboratory, a facility for precision digital imaging under construction at CMU. The purpose of this lab is to provide images with accurate knowledge about ground truth (concerning the scene, illumination, and camera) so that computer vision theories and methods can be tested on real images and evaluated to determine how accurate they really are. The lab aims to provide ground truth data accurate, in the best

circumstances, to the nearest pixel geometrically and the nearest 8-bit pixel value photometrically. There are also many illumination and imaging facilities in the lab that provide increased flexibility or increased complexity of the visual situation, at a cost of reduced precision in the ground truth data.

To accomplish these goals, the lab includes mechanisms to carefully control and measure the direct and indirect illumination in the scene, the positions of objects, and the properties of the camera/digitizer system. Lighting can be provided by a near-point source (5 mm diameter aperture) for high precision, or by a general-purpose track lighting system for flexibility. The work area can be surrounded by black curtains etc. to reduce stray light and indirect illumination. The cameras include a very high-precision CCD camera on a static mount, and an X-Y-X-pan-tilt jig with multiple inexpensive CCDs aligned with each other. Surveyors' transits are used to measure positions of points in space, and other calibration materials are available for all types of camera property measurement. Color imaging by serial selection of filters is also available.

The lab is described as we currently envision it will be equipped when the facilities are operational; the current status is summarized at the end of the paper.

[Smith and Kanade 85]

Smith,D.R., and T. Kanade.
Autonomous scene description with range imagery.
Computer Vision, Graphics, and Image Processing 31(3):322-334, 1985.

This paper presents a program to produce object-centered 3-dimensional descriptions starting from point-wise 3D range data obtained by a light-stripe rangefinder. A careful geometric analysis shows that contours which appear in light-stripe range images can be classified into eight types, each with different characteristics in occluding vs occluded and different camera/illuminator relationships. Starting with detecting these contours in the iconic range image, the descriptions are generated moving up the hierarchy of contour, surface, object to scene. We use conical and cylindrical surfaces as primitives. In this process, we exploit the fact that coherent relationships, such as symmetry, collinearity, and being coaxial, which are present among lower-level elements in the hierarchy allow us to hypothesize upper-level elements. The resultant descriptions are used for matching and recognizing objects. The analysis program has been applied to complex scenes containing cups, pans, and toy shovels.

[Szeliski 85] Szeliski, R.
Multi-resolution stereo using dynamic programming.
IUS internal report , Carnegie Mellon University Computer Science Department,
May, 1985.

[Szeliski 86] Szeliski,R.
Cooperative algorithms for solving random-dot stereograms.
Technical Report CMU-CS-86-133, Carnegie Mellon University Computer Science Department,
June, 1986.
This report examines a number of parallel algorithms for solving random-dot stereograms. A new class of algorithms based on the Boltzmann Machine is introduced and compared to previously developed algorithms. The report includes a review of the stereo correspondence problem and of cooperative techniques for solving this problem. The use of energy functions for characterizing the computational problem, and the use of stochastic optimization techniques for solving the problem are explained.

[Tomita and Kanade 85]
Tomita,F. and T. Kanade.
A 3D vision system: generating and matching shape descriptions in range images,
In H. Hanafusa and H. Inoue, *Robotics Research*, Pages 35-42. MIT Press, 1985.
We have developed a vision system to recognize and locate three dimensional (3D) objects in range images. A light-stripe rangefinder image is first segmented into edges and surfaces. This segmentation is done in 3D space; edges are classified as either 3D straight lines or circular curves, and surfaces are either planar or conic. An object model consists of component edges and surfaces and their interrelationships. Our model representation can accommodate not only objects with rigid, fixed shape, but also objects with articulations between their parts, such as rotational-joint or linear-slide motions. The system supports interactive construction of object models. Using sample scenes, the object models can be generated and modified till they become satisfactory. The matching process is rather straightforward. A transformation from an object model to the scene is hypothesized by initially matching a few scene features with model features. The transformation is then tested with the rest of the features for verification.

[Walker and Herman 87]
Walker, E. and M. Herman.
Geometric reasoning for constructing 3-D scene descriptions from images.
In *Proceedings of the Workshop on Spatial Reasoning and Multisensor Fusion*, AAAI, October, 1987.
There are many applications for a vision system which derives a 3-dimensional model of a scene from one or more images and stores the model for easy retrieval and matching. Geometric reasoning is used at several levels of the derivation, as well as for the eventual matching. Experience with the 3D Mosaic system has shown that domain specific knowledge can be used to drive much of this geometric reasoning. A general framework for the representation and use of domain knowledge is proposed.

[Walker et. al. 87] Walker, E., M. Herman, and T. Kanade.
A framework for representing and reasoning about three-dimensional objects for vision.
In *Proceedings of the AAAI Workshop on Spatial Reasoning and Multisensor Fusion*, AAAI, October, 1987.
The capabilities for representing and reasoning about three-dimensional objects are essential for knowledge-based, 3-D photointerpretation systems that combine domain knowledge with image processing, as demonstrated by such systems as 3D Mosaic and Acronym. Three-dimensional representation of objects is necessary for many additional applications such as robot navigation and 3-D change detection. Geometric reasoning is especially important, since geometric relationships between object parts are a rich source of domain knowledge. A practical framework for geometric representation and reasoning must incorporate projections between a 2-D image and a 3-D scene, shape and surface properties of objects, and geometric and topological relationships between objects. In addition,it should allow easy modification and extension of the system's domain knowledge and be flexible enough to organize its reasoning efficiently to take advantage of the current available knowledge. We are developing such a framework, called the 3D FORM (Frame-based Object Recognition and Modeling) System. This system uses frames to represent objects such as buildings and walls, geometric features such as lines and planes, and geometric relationships such as parallel lines. Active procedures attached to the frames dynamically compute values as needed. Since the order of processing is controlled largely by accessing objects' slots, the system performs both top-down and bottom-up reasoning, depending on the current available knowledge. The FORM system is

being implemented using the CMU-built Framekit tool in Common Lisp. Examples of interpretation using a simple model of a building as a rectangular prism are presented.

4. MACHINE INTELLIGENCE

Machine Intelligence (MI) is the study of how to obtain intelligent action by computers. It encompasses both the attempt to discover the principles whereby intelligent action is possible and to construct computer systems that can perform tasks requiring intelligence. To achieve acceptable levels of performance, artificially intelligent systems increasingly need the ability to acquire, represent, and effectively utilize large amounts of knowledge. In the MI project, we have been investigating methods for acquiring, representing, and utilizing knowledge to obtain intelligent action by computers. For the past three years, we have concentrated our work in the following areas:

- *Knowledge-intensive systems*—Extending expert systems technology by attacking on several fronts the problem of how to develop systems that combine the power and efficiency of a knowledge-based system with the generality and robustness of a general problem solver.
- *Machine learning*—Investigating systems that can learn from the experience of solving a problem and develop techniques through which expert systems can acquire knowledge and skill.
- *Massively parallel architectures*—Exploring massively parallel architectures as means of meeting the computational demands imposed by large constraint-satisfaction problems, particularly in recognition tasks.
- *Rational control of reasoning*—Incorporating rationally controlled reasoning to increase the power, flexibility, and reliability of knowledge-based systems.

4.1. Knowledge-intensive systems

Current knowledge-based systems typically exhibit a rather narrow character—often described as *shallow*. Though substantial knowledge is piled up in the rules and can be released appropriately to perform a given task, the systems have no ability to reason further with that knowledge. They do not understand the basic semantics of the task domain. New rules are not learned from experience nor does behavior with the existing rules become automatically tuned. To point out such limitations is not to be hypercritical. Indeed, the important scientific discovery behind the success of AI knowledge-based expert systems is precisely that sufficient bodies of such shallow knowledge could be assembled, without any of the supporting reasoning and understanding ability, and still prove adequate to perform real consultation tasks in the medical and industrial world.

A major—and widely recognized—item on the agenda for expert systems research is to transform the current generation of systems so they are no longer shallow. Researchers differ on which aspect to tackle first, but candidates are plentiful: reasoning power; internal semantic models; learning ability; explanation ability. Increased reasoning power implies combinatorial search in conjunction with extensive knowledge bases. Internal semantic models imply large data structures and the corresponding processing power to manipulate them. Learning ability implies finally getting beyond the volume of

rules that can be encoded by hand, thus opening the way to systems with a hundred thousand and ultimately a million rules. Such immense rule bases will result not only from extending the scope and depth of knowledge in rule-based systems, but also from introducing automatic learning techniques. Automatic learning is likely to increase by an order of magnitude the density of rules that represent a given microbody of knowledge and its applicability.

4.1.1. Towards a better general problem solver

One of our research concerns is to understand what mechanisms are necessary for a system to demonstrate intelligent behavior, whether they are adequate for a wide range of tasks, and how they work together to form a general cognitive architecture. In the Soar project, we have developed a general architecture for solving problems and learning. Our ultimate goal is to give it all the capabilities it needs to display the full range of intelligent behavior. We have already realized some of these capabilities.

Soar represents all goal-oriented cognitive activity as a heuristic search of problem spaces. Its objective is to find a viable path from a given initial state to a desired final state. Soar and our research strategy for its development should be understood within the context of several decades of research on problem spaces and production systems. Productions, containing both search control and instructions for implementing primitive operators, represent Soar's long-term knowledge about a problem space. Failure to progress toward a solution produces an impasse. The architecture detects the impasse then establishes a subgoal to resolve it. When subgoal searches terminate, Soar builds productions (chunks) that summarize processing within the subgoal. It can then reuse its experience-derived chunks to speed future problem solving in equivalent situations.

The Soar project was originally located entirely at CMU, but in mid-1984, it became a distributed effort among CMU, Xerox PARC, and Stanford University (when Laird moved to PARC and Rosenbloom moved to Stanford). Much of Fall 1984 and Spring 1985 was spent establishing the research at these new locations. In June 1985, we summarized the part of the research program that focused on learning. We solidified our understanding of chunking in Soar (its learning mechanism) by showing that if Soar were given the appropriate tasks, it would automatically accomplish the macro-operator-learning scheme developed by Korf [Laird et al. 85a].

One significant demonstration of Soar's abilities was in 1985. Van de Brug, visiting CMU from DEC, examined a number of different task decompositions for configuring VAXes. Soar handled the same sized tasks as R1, the VAX-configuring expert system used by DEC, and did it within a factor of about two. Of greater interest, he showed how the basic task structure used by Soar could be the basis of a rationalized version of R1 (called Proto-R2) which could also form the basis of a knowledge-acquisition system (called Sear) for R1-like expert systems [vandeBrug et al. 85]. DEC used this scheme to restructure R1 in 1985.

Since 1986, Soar research at CMU has focused on three main areas, each of which

we believe is crucial to successfully investigating the nature of intelligence. First, we completed some architectural details that address aspects of intelligence for which the framework was uncommitted. Second, we continued to use Soar as the foundation of several systems that represent and apply substantial amounts of knowledge for problem solving in complex domains. Third, we began mapping our ideas about Soar into the immense body of data from cognitive psychology, with the intent of presenting a single system of mechanisms that could serve as a unified theory of human cognition. Each of these research foci is strongly influenced by work in the other areas and brings together distinct perspectives not often integrated within artificial intelligence.

For the first research focus, filling in the details of Soar, we concentrated on characterizing the implications of Soar's simple and uniform approach to problem solving and learning. For example, we addressed the following question: How should Soar acquire new problems to solve? The current method requires a "Soarware engineer" to analyze the problem and write a set of Soar productions implementing the problem spaces required. Eventually, we want Soar to build the productions automatically from new task descriptions. We tried several approaches to the task acquisition problem. The strategy we chose has two phases: a comprehension phase that parses the task description (given in pseudo-natural language), and an interpretation phase that executes the internal task representation, acquiring chunks as it resolves impasses. The chunks closely resemble task productions that we would have otherwise built by hand. Though this task acquisition method is still inefficient and under development, we have already demonstrated it in the missionaries-and-cannibals and the eight-puzzle tasks.

The second focus of our research is to employ Soar in building knowledge-intensive systems for "real" tasks. The aim is to duplicate our earlier success with R1-Soar, the partial reimplementation of the VAX-configuring expert system. We made significant progress towards developing knowledge-based systems in the automatic programming domain. Our system Designer-Soar reimplements and extends Designer, a prototype automatic algorithm designer. Our objective with Designer-Soar is to demonstrate that we can profitably integrate diverse knowledge about application domains, algorithm design, and general problem solving into a single framework for designing algorithms. It currently designs several simple set and numeric algorithms, producing data-flow configurations to describe them. A second system, Cypress-Soar, reproduces much of the behavior of the Cypress system (developed by D. Smith at Kestral Institute) that designs divide-and-conquer sorting algorithms. Cypress-Soar has already produced several novel results that combine performance, search, and learning in this domain.

Our third focus is to establish the Soar paradigm as a leading candidate for a unified theory of cognition. This long-term goal requires Soar's behavior to explain, or at least be consonant with, the psychological literature covering the complete spectrum of cognitive activity: problem solving, decision making, routine action, memory, learning, skill, perception, behavior, and language. Substantial parts of the scheme (problem spaces for problem solving, and chunking for learning and skill acquisition, for example) are already in place, and we elaborated other parts of a Soar-based cognitive theory. Progress was most significant on the specification of Soar's mechanisms for interacting

with the external world (the perception-cognition-motor interface), and the use of mental models. The former specifies that the perceptual and motor systems interact with encoding and decoding productions by communicating through working memory. The specification is strongly constrained by psychological reaction-time data, as well as the current structure of Soar. For the latter, we used Johnson-Laird's study of mental models as the basis of a simple system for working with logical syllogisms.

4.1.2. Acquiring knowledge for aerial image interpretation

Interpreting aerial photographs requires extensive knowledge about the scene under consideration. For example, just knowing the scene type—airport, suburban housing development, urban neighborhood—can aid in low- and intermediate-level image analysis and can drive high-level interpretation by constraining searches for plausibly consistent scene models. Building versatile, automated photointerpreters that incorporate such complex knowledge poses a challenging problem.

We have already hand-crafted one system that embodies expert photointerpretive knowledge. The SPAM system, developed primarily under Defense Mapping Agency sponsorship and partly with DARPA support, represents one of the few vertically integrated, knowledge-intensive systems. It utilizes knowledge during all interpretive phases, from pixels to symbolic objects, from segmenting an initial image to generating and describing a final scene model. While other systems such as ACRONYM and 3D Mosaic focus on geometric aspects of isolated objects and VISIONS performs two-dimensional segmentations based upon direct mappings between object color and view-specific spatial relationships, SPAM is unique in employing spatial, structural, and functional knowledge to perform 2-D and 3-D interpretation of complex scenes.

Our research addresses a broad set of topics within the overall knowledge acquisition framework. First and foremost, we are interested in automating the process through which an interpretation system can collect and represent new knowledge—to improve its performance on existing tasks or to gain proficiency on new tasks. Knowledge serves primarily to constrain the search for a plausible interpretation. For SPAM's original airport scene task, we relied mainly on spatial constraints from airport design books and, to a lesser extent, on relationships we observed in aerial imagery. Other task domains, such as aerial suburban scenes, may not have codified spatial guidelines, though they may exhibit similar patterns across many examples. To function effectively in such domains, a versatile system must allow users to identify and measure spatial relationships in representative imagery and then automatically compile the new information into the photointerpreter. This need for automated support is one of many and emphasizes the critical role that specialized tools play in assembling and organizing large knowledge bases.

We have already developed three powerful tools: an interactive user interface for acquiring scene and spatial knowledge, an automatic compiler that transforms new knowledge from a schema-based representation into productions directly executable by our interpretation system, and a performance analyzer that critiques the final image.

During the second half of 1986, we demonstrated the generality of our tools by generating knowledge rules for a new task, interpreting suburban house scenes, and using our compiled knowledge to analyze an image set [McKeown and Harvey 87].

In our work with SPAM, we have attempted to identify knowledge sources that do not suffer such drawbacks as dependency on objects' spectral properties or reliance on viewpoint-specific spatial relationships. We strive rather to exploit spatial relationships to generate a reasoning chain by applying multiple constraints across several interpretive levels. While spectral knowledge can play a role in certain domains, we assert that other types of spatial knowledge offer greater power for interpreting aerial imagery. In particular, we have found the following knowledge types generally available and effective:

- Knowledge for determining and defining appropriate scene domain primitives
- Knowledge of spatial relationships and constraints among the domain primitives
- Knowledge of model decompositions that determine collections of primitives that form "natural" scene components
- Knowledge of methods for combining scene components into complete scene interpretations
- Knowledge of how to recognize and evaluate conflicts among competing interpretations.

SPAM is also discussed in Chapter 3 of this report.

4.1.3. Computational properties of chunks

For several years now, we have been applying the notion of *chunking* to problem solving. Briefly, chunking divides a total problem configuration into smaller "chunks", each of which is easily recognizable and has properties useful in understanding the current situation. Our earliest work, in the chess domain, produced a program that operated on a subset of pawn endings and produced speedups of 10^{14} over previous strategies. This result encouraged us to extend the chunking approach to cover the full domain of pawn endings. Our present program, Chunker, can decompose any pawn ending into chunks.

The basic problem-solving technique employs selective search augmented by excellent knowledge. Strategic plans and chunk properties, up to a dozen properties per chunk, form Chunker's knowledge. Once the program identifies a chunk, it can look up the chunk's properties in a database, or in case of an unusual chunk, compute the properties dynamically. Once it has ascertained each chunk's property values, the program can operate across chunks to identify which side is ahead and what alternative plans each side has. Each chunk restricts what is possible and, out of this reduced search space, alternative plans come quite easily.

We have exerted considerable care to order plans according to potency and likelihood of success. It is quite possible that the most potent plan in a position can, given the various chunk properties, be pronounced to be a win for one side without any further investigation. In such cases the search immediately backtracks to other issues that have yet to be resolved. When the Chunker cannot statically determine a plan's outcome, the search proceeds until the plan either succeeds or is found lacking in some significant way. The solution to difficult problems cannot immediately be looked up, or found through only a moderate amount of combinatoric investigation. To solve problems such as this, it is necessary to have enough accurate domain knowledge to be able to rule out alternatives, that under a less stringent appraisal may appear to be plausible. Only in this way is it possible to search a meaningful subset of the many possible trees of alternatives. It is in this, that Chunker excels.

Chunker can now solve approximately 70% of the problems in the most comprehensive pawn-endings text. The book presents an authoritative treatise on all aspects of the subject, including those at the highest level. Most of its exercises challenge even the world's best players. The problems require showing that a proposed plan will win against any possible counterplan or demonstrating that there is no viable winning plan. Showing that a plan wins or fails to win against a particular counterplan generally involves a highly directed search. Chunker has found about 20 major errors in the book of 1600 problems. It has also found about 30 alternative solutions. One interesting aspect of these errors, is similar to what occurred in the original Chunker work: There is a particular maneuver which can be used to save certain types of positions which is apparently unknown or at least not well known. Chunker has found this maneuver in a number of problems. In any case, we consider Chunker to be just about equal to the highest human level of performance in this domain.

4.2. Machine learning

Learning—including acquiring new information, formulating new heuristics, adapting to new situations, and refining problem-solving skills—forms a major component of intelligent behavior. We are investigating aspects of machine learning that include experiential learning in problem solving and integrated systems that adapt to new environments. Much of the earlier research into machine learning takes the form of learning from examples in which the machine has no opportunity to manipulate the situation and to perform experiments to gain more knowledge. In the World Modelers and Prodigy projects, we are pursuing a more interactive approach. While expert systems have demonstrated their power and utility in many domains, manually acquiring the requisite knowledge from an expert and transforming it into formal rules typically remains a time-consuming and difficult process. Machine learning techniques offer a potential solution to the "knowledge bottleneck" by automating the knowledge acquisition process. In particular, by developing problem-solving systems that can learn by experience and instruction, we can enable human domain experts to transfer their knowledge in a natural and efficient manner. The World Modelers project is building a simulated environment in which we can examine the interactive aspects of learning. The Prodigy project also deals in interactive learning but focuses on the learning system itself.

4.2.1. Learning through experience

We are developing the Prodigy system), a "learning apprentice" intended to facilitate the acquisition of problem-solving expertise [Minton et al. 86]. Our ultimate objective is to create a unified architecture for building instructable expert systems in multiple domains, an architecture that enables learning from both expert instruction and direct experience.

Acquiring problem-solving expertise

The initial version of Prodigy's problem solving architecture was modeled after that of Strips, and much of the early part of 1985 was devoted to making changes and improvements on the Strips approach. These included adding a simple reason-maintenance system and enabling the problem solver to have direct control over the invocation of inference rules as well as operators. We adopted a variation of predicate calculus as the input language and revised to be more intelligible to human users of the system. In addition, we designed the problem solver its activities could be controlled by the addition of search control rules.

The Prodigy system comprises a general-purpose problem solver integrated with an explanation-based learning module (EBL) and a knowledge-refinement interface (KRI). The Strips paradigm offers a stable, tried-and-true base for our learning research. Prodigy's problem solver also incorporates a flexible, rule-guided control structure. Control rules provide a means for distinguishing control knowledge (when to work on a task) from domain knowledge (how to accomplish a task) and may encode either domain-specific or domain-independent search heuristics. Prodigy represents all its knowledge, including control rules and domain operators, using a uniform, logic-based description language.

Prodigy's EBL facility learns domain-specific control rules by analyzing problem-solving traces. While previous explanation-based learning systems could learn by analyzing problem solutions, EBL can also learn from additional experiences, including problem-solving failures and goal interferences. The KRI module enables a domain expert to instruct and interact with Prodigy. In the second half of 1986, we added a facility that allows Prodigy to learn from experimentation and handle situations where its domain operators have been incorrectly or incompletely specified.

New knowledge from current explanations

One challenge in developing systems that can learn from their own experience lies in how to capture and efficiently exploit relevant experience. Our strategy is to build a *general* explanation facility that is also closely tied to Prodigy's steps in solving a *particular* problem. This facility will enable Prodigy to explain its problem solutions so that both it and humans can use the information.

Earlier systems restricted their target concepts to the problem solver's actual goals. That is, they could transform a failure to "stack block 27 on cylinder 5" into new knowledge, but had no higher-level concept for "the stack operator failed at node 7" in the search tree. In the former case, the mapping from problem-solving trace to causal

explanation is fairly obvious, though limited in applicability. The latter case, however, requires an explicit means of mapping from trace to explanation proving some more general concept such as "this operator failed". We have designed an efficient "explanation-based specialization" (EBS) method that handles the more general case. Our initial EBS module links EBL and the problem solver, mapping from trace to explanation through explicit discrimination functions that indicate which axioms are appropriate.

Mechanically generated explanations, unfortunately, tend toward arcane verbosity, as researchers in automatic theorem-proving would agree. Humans have difficulty reading them and other programs have difficulty evaluating them. EBS is no different, so we designed a procedure to meet both clients' needs. Our "compression analysis" method embodies a post-processing strategy that rewrites learned control rules, increasing their readability and reducing their match cost. The compressor employs partial evaluation, truth-preserving logical transformations, and domain-specific simplification rules.

Merely identifying and formulating new knowledge cannot guarantee that it will contribute a cost-effective control rule. Quite possibly the time to match a rule with preconditions may exceed any savings in solution time. To address this issue and give Prodigy a *selective* learning capability, we designed a method that empirically analyzes control rule utility, as suggested in [Minton 85a]. Our metric compares a rule's average match cost to its average savings, adjusting for application frequency.

Finally, since the ultimate test is how well Prodigy solves a realistic problem, we created an experimental test domain. Our domain describes the expertise required for crafting a primary telescope mirror from raw materials and includes operators for grinding, aluminizing, and polishing. We produced an initial design for methods of monitoring a solution plan while it executes, dynamically replanning when reality diverges from expectation, and learning through experiment. These capabilities represent preliminary steps toward learning that can correct an incomplete or inaccurate world model.

4.2.2. Integrating learning in a reactive environment

The World Modelers Project explores machine learning within a simulated reactive environment that facilitates designing, implementing, and testing integrated learning systems [Carbonell and Hood 86]. Researchers can define autonomous "agents" whose "bodies" can move about the environment, performing simple actions such as pushing objects. An agent's "mind" resides in a program that learns how to satisfy its predefined needs and priorities. Our project goals include discovering learning techniques applicable to a wide range of real-world learning tasks such as planning and sensory-motor skill acquisition. We also seek methods for combining such learning techniques to form a complete, autonomous agent that can gradually acquire knowledge through experience and adapt to a changing environment.

Our work focuses on constructing agents that can survive within the environment for extended periods. That is, we strive to build complete cognitive systems that continue

acquiring new abilities without losing those they already have. We seek robust learning techniques that apply in a wide variety of situations and that remain insensitive to small environmental changes. An agent should be able to obtain food, for example, even if it knows only approximately where to look. Such techniques offer the greatest promise of successful transfer to real-world robots that must deal with environments about which they have incomplete knowledge. During the latter half of 1986, we extended our agents' cognitive architectures to incorporate learning-from-experiment mechanisms and built supporting software to help implement such agents.

As a first step toward building organisms capable of interesting learning behavior, we designed a cognitive architecture partitioned into ten basic components: the internal state generator, the object and event recognizers, the cognitive map constructor, the focal attention mechanism, the working memory, the long term memory, the scheduler, the planner, and the schema-learning mechanisms [Mozer and Gross 85]. We designed the architecture as a framework within which subsystems could be independently designed and implemented. During the first half of 1985, we completed designing and implementing one such subsystem, a schema-based event memory.

Extending agent architectures

To create a substrate for more sophisticated abilities, we developed a method of active learning through which an agent can exploit environmental feedback to refine its understanding of operator capabilities. This experimental learning—trying an action sequence and observing the result, in effect—permits an agent to begin "life" from a very basic specification and almost no knowledge of its world. Our method allows an agent to improve its knowledge by discovering which environmental features are relevant to selecting a particular operator. When its initially simple operator descriptions prove inadequate to identify an effective operator, the agent can enrich the descriptive templates. To develop a new template feature, the agent searches heuristically for a linear discriminator function that will distinguish cases where the operator succeeds from those where it fails unexpectedly.

Software for building agents

Each learning agent needs an interface between itself and the environment simulator—the "mind-body" connection. Such an interface provides mechanisms for transforming and filtering the available, potentially voluminous sensory data down to those few data that the agents' higher cognitive mechanisms can perceive directly. We designed and implemented a generalized sensor-effector interface that meets this need. Our design provides a set of shared utilities and offers sufficient versatility that it will reduce the code each investigator must write when building a new agent.

In another move to simplify building agents, we implemented a production system package that combines frame-like data structures with priority-ordered control rules. Our Rulekit package employs a fast, Rete-style match algorithm and, due to its inheritance capability, offers more powerful and flexible pattern-matching than standard OPS packages. Rulekit's conflict-resolving mechanisms, based on agendas, also yield greater flexibility and facilitate obtaining desirable agent behavior. We can, for example, assign invocation precedence to higher-priority rules. This strategy would allow intense

sensory data, such as a loud noise, to interrupt an agent's current activity and let the agent attend to a more urgent stimulus. Rulekit provides a general-purpose AI tool, simplifying the investigator's task by transparently handling such issues as match efficiency. Beyond the World Modelers domain, other projects in machine learning and expert systems have recognized Rulekit's value and currently use it.

Seeing what transpires within the simulated physical environment has proven crucial to debugging the simulator. We recently upgraded our monitoring capability by implementing a graphical interface on a Sun workstation. The interface allows investigators to monitor agents' behavior as they interact with other agents and objects. The new implementation exploits specialized color graphics hardware to achieve a hundredfold increase in drawing speed and a tenfold increase in resolution over our previous implementation. With it, we have identified and corrected numerous, previously undetected qualitative problems in collision resolution.

Finally, we shipped our world simulator to the UC-Irvine group, which will also use it to study learning within reactive environments. They, however, will investigate a different task set and will probably test the environment in areas where our group has not ventured. Our success in exporting the simulator augurs well for porting the system to other sites, too.

4.3. Massively parallel architectures

Humans apparently solve problems in a "knowledge-intensive" mode, applying small amounts of search when necessary. The human strategy is flexible and avoids the need to encode *all* knowledge. Many successful AI systems mimic the human style and expert systems offer the prime example. Competitive gaming systems typically employ the opposite scheme, relying primarily on search.

In the quest for higher performance, "more of the same" offers diminishing amounts of "better". Clearly, a more productive approach would evolve knowledge-intensive systems toward increased search or introduce intelligence into search-intensive systems. Our research offers an opportunity to study the effects of *extremely* fast—and relatively clever—searches in very large problem spaces. The opportunity here is significant because we have no experience with intelligent systems, human or mechanical, solving problems in this manner.

4.3.1. Parallel search and pattern recognition

Our work on parallel architectures has concentrated mainly on the Hitech chess machine, which achieves its success from parallelism in the right places. Hitech has now reached a National rating of 2359, making it approximately the 180th best US chess player.

Hitech's search algorithm must identify possible moves, determine whether a given position is legal, recognize positions seen before, and evaluate the candidate position,

among other things. To minimize elapsed time, Hitech performs these tasks in parallel. This strategy alone allows it to process approximately 175,000 positions/second, comparable to the fastest chess program on a 4XMP-Cray.

Parallelism is most crucial in the evaluation stage. At Hitech's search rate, it can spend, at most, one microsecond evaluating each position. During that interval, a super-fast, general-purpose machine could execute possibly 50 instructions, a number that could not go very far in evaluating a complex situation, even given the power of vector instructions. We have found that pattern recognition complements Hitech's powerful search extremely well, though other systems typically avoid it because either:

- On a serial machine, examining all potential patterns is simply too expensive.
- Where pattern-specific hardware is employed, adding a new pattern or changing an old one means building new hardware.

Hitech avoids these problems by having *programmable* pattern recognition hardware. At present there are 22 such units, each capable of recognizing patterns of limited complexity. Before Hitech begins a search, a software Oracle analyzes the root position and decides which patterns from its pattern library should be loaded into each unit. This provides both speed and flexibility, since the loading occurs only once per search. Since we incorporated the parallel pattern approach, Hitech's rating has climbed about 200 points, or one full rating category from high Expert to high Master.

4.3.2. SUPREM: A new search architecture

Out of our Hitech research has emerged a new search architecture, which we call SUPREM (Search Using Pattern Recognition as an Evaluation Mechanism). The system architecture has two parts:

- The Oracle is SUPREM's primary knowledge repository and has all knowledge the system needs to operate. Since the knowledge is domain-dependent, each domain requires a unique Oracle. After analyzing the root position, the Oracle selects which patterns should be loaded into which units, and directs the compilation and down-loading of these patterns.
- The Searcher executes the search, and evaluates the nodes of the search tree. This involves a move generator that can order the legal moves according to their potency so that likely best solutions are tried first. It also involves evaluation using the Pattern Recognizer units. These units retrieve values whenever the candidate state—as a pattern of state components — matches pre-tabulated patterns in the recognizer memory. The outputs of these recognizer units is summed to form the evaluation of a node. The tree of possibilities is evaluated in the usual way by backing up the values of leaf nodes to produce a more informed view of what the value of any action at the root really is.

During 1986, the pattern recognizing units were made more powerful, so as to allow global context— the most essential characteristics of the current state— to influence the

evaluation. For instance, the interest in King safety is very much dependent upon how much opponent material exists for attacking the King. As the amount of material lessens, so does the interest in protecting the King. In 1987, we have been developing software to take advantage of this new hardware. This is a difficult undertaking involving much tuning to determine just what the global state variables (which we have called Application Coefficients in earlier work) should look like, and how much influence each should have.

We have also been working on new search algorithms since it is becoming apparent that in order to play chess at the highest level, it will be necessary to search deeper than any chess machine presently can. To this end a new second generation hardware move generator has been built which is faster than the Hitech move generator, only requires a single chip, and can be paired with other identical chips to make a multiprocessor searching several parts of the tree at the same time. This chip with very rudimentary support was able to achieve an even score in the 1986 ACM North American Computer Chess Championship.

4.3.3. Boltzmann Networks

The Boltzmann Machine group, an interdisciplinary research team from the Computer Science and Psychology departments, is investigating a class of fine-grained, massively parallel computer architectures that may allow us to build a fast, general recognition engine. The machine can be trained by showing it examples of the desired input/output mappings and has some capacity for generalizing from the cases it has seen to similar cases. Boltzmann networks resemble neuron networks and may help us to understand how such operations are carried out by the human brain. In addition, Boltzmann networks are good candidates for wafer-scale VLSI technology because they employ a distributed representation that is inherently fault-tolerant.

We conducted a series of experiments on small-scale Boltzmann networks using simulators. At present, we have an interesting mathematical result that guarantees a certain learning procedure will build internal representations that allow the connection strengths to capture the underlying constraints implicit in a large ensemble of examples taken from a domain. We also have simulations that show that the theory works for some simple cases, but the current version of the learning algorithm is very slow [Ackley et al. 85].

In an attempt to speed up the learning, we are investigating variations on the Boltzmann learning algorithms. We are focusing on back-propagation, a technique that was discovered by Hinton, along with Rumelhart and Williams of UCSD. Back-propagation learns from examples without the costly "simulated annealing" searches of the Boltzmann architecture. Similar techniques were considered years ago, but were rejected because the learning process could get stuck. Fortunately, we have found, based on simulations, that this seldom happens in practice, and for some kinds of problems, back-propagation is one or two orders of magnitude faster than the pure Boltzmann architecture [Hinton et al. 86].

Boltzmann and back-propagation networks are naturally suited for recognition tasks because the networks are trained to produce some particular response for a given class of inputs. But, to broaden the architecture's application to more than just recognition tasks, Touretzky has been studying the problem of building more conventional symbol-processing AI architectures on a connectionist substrate [Touretzky and Hinton 85].

4.4. Rational control of reasoning

On the more theoretical side of Machine Intelligence, Doyle has been investigating issues related to rational control of reasoning. Rational control of reasoning aims at increasing the power, flexibility, and reliability of knowledge-based systems. Current techniques are relatively unreliable and inflexible, since they may fail on one problem even though they succeed on closely related ones, and since excessive effort is often necessary to revise them to correct such failures. Rationally controlled reasoning reduces unreliability, for the hallmarks of rationality are flexibility and comprehensiveness, taking everything into account. Doyle's research uses tools from modern logic, decision theory, and the theory of algorithms to develop formal specifications and designs for agents that rationally and deliberately control their own reasoning and organization as well as their external actions.

Doyle first explains each of the central topics in AI in terms of rationally planned and conducted revisions of the agent's attitudes. Rationally adopting and revising beliefs and probabilities forms the basis of reasoning, learning, and reason maintenance. Rationally adopting and revising goals and preferences forms the basis of problem solving, search, and decision making. Rationally adopting and revising intentions and priorities forms the basis of planning. Further, most issues concerning meta-reasoning, reflection, and control of reasoning are more clearly described and evaluated as aspects of rationally planned reasoning.

Doyle then applies theories from logic and decision theory to formalize the special sorts of decisions that arise in controlling one's own reasoning. These formalizations connect the common non-numerical AI techniques with the common numerical statistical techniques in a theoretically rigorous way. This permits ready connections between ideas and techniques in AI and ideas and techniques in logic, statistics, decision theory, economics, and operations research, facilitating transportation of good ideas and techniques into and out of AI.

During the first half of 1987, Doyle completed a monograph on rational control, and began circulating the draft in July 1987. The monograph, titled *Artificial Intelligence and Rational Self-Government*, is an initial presentation of the rational view of artificial intelligence theories. One specific accomplishment is an application of Dana Scott's theory of information systems to describing the internal logic of the agent's states. This theory addresses some sort of unreliability through notions of constitutive logics and constitutive intentions, which are limited logics and self-specifications that the agent automatically respects without special control. Such abstract logics permit formal presentations of AI architectures that are just as rigorous as formal theories of programming language semantics.

Another accomplishment is a formal theory of decision-making under incomplete and inconsistent beliefs and preferences. This theory is based on qualitative comparisons of relative likelihood and preferability, and consistent selections from inconsistent sets. Expressing rational control knowledge qualitatively instead of in terms of inconvenient numerical representations enhances the flexibility with which the agent's beliefs may be modified, either by itself or by its informants during knowledge acquisition. As an added attraction, one special case of this theory is formally identical to the standard theory of group decision-making and public choice. This means that ideas about decision-making with conflicting preferences developed in the study of political, social, and business organizations may be readily transformed into techniques with which artificial agents might overcome inconsistencies in their knowledge.

4.5. Bibliography

[Ackley et al. 85] Ackley, D.H., G.E. Hinton, and T.J. Sejnowski.
A learning algorithm for Boltzmann machines.
Cognitive Science 9(1), January-March, 1985.

The computational power of massively parallel networks of simple processing elements resides in the communication bandwith provided by the hardware connections between elements. These connections can allow a significant fraction of the knowledge of the system to be applied to an instance of a problem in a very short time. One kind of computation for which massively parallel networks appear to be well suited is large constraint satisfaction searches, but to use the connections efficiently two conditions must be met: first, a search technique that is suitable for parallel networks must be found; second, there must be some way of choosing internal representations which allow the preexisting hardware connections to be used efficiently for encoding the constraints in the domain being searched. We describe a general parallel search method, based on statistical mechanics, and we show how it leads to a general learning rule for modifying the connection strengths so as to incorporate knowledge about a task domain in an efficient way. We describe some simple examples in which the learning algorithm creates internal representations that are demonstrably the most efficient way of using the preexisting connectivity structure.

[Berliner and Ebeling 86]
Berliner, H. and C. Ebeling.
The SUPREM architecture: A new intelligent paradigm.
Artificial Intelligence 28:3-8, 1986.

[Bisiani 87] Bisiani, R.
A software and hardware environment for developing AI applications on parallel processors.
In *Proceedings of the 5th National Conference on AI*, AAAI, August, 1987.

This paper describes and reports on the use of an environment, called Agora, that supports the construction of large, computationally expensive and loosely-structured systems, e.g. knowledge-based systems for speech and vision understanding. Agora can be customized to support the programming model that is more suitable for a given application. Agora has been designed explicitly to support multiple languages and highly parallel computations. Systems built with Agora can be executed on a number of general purpose and custom multiprocessor architectures.

[Carbonell 85] Carbonell, J.G.
Derivational analogy: A theory of reconstructive problem solving and expertise acquisition.
Technical Report CMU-CS-85-115, Carnegie Mellon University Computer Science Department,
March, 1985.
Derivational analogy, a method of solving problems based on the transfer of past experience to new problem situations, is discussed in the context of other general approaches to problem solving. The experience transfer process consists of recreating lines of reasoning, including decision sequences and accompanying justifications, that proved effective in solving particular problems requiring similar initial analysis. The role of derivational analogy in case-based reasoning and in automated expertise acquisition is discussed.

[Carbonell and Hood 85]
Carbonell, J.G. and G. Hood.
The World Modelers project: Objectives and simulator architecture.
In *Proceedings of the Third International Machine Learning Workshop*, June, 1985.
Machine learning has long sought to construct complete, autonomous learning systems that start with general inference rules and learning techniques, and gradually acquire complex skills and knowledge through continuous interaction with an information-rich external environment. The World Modelers project provides a simplified artificial environment—a continuous three-dimensional physical model of the world—to facilitate the design, implementation, and testing of integrated learning systems. This paper presents the rationale for building the simulator, and briefly describes its capabilities and the system architecture underlying its implementation.

[Carbonell and Hood 86]
Carbonell, J.G. and G. Hood.
The World Modelers project: Learning in a reactive environment,
In Mitchell, T.M., J.G. Carbonell, and R.S. Michalski, *Machine Learning: A Guide to Current Research*, Pages 29-34. Kluwer Academic Press, 1986.

[Doyle 85a] Doyle, J.
Circumscription and implicit definability.
Journal of Automated Reasoning 1:391-405, 1985.
We explore some connections between the technique of circumscription in artificial intelligence and the notion of implicit definition in mathematical logic. Implicit definition can be taken as the informal intent, but not necessarily the formal result, of circumscription. This raises some questions

for logical theory and suggests some implications for artificial intelligence practice. The principal implication is that when circumscription 'works' its conclusions can be explicitly described.

[Doyle 85b] Doyle, J.
Expert systems and the 'myth' of symbolic reasoning.
IEEE Transactions on Software Engineering SE-11(11), November, 1985.
Elements of the artificial intelligence approach to expert systems offer great productivity advantages over traditional approaches to application systems development, even though the end result may be a program employing AI techniques. These productivity advantages are the hidden truths behind the 'myth' that symbolic reasoning programs are better than ordinary ones.

[Doyle 85c] Doyle, J.
Reasoned assumptions and pareto optimality.
Technical Report CMU-CS-85-121, Carnegie Mellon University Computer Science Department,
December, 1985.
Default and non-monotonic inference rules are not really epistemological statements, but are instead desires or preferences of the agent about the makeup of its own mental state (epistemic or otherwise). The fundamental relation in non-monotonic logic is not so much self-knowledge as self-choice of self-determination, and the fundamental justification of the interpretations and structures involved come from decision theory rather than from logic and epistemology.

[Goetsch 86] Goetsch, G.
Consensus: A statistical learning procedure in a connectionist network.
Technical Report CMU-CS-86-131, Carnegie Mellon University Computer Science Department,
May, 1986.
We present a new scheme for the activity of neuron-like elements in a connectionist network. The Consensus scheme is based on statistical inference. The guiding principle of Consensus is that decisions should be deferred until sufficient evidence accumulates to make an informed choice. Consequently, large changes in network structure can be made with confidence. Nodes have an awareness of their role and utility in the network which allows them to increase their effectiveness. The reinforcement scheme utilizes the notion of confidence so that only nodes proven to contribute successfully issue reinforcements. Nodes are grouped into communities to exploit their collective

knowledge which exceeds any individual member. The network was tested against several problems and was able to find suitable encodings to solve them.

[Gupta et al. 86] Gupta, A., C. Forgy, A. Newell, and R. Wedig.
Parallel algorithms and architectures for rule-based systems.
In *Thirteenth Annual International Symposium on Computer Architecture*, IEEE, June, 1986.

Rule-based systems, on the surface, appear to be capable of exploiting large amounts of parallelism-it is possible to match each rule to the data memory in parallel. In practice, however, we show that the speed-up from parallelism is quite limited, less than 10-fold. The reasons for the small speed-up are: (1) the small number of rules relevant to each change to data memory; (2) the large variation in the processing required by the relevant rules; and (3) the small number of changes made to data memory between synchronization steps. Furthermore, we observe that to obtain this limited factor of 10-fold speed-up, it is necessary to exploit parallelism at a very fine granularity. We propose that a suitable architecture to exploit such fine-grain parallelism is a bus-based shared-memory multiprocessor with 32-64 processors. Using such a multiprocessor (with individual processors working at 2 MIPS), it is possible to obtain execution speeds of about 3800 rule-firings/sec. This speed is significantly higher than that obtained by other proposed parallel implementations of rule-based systems.

[Gupta et al. 87] Gupta, A., C.L. Forgy, D. Kalp, A. Newell, and M. Tambe.
Results of parallel implementation of OPS5 on the Encore multiprocessor.
Technical Report CMU-CS-87-146, Carnegie Mellon University Computer Science Department,
August, 1987.
Anoop Gupta is now a member of the Computer Science Department, Stanford University.

Until now, most results reported for parallelism in production systems (rule-based systems) have been simulation results—very few real parallel implementations exist. In this paper, we present results from our parallel implementation of OPS5 on an Encore multiprocessor with 16 CPUs. The implementation exploits very fine-grained parallelism to achieve significant speed-up. Our implementation is distinct from other parallel implementations in that we attempt to parallelize a highly optimized C-based implementation of OPS5. This is in contrast to other efforts where slow lisp-based implementations are being parallelized. The paper discusses both the overall structure and the low-level

issues involved in the parallel implementation and presents the performance numbers that we have obtained.

[Hinton and Lang 85]
Hinton, G.E. and K.J. Lang.
Shape recognition and illusory conjunction.
In *Proceedings of the International Joint Conference on Artificial Intelligence-85*, Pages 252-259. IJCAI, 1985.

One way to achieve viewpoint-invariant shape recognition is to impose a canonical, object-based frame of reference on a shape and to describe the positions, sizes and orientations of the shape's features relative to the imposed frame. This computation can be implemented in a parallel network of neuron-like processors, but the network has a tendency to make errors of a peculiar kind: When presented with several shapes it sometimes perceives one shape in the position of another. The parameters can be carefully tuned to avoid these 'illusory conjunctions' in normal circumstances, but they reappear is the visual input is replaced by a random mask before the network has settled down. Treisman and Schmidt (1982) have shown that people make similar errors.

[Hinton et al. 86] Hinton, G.E., J.M. McClelland, and D.E. Rumelhart.
Distributed representations,
In Rumelhart, D.E. and J.L. McClelland, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Bradford Books/MIT Press, 1986.

Every representational scheme has its good and bad points. Distributed representations are no exception. Some desirable properties arise very naturally from the use of patterns of activity as representations. Other properties, like the ability to temporarily store a large set of arbitrary associations, are much harder to achieve. As we shall see, the best psychological evidence for distributed representations is the degree to which their strengths and weaknesses match those of the human mind.

The first section of this chapter stresses some of the virtues of distributed representations. The second section considers the efficiency of distributed representations, and show clearly why distributed representations can be better than local ones for certain classes of problems. A final section discusses some difficult issues which are often avoided by advocates of distributed representations, such as the representation of constituent structure and the sequential focusing of processing effort on different aspects of a structured object.

[Hood 85] Hood, G.
Neural modeling as one approach to machine learning.
In *Proceedings of the Third International Machine Learning Workshop*, June, 1985.
In this paper I propose that a neural modeling approach is reasonable for investigating certain low-level learning processes such as are exhibited by invertebrates. These include habitation, sensitization, classical conditioning, and operant conditioning. Recent work in invertebrate neurophysiology has begun to provide much knowledge about the underlying mechanisms of learning in these animals. Guided by these findings, I am constructing simulated organisms which will display these basic forms of learning.

[Iwasaki 87] Iwasaki, Y.
Generating behavior equations from explicit representation of mechanisms.
Technical Report CMU-CS-87-131, Carnegie Mellon University Computer Science Department,
June, 1987.
The methods of causal ordering and comparative statics provide an operational means to determine the casual relations among the variables and mechanisms that describe a device, and to assess the qualitative effects of a given disturbance to the system. However, for correct application of the method of causal ordering, the equations comprising the model of the device must be such that each of them stands for a conceptually distinct mechanism. In this paper, we discuss the issue of building a model that meets this requirement and present our solution. The approach we have taken for building device models in our domain of a power plant is to represent explicitly one's understanding of mechanisms underlying an equation model as flows of matter and energy. A system was implemented to generate structural equations automatically from this representation. We discuss the results and some of the problems encountered along the way.

[Iwasaki and Simon 85]
Iwasaki, Y. and H. Simon.
Causality in device behavior.
Technical Report CMU-CS-85-118, Carnegie Mellon University Computer Science Department,
March, 1985.
This paper shows how formal characterizations of causality and of the method of comparative statics, long used in economics, thermodynamics and other domains, can be applied to clarify and make rigorous the qualitative causal

calculus recently proposed by de Kleer and Brown (1984). The formalization shows exactly what assumptions are required to carry out causal analysis of a system of interdependent variables in equilibrium and to propagate disturbances through such a system.

[Iwasaki and Simon 86]

Iwasaki, Y. and H. Simon.
Theories of causal ordering: Reply to de Kleer and Brown.
Technical Report CMU-CS-86-104, Carnegie Mellon University Computer Science Department,
January, 1986.

In their reply to our paper, Causality in Device Behavior, de Kleer and Brown seek to establish a clear product differentiation between the well-known concepts of causal ordering and comparative statics, on the one side, and their mythical causality and qualitative physics, on the other. Most of the differences they see, however, are invisible to our eyes. Contrary to their claim, the earlier notion of causality, quite as much as the later one, is qualitative and derives the relationship between the equations and their underlying components which comprise the modeled system. The concepts of causal ordering and comparative statics offer the advantage of a formal foundation that makes clear exactly what is being postulated. Hence, they can contribute a great deal to the clarification of the causal approaches to system analysis that de Kleer and Brown are seeking to develop.

In this brief response to the Comments, we discuss the source of the structural equations of the causal ordering approach, and we challenge more generally the claim that there are inherent differences (e.g. in the case of feedback) between the engineer's and the economist's approach to the study of system behavior.

[Kahn and McDermott 85]

Kahn, G. and J. McDermott.
MUD: A drilling fluids consultant.
Technical Report CMU-CS-85-116, Carnegie Mellon University Computer Science Department,
March, 1985.

This paper reports on MUD, a drilling fluids consultant developed at Carnegie Mellon University. MUD is able to diagnose fluid problems and recommend treatments for their correction. MUD's functionality, its approach to diagnosis, and its treatment strategies are discussed. In addition, we examine why MUD's approach to diagnosis is successful given domain constraints, and draw several conclusions with respect to knowledge acquisition strategies.

[Laird 85] Laird, J.E.
Soar 4.0 user's manual
1985.

The Soar software is available for non-commercial research purposes and it may be copied only for that use. Any questions concerning the use of Soar should be directed to John E. Laird at the address below. This software is made available as is and Xerox Corporation makes no warranty about the software, its performance, or the accuracy of this manual describing the software. All aspects of Soar are subject to change in future releases.

[Laird et al. 85a] Laird, J.E., P.S. Rosenbloom, and A. Newell.
Chunking in Soar: the anatomy of a general learning mechanism.
Technical Report CMU-CS-85-154, Carnegie Mellon University Computer Science Department,
August, 1985.

The goal of the *Soar* project is to build a system capable of general intelligent behavior. We seek to understand what mechanisms are necessary for intelligent behavior, whether they are adequate for a wide range of tasks - including search-intensive tasks, knowledge-intensive tasks, and algorithmic tasks - and how they work together to form a general cognitive architecture. One necessary component of such an architecture, and the one on which we focus in this paper, is a general learning mechanism. A general learning mechanism would possess the following properties. *Task generality.* It can improve the system's performance on all of the tasks in the domains. *Knowledge generality.* It can base its improvements on any knowledge available about the domain. *Aspect generality.* It can improve all aspects of the system. Otherwise there would be a *wandering-bottleneck problem*in which those aspects not open to improvement would come to dominate the overall performance effort of the system. *Transfer of learning.* What is learned in one situation will be used in other situations to improve performance. It is through the transfer of learned material that *generalization*, as it is usually studied in artificial intelligence, reveals itself in a learning problem solver.

There are many possible organizations for a general learning mechanism, each with different behavior and implications. The one adopted in **Soar** is the *simple experience learner.* There is a single learning mechanism that bases its modifications on the experience of the problem solver. The learning mechanism is fixed, and does not perform any complex problem solving.

[Laird et al. 85b] Laird, J., P. Rosenbloom, A. Newell, J. McDermott, and E. Orciuch. *Two Soar studies.*
Technical Report CMU-CS-85-110, Carnegie Mellon University Computer Science Department,
January, 1985.

The first paper is titled *Towards Chunking* as a *General Learning Mechanism* (Laird, Rosenbloom, & Newell, 1984). Chunks have long been proposed as a basic organizational unit for human memory. More recently chunks have been used to model human learning on simple perceptual-motor skills. In this paper we describe recent progress in extending chunking to be a general learning mechanism by implementing it within Soar. By implementing chunking within a general-problem solving architecture we take significant steps toward a general problem solver that can learn about all aspects of its behavior. We demonstrate chunking in Soar on three tasks: the Eight Puzzle, Tic-Tac-Toe, and a part of the R1 computer-configuration task. Not only is there improvement with practice but chunking also produces significant transfer of learned behavior, and strategy acquisition.

The second paper, titled *R1-Soar: An Experiment in Knowledge-Intensive Programming in a Problem-Solving Architecture* (Rosenbloom, Laird, McDermott Newell, & Orciuch, 1984), presents an experiment in knowledge-intensive programming in Soar. In Soar, knowledge is encoded within a set of problem spaces, yielding a system capable of reasoning from first principles. Expertise consists of additional rules that guide complex problem-space searches and substitute for expensive problem-space operators. The resulting system uses both knowledge and search when relevant. Expertise knowledge is acquired either by having it programmed, or by a chunking mechanism that automatically learns new rules reflecting the results implicit in the knowledge of the problem spaces. The approach is demonstrated on the computer-system configuration task, the task performed by the expert system, R1.

[Lehr 86] Lehr, T.F.
The implementation of a production system machine.
In *Proceedings of the Nineteenth Annual Hawaii International Conference on System Sciences*, University of Hawaii, January, 1986.
Also available as Technical Report CMU-CS-85-126.

The increasing use of production systems has drawn attention to their performance drawbacks. This paper discusses the architecture and implementation of a uniprocessor OPS production system machine. A brief tutorial on the OPS

production system and its Rete algorithm introduces salient issues that temper the selection of a uniprocessor architecture and implementation. It is argued that general features of Reduced Instruction Set Computer (RISC) architectures favorably address these issues. The architecture and a RTL description is presented for a pipelined RISC processor designed specifically to execute OPS. The processor has a static branch prediction strategy, a large register file and separate instruction and data fetch units.

[McKeown and Harvey 87]
McKeown, D.M. Jr. and W.A. Harvey.
Automating knowledge acquisition for aerial image interpretation.
Technical Report CMU-CS-87-102, Carnegie Mellon University Computer Science Department,
January, 1987.
The interpretation of aerial photographs requires a lot of knowledge about the scene under consideration. Knowledge about the type of scene: airport, suburban housing development, urban city, aids in low-level and intermediate level image analysis, and will drive high-level interpretation by constraining search for plausible consistent scene models. Collecting and representing large knowledge bases requires specialized tools. In this paper we describe the organization of a set of tools for interactive knowledge acquisition of scene primitives and spatial constraints for interpretation of aerial imagery. These tools include a user interface for interactive knowledge acquisition, the automated compilation of that knowledge from a schema-based representation into productions that are directly executable by our interpretation system, and a performance analysis tool that generates a critique of the final interpretation. Finally, the generality of these tools is demonstrated by the generation of rules for a new task, suburban house scenes, and the analysis of a set of imagery by our interpretation system.

[Minton 85a]
Minton, S.N.
Selectively generalizing plans for problem solving.
In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, August, 1985.
Problem solving programs that generalize and save plans in order to improve their subsequent performance inevitably face the danger of being overwhelmed by an ever-increasing number of stored plans. To cope with this problem, methods must be developed for selectively learning only the most valuable aspects of a new plan. This paper describes MORRIS, a heuristic problem solver that measures the utility of plan fragments to determine

whether they are worth learning. MORRIS generalizes and saves plan fragments if they are frequently used, or if they are helpful in solving difficult subproblems. Experiments are described comparing the performance of MORRIS to a less selective learning system.

[Minton 85b] Minton, S.N.
A game-playing program that learns by analyzing examples.
Technical Report CMU-CS-85-130, Carnegie Mellon University Computer Science Department,
May, 1985.
This paper describes a game-playing program that learns tactical combinations. The program, after losing a game, examines the opponent's moves in order to identify how the opponent forced the win. By analyzing why this sequence of moves won the game, a generalized description of the winning combination can be produced. The combination can then be used by the program in later games to force a win or to block an opponent's threat. This technique is applicable for a wide class of games including tic-tac-toe, gomoku and chess.

[Minton 85c] Minton, S.N.
Overview of the Prodigy learning apprentice.
In *Proceedings of the Third International Machine Learning Workshop*, June, 1985.
This paper briefly describes the Prodigy system, a learning apprentice for robot construction tasks currently being developed at Carnegie Mellon University. After solving a problem, Prodigy re-examines the search tree and analyzes its mistakes. By doing so, Prodigy can often find efficient tests for determining if a problem solving method is applicable. If adequate performance cannot be achieved through analysis alone, Prodigy can initiate a focused dialogue with a teacher to learn the circumstances under which a problem solving method is appropriate.

[Minton et al. 86] Minton, S.N., J.G. Carbonell, C.A. Knoblock, D. Kuokka, and H. Nordin.
Improving the effectiveness of explanation-based learning.
In *Proceedings of the Workshop on Knowledge Compilation*, September, 1986.
In order to solve problems more effectively with accumulating experience, a system must be able to extract, analyze, represent and exploit search control knowledge. While previous research has demonstrated that explanation-based learning is a viable method for acquiring search control knowledge, in practice explanation-based techniques may generate complex expressions that are computationally expensive to use. Better results may be obtained by

explicitly reformulating the learned knowledge to maximize its effectiveness. This paper reports on the PRODIGY learning apprentice, an instructable, general-purpose problem solver that combines *compression analysis* with explanation-based learning, in order to formulate useful search control rules that satisfy the dual goals of generality and simplicity.

[Mozer and Gross 85]
Mozer, M.C. and K.P. Gross.
An architecture for experiential learning.
In *Proceedings of the Third International Machine Learning Workshop*, June, 1985.
This paper describes a cognitive architecture for an intelligent organism residing in the World Modelers environment. The architecture is partitioned into ten basic components: the internal state generator, the object and event recognizers, the cognitive map constructor, the focal attention mechanism, the working memory, the long term memory, the goal scheduler, the planner, and the schema-learning mechanisms. A uniform procedural representation is necessary for interactions among the components.

[Rappaport 85] Rappaport, A.
Goal-free learning by analogy.
In *Proceedings of the Third International Machine Learning Workshop*, June, 1985.
The purpose of this research is to propose and study mechanisms for incremental learning by goal-free learning by analogy in an information-rich world. A similarity matrix is obtained on which a clustering analysis is performed. The abstractions obtained are transformed into a plan of action which may be considered an imitation of previously observed behavior. While the agent has no explicit idea of the original goals, it acquires a subjective knowledge by an *a posteriori* identification of goals. We discuss such mechanisms for the building of a concept-based behavior and the goal-free acquisition of knowledge on which knowledge-intensive learning methodologies can then be applied.

[Rosenbloom and Laird 86]
Rosenbloom, P.S. and J.E. Laird.
Mapping explanation-based generalization onto Soar.
In *Proceedings AAAI-86: 5th National Conference on Artificial Intelligence*, AAAI, August, 1986.
Explanation-based generalization (EBG) is a powerful approach to concept formation in which a justifiable concept definition is acquired from a single training example and an underlying theory of how the example is an instance of the

concept. Soar is an attempt to build a general cognitive architecture combining general learning, problem solving, and memory capabilities. It includes an independently developed learning mechanism, called chunking, that is similar to but not the same as explanation-based generalization. In this article we clarify the relationship between the explanation-based generalization framework and the Soar/chunking combination by showing how the EBG framework maps onto Soar, how several EBG concept-formation tasks are implemented in Soar, and how the Soar approach suggests answers to four of the outstanding issues in explanation-based generalization.

[Rosenbloom and Newell 85]

Rosenbloom, P.S. and A. Newell.
The chunking of goal hierarchies: A generalized model of practice,
In Michalski, R.S., J.G. Carbonell, and T.M. Mitchell, *Machine Learning: An Artificial Intelligence Approach, Volume II.* Morgan Kaufmann Publishers, Inc.: Los Altos, CA, 1985.

This chapter describes recent advances in the specification and implementation of a model of practice. In previous work the authors showed that there is a ubiquitous regularity underlying human practice, referred to as the *power law of practice*. They also developed an abstract law of practice, called the *chunking theory of learning*. This previous work established the feasibility of the chunking theory for a single 1023-choice-reaction-time task, but the implementation was specific to that one task. In the current work a modified formulation of the chunking theory is developed that allows a more general implementation. In this formulation, task algorithms are expressed in terms of hierarchical goal structures. These algorithms are simulated within a goal-based production-system architecture designed for this purpose. *Chunking* occurs during task performance in terms of the parameters and results of the goals experienced. It improves the performance of the system by gradually reducing the need to decompose goals into their subgoals. This model has been successfully applied to the task employed in the previous work and to a set of stimulus-response capability tasks.

[Rosenbloom et al. 85]
Rosenbloom, P.S., J.E. Laird, J. McDermott, A. Newell, and E. Orciuch.
R1-Soar: An experiment in knowledge-intensive programming in a problem-solving architecture.
In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, 1985.
Also available in *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*, Denver, 1984, and as part of Technical Report CMU-CS-85-110.

[Saito and Tomita 86]
Saito, H. and M. Tomita.
On automatic composition of stereotypic documents in foreign languages.
Technical Report CMU-CS-86-107, Carnegie Mellon University Computer Science Department,
December, 1986.
This paper describes an interactive system that composes high quality stereotypic documents. The language for the interaction is totally independent from the target language in which the documents are written; that is, a user can produce documents in a foreign language by interacting with the system in his language without any knowledge of the foreign language. It is also possible to produce documents in several languages simultaneously. The idea is that the system first builds, by interaction, a semantic content which contains enough information to produce the documents. Then the system composes the document by looking at the specification file, which specifies the stereotypic document of a particular language. A new type of document or a new target language can be added to the system by simply creating a new specification file without altering the program itself. A successful pilot system has been implemented at the Computer Science Department, Carnegie Mellon University.

[Saxe 85]
Saxe, J.B.
Decomposable searching problems and circuit optimization by retiming: Two studies in general transformations of computational structures.
Technical Report CMU-CS-85-162, Carnegie Mellon University Computer Science Department,
August, 1985.
An important activity in the advancement of knowledge is the search for general methods: techniques applicable to large classes of problems. This dissertation studies general *transformations* of computational structures in two domains (1) design of data structures for *decomposable searching*

problems and (2) optimization of *synchronous digital circuits.*

[Shen 87] Shen, W.
Functional transformations in AI discovery systems.
Technical Report CMU-CS-87-117, Carnegie Mellon University Computer Science Department,
April, 1987.

The power of scientific discovery systems derives from two main sources: a set of heuristics that determine *when* to apply a creative operator (an operator for forming new operators and concepts) in a space that is being explored; and a set of creative operators that determine *what* new operators and concepts will be created for that exploration. This paper is mainly concerned with the second issue. A mechanism called *functional transformations* (FT) shows promising power in creating new and useful creative operators during exploration. The paper discusses the definition, creation, and application of functional transformations, and describes how the system ARE, starting with a small set of creative operations and a small set of heuristics, uses FT's to create all the concepts attained by Lenat's AM system and others as well. Besides showing a way to meet the criticisms of lack of parsimony that have been leveled against AM, ARE provides a route to discovery systems that are capable of "refreshing" themselves indefinitely by continually creating new operators.

[Stern and Lasry 85]
Stern, R.M. and M.J. Lasry.
Dynamic speaker adaptation for feature-based isolated word recognition.
In *IEEE Transactions on Acoustics, Speech, and Signal Processing,*
IEEE, May, 1985.

In this paper we describe efforts to improve the performance of Feature, the Carnegie Mellon University speaker-independent speech recognition system that classifies isolated letters of the English alphabet, by enabling the system to learn the acoustical characteristics of individual speakers. Even when features are designed to be speaker-independent, it is frequently observed that feature values may vary more from speaker to speaker than from letter to letter. In these cases it is necessary to adjust the system's statistical description of the features of individual speakers to obtain optimum recognition performance. This paper describes a set of dynamic adaptation procedures for updating expected feature values during recognition. The algorithm uses maximum *a posteriori* probability (MAP) estimation techniques to update the mean vectors of sets of

feature values on a speaker-by-speaker basis. The MAP estimation algorithm makes optimal use of both knowledge of the observations input to the system from an individual speaker, and the relative variability of the features' mean vectors across the various letters enables the system to adapt its representation of similar sounding letters after any one of them is presented to the classifier. The use of dynamic speaker adaptation improves classification performance of Feature by 49% after four presentations of the alphabet, when the system is provided with *a posteriori* knowledge of which specific utterance had been presented to the classifier from a particular user. Performance can be improved by as much as 31% when the system is allowed to adapt passively, without any information from individual users.

[Touretzky 86a] Touretzky, D.S.
BoltzCONS: Reconciling connectionism with the recursive nature of stacks and trees.
In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Cognitive Science Society, August, 1986.

Stacks and trees are implemented as distributed activity patterns in a simulated neural network called BoltzCONS. The BoltzCONS architecture employs three ideas from connectionist symbol processing—coarse coded distributed memories, pullout networks, and variable binding spaces, that first appeared together in Touretzky and Hinton's neural net production system interpreter. In BoltzCONS, a distributed memory is used to store triples of symbols that encode cons cells, the building blocks of linked lists. Stacks and trees can then be represented as list structures. A pullout network and several variable binding spaces provide the machinery for associative retrieval of cons cells, which is central to BoltzCONS' operation. Retrieval is performed via the Boltzmann Machine simulated annealing algorithm, with Hopfield's energy measure serving to assess the results. The network's ability to recognize shallow energy minima as failed retrievals makes it possible to traverse binary trees of unbounded depth without maintaining a control stack. The implications of this work for cognitive science and connectionism are discussed.

[Touretzky 86b] Touretzky, D.S.
Representing and transforming recursive objects in a neural network, or "Trees *do* grow on Boltzmann machines".
In *Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics*, IEEE, October, 1986.

BoltzCONS is a neural network that manipulates symbolic data

structures. The name reflects the system's mixed representational levels: it is a Boltzmann Machine in which Lisp cons cell-like structures appear as an emergent property of a massively parallel distributed representation. BoltzCONS is controlled by an attached neural network production system interpreter also implemented as a Boltzmann Machine. Gated connections allow the production system and BoltzCONS to pass symbols back and forth. A toy example is presented where BoltzCONS stores a parse tree and the production system contains a set of rules for transforming parse trees from active to passive voice. The significant features of BoltzCONS are its ability to represent structured objects and its generative capacity, which allows it to create new symbol structures on the fly.

[Touretzky and Hinton 85]
Touretzky, D.S. and G.E. Hinton.
Symbols among the neurons: Details of a connectionist inference architecture.
In *Proceedings of the International Joint Conference on Artificial Intelligence-85*, Pages 238-243. IJCAI, 1985.
Pattern matching and variable binding are easily implemented in conventional computer architectures. In a distributed neural network architecture each symbol is represented by activity in many units and each unit contributes to the representation of many symbols. Manipulating symbols using this type of distributed representation is not as easy as with a local representation where each unit denotes one symbol, but there is evidence that the distributed approach is the one chosen by nature. We describe a working implementation of a production system interpreter in a neural network using distributed representations for both symbols and rules. The research provides a detailed account of two important symbolic reasoning operations, pattern matching and variable binding, as emergent properties of collections of neuron-like elements. The success of our production system implementation goes some way towards answering a common criticism of connectionist theories: that they aren't powerful enough to do symbolic reasoning.

[vandeBrug et al. 85]
van de Brug, A., J. Bachant, and J. McDermott.
Doing R1 with style.
In *Proceedings of the Second Conference on Artificial Intelligence Applications*, IEEE, 1985.
A premise of this paper is that much of an expert system's power is due to the strong constraints on the way its knowledge can be used. But the knowledge that an expert system has is seldom explicated in terms of uses, nor does

there seem to be much interest in identifying the source of the usage constraints. The work reported in this paper explores the relationship between a problem-solving method and the various roles knowledge plays in a computer system configuration task. The results suggest that the knowledge in a system like R1 can be represented more coherently if the problem-solving method is exploited to explicitly define the various knowledge roles.

5. PROGRAMMING TECHNOLOGY

Programming technology comprises both the principles and knowledge (the know-how) and the tools (primarily software systems) used to produce software—compilers, debuggers, editors, design systems, etc. The high cost of producing software creates a need for increasingly sophisticated environments and tools that a programmer can use to develop and maintain software.

The Gandalf project explores two key issues in improving programming technology: What kinds of expert knowledge about system building can we incorporate into a programming environment, and in what respects can we make system development a cooperative effort between the environment and the user? Our primary strategy is to create effective methods and systems for evolving software development environments automatically and intelligently. We divide our efforts between building tools for generating environments and testing our ideas and tools by applying them to expert systems.

We continued work on three established fronts. We completed our design for a transform program generator and implemented enough of it to demonstrate its basic feasibility. We also worked out the details of how a designer can specify alternative data views for a target environment. Finally, we completed significant product development on Gandalf itself, our fundamental environment-generating system.

5.1. Generating Transform Programs

A serious problem in programming environments and operating systems is that existing software becomes invalid when the environment or operating system is replaced by a new release. Unfortunately, there has been no systematic treatment of the issues involved in updating an existing environment. Current approaches are manual, ad hoc, and time-consuming for both environment implementors and program users.

We have developed a way to move existing programs automatically from one version of the underlying environment to the next. To see the potential of automating the transition process, consider that it took us more than half a year to convert from UNIX 4.2 BSD to the newer 4.3 version. With our new strategy, we could accomplish such a conversion in days rather than months.

Our approach eases the tasks facing environment implementors and introduces a higher-level role: the environment designer. Environment descriptions, as well as all existing software, reside in a comprehensive database. The designer specifies a revised environment by changing the formal environment descriptions. A "transformer generator" tracks the alterations, then builds a transformer program that can map old data formats and values into new ones. When an environment user later attempts to access or modify a database-resident program specified under the old structural grammar, the transformer program automatically converts old data structures to the new organization [Garlan et al. 86].

5.2. Toward an Environment Generator with Views

Gandalf environments are typically highly specialized systems for controlling data. Within an environment, a user can manipulate the available database objects via tools that perform legal operations on those objects. A user might employ distinct environments, for instance, when editing text and developing programs. One of our goals is to make it easy to design numerous customized environments and then connect them so the user can migrate among them according to task demands.

Our major concern centered on the idea that both tools and human users want to look at software database objects in different ways at different times. While constructing software, for example, a user may want the environment to display the abstract syntax tree's structure. When maintaining a software system, the programmer may wish to browse a high-level system outline to rapidly locate the place for modifications. Tools, on the other hand, typically don't care about the display at all. A semantic analyzer might search for type declarations but ignore documentation text. A program-managing package might not be concerned with code at all but may want to know who last modified a particular procedure. Our interest is to provide a means of merging the implementation of distinct views so that users and tools can access software objects through a variety of views. We find the problem of extending an existing database particularly interesting. How can new tools and new views be added and integrated with existing ones? New tools are hard to add because, usually, a single data representation must serve all tools. The main difficulty is defining a data format that satisfies all the tools.

Our solution [Garlan 87] extends existing structure-oriented approaches to tool integration by allowing a designer to define a tool in terms of its "views" into the common database. A view describes the data types the tool contains and the primitives that comprise its permissible operations. Tools can share data objects but each tool accesses objects only through its own views. The designer adds new tools by defining new views, and the database thus synthesizes all views that the environment's tools define. We have now specified the types of primitives that will be available in the designer's environment and which will enable him to express view descriptions.

The underlying basis of our approach consists of an object-oriented notation that is independent of any particular programming language and a translator for generating executable code from the notation. Reusable software building blocks are written in this notation, which provides flexible means for combining both data structures and algorithms [Kaiser and Garlan 87]. We used techniques from software generation and object-oriented programming to design a translator to produce efficient executable systems.

From software generation techniques, we used the concept of a declarative notation that is independent of any particular programming language but that can be translated into an efficient implementation. To this idea, we added the object-oriented programming concepts of inheritance and of encapsulating behavior with data structures. We

have added our unique concept of *merging* both data structures and operations. Other object-oriented languages merge data structures, in the sense of inheriting instance variables defined by a superclass, but no other notation besides attribute grammars supports combining algorithms on the basis of dependencies.

5.2.1. Designing an Environment's Views

During the second half of 1986, we completed the first phases of designing and implementing a new structure-oriented environment generator that employs views as its fundamental building blocks. To evaluate our design, we employed it in extending IDL-based systems to support concurrent views. These extensions led to a model of tool integration that combines the flexibility provided by sequentially-oriented tools with the benefits of close cooperation and database management provided by database-oriented tools. The extensions are based on the idea that some IDL descriptions can be treated as views of shared *views* of a common object base. These views provide tools with abstract interfaces to shared data in the same spirit in which IDL structures now provide abstract interfaces between tools.

Our IDL extensions allow several tools to share access to a common data pool. Each tool's data interface is defined by an IDL structure that determines its *view* of the objects. Mappings between views are to be handled automatically by the IDL translator and database support mechanisms.

Our work included the following:

- extending Snodgrass' SoftLab model of tool integration to allow tools to specify collections of structures as common views of a shared pool of objects. Tools thus act as scoping units for sharing and cooperation.
- augmenting the IDL collection types (sequence and set) with additional primitives (indexed table, sorted table, array, multi-set). The purpose of this extension is to give the implementor greater diversity in specifying at an abstract level operationally distinct groupings of objects.
- introducing the notion of type compatability as mappings between operations of one type and those of another. Type compatability extends to IDL structure compatability and provides an operational interpretation for describing a set of objects with two different IDL structures.
- adding notation, called *dynamic views*, that allows one IDL structure to describe its contents in terms of properties of nodes in another structure. Coupled with a shared database, dynamic views serve the function of associate query found in database systems.

We also implemented a special case of views to support display tools. Several working environments now use this implementation. Approximately 2000 students used these environments at CMU, Stanford, and NYU.

5.3. Gandalf Product Development

The Gandalf System forms the foundation for our work in generating environments automatically, providing a workbench for creating and developing interactive programming environments. The system itself includes four specialized environments that a designer uses to specify and fine-tune target user environments, each of which offers task-specific tools and facilities. Gandalf eliminates the economically impractical process of handcrafting individual environments and permits environment designers to generate families of software development environments, semi-automatically and without excessive cost.

Gandalf-produced environments permit interactions not available in traditional environments. They integrate programming tools, which ease the programming process, and system development support, which reduces the degree to which a software project must depend on goodwill among its members. In practice, our industrial customers have built environment prototypes and small control systems where, for example, a user can modify system-supplied templates and icons to prepare reports on physical parameters in a manufacturing process.

In the first half of 1987, we introduced an improved, more marketable Gandalf System. The enhancements represent basic software engineering that will aid a potential environment designer in understanding the system.

5.3.1. Concurrency and segmentation in large software databases

In 1986 the Gandalf System generated environments for programming-in-the-small, though systems like the C Prototype demonstrated that programming-in-the-large and programming-in-the-many can be successfully approached with proper database support. Providing this "proper database support" meant addressing the diverse requirements and operating characteristics that programming-in-the-small, programming-in-the-large, and programming-in-the-many have with respect to the integrated database.

The most promising approach appeared to be a method of segmenting the tree structured databases into a collection of smaller trees. The original structure is preserved (by symbolic pointers between segments) so there is a single virtual database comprised of many smaller segments, each residing in a separate file on secondary storage. This scheme provides the segmentation needed for large software databases since a single user will typically need only one or two segments in his address space at any one time. It supports concurrent access into the database by multiple users since different users can access different segments without readers/writers problems, and semaphores can be associated with segments. This scheme also provides modularity in the database's grammar description. For example, in a version control system, procedures would be stored in separate segments and the module description (which would contain pointers to a set of procedures for that module) would be in another segment. Since the grammar for the module level should be unrelated to the grammar of the procedures, segmentation at the boundary provides a natural means for keeping the two grammars separate.

One of the issues in this scheme that we had to address was the behaviour of the system at and across the boundaries of segments. An early version of the Gandalf System used a paging scheme top fault in segments whenever needed. Any node in a database grammar could be designated as a "filenode" that would be a root for a new segment. This was to be completely transparent to the user. This approach failed because there was no way to localize the page-in/page-out code for segments. All operations in the database kernel could never be sure if they were looking at a filenode or real node, so each operation had to test. Even implementor-written semantic routines would have to test for filenodes. This code became very hard to maintain and very expensive to execute. Also, there was neither concurrency nor modularity in the database grammar.

To solve these problems, we decided to incorporate into the grammar a special type of terminal node that corresponds to a segment. This terminal node contains the symbolic pointer to the actual segment. Nodes of this type will usually appear as any other terminal node to the user, the database kernel, and the semantic routines. A few special operations will allow the system to change its focus of attention from one segment to another. Since we expect segment boundaries to occur at natural points, such as in the module and procedure example given above, explicit commands and operations to change context should also be natural for users and semantic daemons. This eliminated the problems associated with the filenode approach by localizing the segment swapping code to those special operations.

5.3.2. Specifying tools

During the second half of 1985 we focused our attention on specifying tools for a software development environment. We incorporated the language design from the first half of 1985 into a new version of the Gandalf System for generating language-oriented software development environments. Toward this end, we rewrote the Gandalf Kernel to support the description of tools written in Action Routine Language (ARL) [Ambriola and Staudt 86]. This involved adding kernel support for ARL primitives, attributes, signals, and transactions.

We also implemented the implementor's environment for producing structure editors using ARL. This environment consists of two systems, ALOEGEN for generating environment descriptions, and DBgen for linking environment descriptions to form a working editor. Both systems were bootstrapped using ARL itself. We distributed copies of the new system, including a tutorial introduction, as a beta test release to a number of research groups both within the CMU computer science department and outside [Staudt 86].

In a parallel development, we implemented a new generation of user interface to run on inexpensive bitmapped personal computers such as the Macintosh. We based it on VIZ, a language for describing flexible unparsing, and developed a corresponding user interface to support multiple views of a program [Garlan 85]. This was made available to the general public within the year as a novice programming environment for Pascal.

5.4. Bibliography

[Ambriola and Montangero 85]
Ambriola, V. and C. Montangero.
Automatic generation of execution tools in a Gandalf environment.
The Journal of Systems and Software 5(2):155-172, May, 1985.
The formal definition of a programming language in denotational style is taken as the basis for the automatic generation of its interpreter. The facilities available in Gandalf are exploited to implement and integrate such a generation technique in a Gandalf environment.

[Ambriola and Staudt 86]
Ambriola, V. and B.J. Staudt.
The ALOE action routine language manual.
Technical Report CMU-CS-86-129, Carnegie Mellon University Computer Science Department,
May, 1986.
ARL (Action Routine Language) is a special-purpose language for the manipulation of abstract syntax trees with attributes. It is the language in which the users of the Gandalf System write the semantic processing of Gandalf-style environments. This document describes ARL 1.3.
The Gandalf system provides a user with the ability to generate a language-based programming environment given a description of the language. The syntax of the language is described using a BNF-like notation. The semantics of the language is described using attributes and ARL routines. The syntactic and semantic language descriptions are linked with a standard editing kernel provided by the Gandalf system to produce a language-specific programming environment. Environments generated in this manner are often called ALOEs (A Language-Oriented Editor). The terms Gandalf environment and Aloe are used interchangeably in this paper. Programs that are written with a Gandalf environment are maintained as attributed syntax trees, not text. ARL was therefore designed with trees as the basic data type. The primitive operations in ARL provide facilities to browse and manipulate tree structures.

[Anantharaman et al. 85]
Anantharaman, T.M., E.M. Clarke, M.J. Foster, and B. Mishra.
Compiling path expressions into VLSI circuits.
In *Twelfth Annual ACM Symposium on Principles of Programming Languages*, ACM, January, 1985.
Path expressions were originally proposed by Campbell and Habermann as a mechanism for process synchronization at the monitor level. We argue that path expressions are also useful for specifying the behavior of complicated

asynchronous circuits, and in this paper we consider the possibility of directly implementing them in hardware.

Our implementation is complicated in the case of multiple path expressions by the need for synchronization on event names that are common to more than one path. Moreover, since events are inherently asynchronous in our model, all of our circuits must be self-timed.

Nevertheless, the circuits produced by our construction have area proportional to N@k(dot)log(N) where N is the total length of the multiple path expression under consideration. This bound holds regardless of the number of individual paths or the degree of synchronization between paths.

[Barbacci et al. 85a]
Barbacci, M.R., W.H. Maddox, T.D. Newton, and R.G. Stockton.
The Ada+ front end and code generator.
In *Proceedings of the 1985 International Ada Conference: Ada in Use*, May, 1985.

The Ada + compiler is being written as a part of the Spice project at Carnegie Mellon University, and is intended eventually to be a full implementation of the Ada programming language. A preliminary version has been released within the university and runs on both the PERQ workstation and the Digital Equipment Corporation VAX, producing code for the PERQ under the Accent operating system. This paper deals with the compilation issues of the Ada+ project.

[Barbacci et al. 85b]
Barbacci, M.R., S. Grout, G. Lindstrom, M. Maloney, E. Organick, and D. Rudsill.
Ada as a hardware description environment: an initial report.
In *Proceedings of the IFIP Seventh International Symposium on Computer Hardware Description Languages*, CHDL, August, 1985.
Also available as Technical Report CMU-CS-85-104.

This paper reports on our initial results in using Ada as a Hardware Description Language. Ada provides abstraction mechanisms to support the development of large software systems. Separate compilation as well as nesting of packages, tasks, and subprograms allow the construction of modular systems communicating through well defined interfaces. The complexity of modern chips (e.g. those proposed in the VHSIC program) will require the use of those features that make Ada a good language for programming-in-the-large.

[Barbacci et al. 85c]
Barbacci, M.R., G. Lindstrom, M. Maloney, and E. Organick.
Representing time and space in an object oriented hardware description language.
Technical Report CMU-CS-85-105, Carnegie Mellon University Computer Science Department,
January, 1985.

Hardware description languages (HDLs) will clearly play a vital role in the comprehensive VLSI design tools of the future. Now that the requirements for such HDLs are becoming better understood, it is becoming increasingly evident that the central issues are abstraction, modularity, and complexity management --- the same issues faced by designers of large scale software systems, rather than low-level technological details (although these must ultimately be served as well).

Consequently, we argue that Ada, constituting the most advanced, carefully conceived, and (soon to be) widely available modern high-order programming language, forms not only an adequate but a compelling choice as an HDL. Specifically, Ada offers separate compilation as well as nesting of packages, tasks, and subprograms. These, and other important features of Ada, allow the construction of modular systems communicating through well defined interfaces.

This paper demonstrates how placement and routing information can be incorporated into Ada hardware descriptions: another paper, "Ada as a Hardware Description Language: An Initial Report", submitted to the IFIP 7th International Symposium on Computer Hardware Description Languages and their Applications, Tokyo, August 1985 shows how component and signal propagation delays over carriers are also incorporated into the same hardware descriptions.

[Brookes 85a]
Brookes, S.D. and A.W. Roscoe.
Deadlock analysis in networks of communicating processes.
Technical Report CMU-CS-85-111, Carnegie Mellon University Computer Science Department,
February, 1985.

We use the failures of Communicating Sequential Processes to describe the behavior of a simple class of networks of communicating processes, and we demonstrate this fact by proving some results which help in the analysis of deadlock in networks. In particular, we formulate some simple theorems which characterize the states in which deadlock can occur, and use them to prove some theorems on the absence of global deadlock in certain classes of systems. Some examples are given to show the utility of these results.

[Brookes 85b] Brookes, S.
On the axiomatic treatment of concurrency.
Technical Report CMU-CS-85-106, Carnegie Mellon University Computer Science Department,
February, 1985.

This paper describes a semantically-based axiomatic treatment of a simple parallel programming language. We consider an imperative language with shared variable concurrency and a critical region construct. After giving a structural operational semantics for the language we use the semantic structure to suggest a class of assertions for expressing semantic properties of commands. The structure of the assertions reflects the structure of the semantic representation of a command. We then define syntactic operations on assertions which correspond precisely to the corresponding syntactic constructs of the programming language; in particular, we define sequential and parallel composition of assertions. This enables us to design a truly compositional proof system for program properties. Our proof system is sound and relatively complete. We examine the relationship between our proof system and the Owicki-Gries proof system for the same language, and we see how Owicki's parallel proof rule can be reformulated in our setting. Our assertions are more expressive than Owicki's, and her *proof outlines* correspond roughly to a special subset of our assertion language. Owicki's parallel rule can be thought of as being based on a slightly different form of parallel composition of assertions; our form does not require *interference-freedom*, and our proof system is relatively complete without the need for auxiliary variables. Connections with the 'Generalized Hoare Logic' of Lambort and Schnieder, and with the Transition Logic of Gerth, are discussed briefly, and we indicate how to extend our ideas to include some more programming constructs, including conditional commands, conditional critical regions, and loops.

[Brookes and Roscoe 85]
Brookes, S.D. and A.W. Roscoe.
An improved failures model for communicating processes.
Technical Report CMU-CS-85-112, Carnegie Mellon University Computer Science Department,
February, 1985.

We extend the failures model of communicating processes to allow a more satisfactory treatment of divergence in addition to deadlock. The relationship between the revised model and the old model is discussed, and we make some connection with various models proposed by other authors.

[Bruegge 85a] Bruegge,B.
Adaptability and portability of symbolic debuggers.
Technical Report CMU-CS-85-174, Carnegie Mellon University Computer Science Department,
September, 1985.

The design and implementation of symbolic debuggers for complex software systems is not a well understood area. This is reflected in the inadequate functionality of existing debuggers, many of which are seldomly used. For example, 30% of all programmers asked in a questionnaire (which was distributed as part of this thesis work) do not use a debugger at all or only very infrequently (Bruegge, 1984). Yet debugging tools are needed: Many software systems are produced by the cooperative effort to many designers and programmers, sometimes over several years, resulting in products that inevitably contain bugs.

[Bruegge 85b] Bruegge, B.
Debugging Ada.
Technical Report CMU-CS-85-127, Carnegie Mellon University Computer Science Department,
May, 1985.

The complexity of the Ada language poses several problems for the builder of a debugger. We identify the Ada language constructs that cause these problems and propose solutions that can be incorporated in a debugger based on Pascal. Several of the solutions involve changes in the symbol table of the Ada compiler, others are based on the argument that having to obey the language rules is an obstacle when debugging programs.

[Bryant 85] Bryant, R.E.
Symbolic verification of MOS circuits.
Technical Report CMU-CS-85-120, Carnegie Mellon University Computer Science Department,
April, 1985.

The program MOSSYM simulates the behavior of a MOS circuit represented as a switch-level network *symbolically*. That is, during simulator operation the user can set an input to either 0, 1, or a Boolean variable. The simulator then computes the behavior of the circuit as a function of the past and present input variables. By using heuristically efficient Boolean function manipulation algorithms, the verification of a circuit by symbolic simulation can proceed much more quickly than by exhaustive logic simulation. In this paper we present our concept of symbolic simulation, derive an algorithm for switch-level symbolic simulation, and present experimental measurements from MOSSYM.

[Chandhok et al. 85]
Chandhok, R., D. Garlan, D. Goldenson, P. Miller, and and M. Tucker.
Programming environments based on structure editing: the GNOME approach.
In *AFIPS Conference Proceedings of the 1985 National Computer Conference*, Pages 359-369. 1985.
The use of integrated programming environments based on structure editing is an emerging technology that has now reached the stage of being both demonstrably useful and readily implementable. We have outlined some of the salient aspects of our work in developing the GNOME and MacGNOME programming environments and suggested paths of implementation that seem to be worth traveling. A predominant theme in all of this has been the need to separate policy from mechanism. While the choice of user interface policies will probably differ widely from those we have made here, the mechanisms that we have sketched will nonetheless be applicable to future environments.

[Dill and Clarke 85]
Dill, D.L. and E.M. Clarke.
Automatic verification of asynchronous circuits using temporal logic.
In *1985 Chapel Hill Conference on VLSI*, Computer Science Press, May, 1985.
Also available as Technical Report CMU-CS-85-125.
We present a method for automatically verifying asynchronous sequential circuits using temporal logic specifications. The method takes a circuit described in terms of boolean gates and Muller elements, and derives a state graph that summarizes all possible circuit executions resulting from any set of finite delays on the outputs of the components. The correct behavior of the circuit is expressed in CTL, a temporal logic. This specification is checked against the state graph using a 'model checker' program. Using this method, we discover a timing error in a published arbiter design. We give a corrected arbiter, and verify it.

[Durham 86]
Durham, I.
Abstraction and the methodical development of fault-tolerant software.
PhD thesis, Carnegie Mellon University Computer Science Department, February, 1986.
Also available as Technical Report CMU-CS-86-112.
The reliable operation of software is a factor of increasing importance with the use of computers for critical functions. Software in general is demonstrably unreliable, particularly in the presence of external failures. Software that continues to provide reliable, if degraded, service in spite of

external failures is termed *Fault-Tolerant*. Fault-tolerant software uses redundancy in code and data to recover from failures. Because few tools are available to guide the introduction of redundancy for the most cost-effective improvement in reliability, an *ad hoc* approach is commonly used. Unfortunately, such an approach cannot guarantee that the most serious potential failures have even been recognized. There is, therefore, a need for a methodical approach to deciding where to introduce redundancy. Abstraction has provided a foundation for the methodical development of correct software. As a conceptual tool, it simplifies the structure of software and supports both the precise specification of its behavior in the absence of failures and the ease of reasoning about it. This thesis provides a foundation for the methodical development of fault-tolerant software using abstraction as the basis for describing both failures and the behavior of software in the presence of those failures.

[Ellison and Staudt 85]
Ellison, R.J. and B.J. Staudt.
The evolution of the Gandalf system.
The Journal of Systems and Software 5(2):107-120, May, 1985.
The Gandalf System is used to generate highly interactive software development environments. This paper describes some design decisions made during the development of the Gandalf system and the system's applicability to the generation of single-user programming environments and multi-user software development environments.

[Garlan 85]
Garlan, D.
Flexible unparsing in a structure editing environment.
Technical Report CMU-CS-85-129, Carnegie Mellon University Computer Science Department,
April, 1985.
Generators of structure editing-based programming environments require some form of *unparse specification language* with an implementor that can describe mappings between objects in the programming environment and concrete, visual representations of them. They must also provide an *unparser* to execute those mappings in a running programming environment. We describe one such unparse specification language, called VIZ, and its unparser, called UAL. VIZ combines in a uniform descriptive framework a variety of capabilities to describe flexible views of a programming database using a library of high-level formatting routines that can be customized and extended by the implementor. The UAL unparser allows the highly conditional unparse mappings of VIZ to be executed

efficiently. Its implementation is based on the automatic generation of explicit *display views,* together with a scheme for efficient incremental updating of them in response to arbitrary changes to objects in the programming environment.

[Garlan 86] Garlan, D.
Views for tools in integrated environments,
Proceedings of the 1986 International Workshop on Advanced Programming Environments. Springer-Verlag, 1986.
This paper addresses the problem of building tools for integrated programming environments. Integrated environments have the desirable property that the tools in it can share a database of common structures. But they have the undesirable property that these tools are hard to build because typically a single representation of the database must serve all tools. The solution proposed in this work allows tools to maintain appropriate representations or "views" of the objects they manipulate while retaining the benefits of shared access to common structures. We illustrate the approach with two examples of tools for an environment for programming-in-the-large, and outline current work in progress on efficient implementations of these ideas.

[Garlan 87] Garlan, D.
Views for tools in integrated environments.
Technical Report CMU-CS-87-147, Carnegie Mellon University Computer Science Department,
May, 1987.
Integrated environments have the desirable property that the tools in them may share a database of common structures. But they have the undesirable property that tools are hard to add to an environment because typically a single representation of the database must serve the needs of all tools. The solution described in this thesis allows an implementor to define each tool in terms of a collection of "views" of the objects to be manipulated. A view is a description of a common database, defined in such a way that objects can be shared among a collection of tools, each tool accessing objects through the views it defines. New tools are thus added by defining new views. The common database then becomes the synthesis of all of the views defined by the tools in the environment.

[Garlan et al. 86] Garlan, D., C.W. Krueger, and B.J. Staudt.
A structural approach to the maintenance of structure-oriented environments.
In *Proceedings of The ACM SIGSOFT/SIGPLAN Software Engineering symposium on Practical Software Development Environments*, ACM SIGSOFT/SIGPLAN, Palo Alto, CA, December, 1986.

A serious problem for programming environments and operating systems is that existing software becomes invalid when the environment or operating system is replaced by a new release. Unfortunately, there has been no systematic treatment of the problem; current approaches are manual, ad hoc, and time consuming both for implementors of programs and for their users. In this paper we present a new approach. Focusing on a solution to the problems for structure-oriented environments, we show how automatic converters can be generated in terms of an implementor's changes to formal descriptions of these environments.

[Gunter 85] Gunter, C.
Profinite solutions for recursive domain equations.
Technical Report CMU-CS-85-107, Carnegie Mellon University Computer Science Department,
February, 1985.

The purpose of the dissertation is to introduce and study the category of profinite domains. The study emphasizes those properties which are relevant to the use of these domains in a semantic theory, particularly the denotational semantics of computer programming languages. An attempt is made to show that the profinites are an especially natural and, in a sense, *inevitable* class of spaces. It is shown, for example, that there is a rigorous sense in which the countably based profinites are the largest category of countably based spaces closed under the function space operation. They are closely related to other categories which appear in the domain theory literature, particularly strongly algebraic domains (**SFP**) which form a significant subcategory of the profinites. The profinites are *bicartesian closed*-a noteworthy property not not possessed by **SFP** (because it has no coproduct). This gives rise to a rich type structure on the profinites which makes them a pleasing category of semantic domains.

[Habermann 85] Habermann, A.N.
Automatic generation of execution tools in a Gandalf environment.
The Journal of Systems and Software 5(2):145-154, May, 1985.

Information generated in a programming environment is often allowed to grow indefinitely. Designer and user alike are counting on standard backup and disc clearing procedures

for archiving old data. In this paper we take the view that one should distinguish between relevant old data that is purposely archived and obsolete information that should automatically be deleted. The two main topics of the paper are the strategies and mechanisms for deleting information and the facilities available to designers of programming environments to specify deletion strategies. Information can be deleted applying a passive or an active strategy. With the passive strategy, information will not actually be deleted until it is certain that there is no interest in it any longer. With the active strategy, an object is immediately deleted when it becomes obsolete, while users of the object are notified of the deletion event. This paper discusses various implementation of these two strategies and shows when they apply. Taking the view that it must be easy to modify and fine tune programming environments, much attention must be given to the designers support environment for generating programming environments. This paper discusses in particular the facilities for expressing the semantics of names in an environment. Various naming modes are useful for a designer to specify the deletion strategies for his target programming environment. Details are illustrated by applying the ideas to an environment for software development and maintenance.

[Habermann 86] Habermann, A.N.
Technological advances in software engineering.
In *Proceedings of the 1986 ACM Computer Science Conference*,
Pages 29-37. ACM, Cincinnati, February, 1986.

A major challenge for software engineering today is to improve the software production process. Nowadays, most software systems are handcrafted, while software project management is primarily based on tenuous conventions. Software engineering faces the challenge of replacing the conventional mode of operation by computer-based technology. This theme underlies the Software Engineering Institute that the DoD has established at Carnegie Mellon University. Among the contributors to software development technology are ideas, such as workstations, and programming environments that provide integrated sets of tools for software development and project management. Facilities and tools are by themselves not sufficient to achieve an order of magnitude improvement in the software production process. Future directions in software engineering must emphasize a constructive approach to the design of reusable software and to automatic generation of programs. The author briefly explores the promising technology that can be used to implement these ideas.

[Habermann and Notkin 86]
Habermann, A.N. and D.S. Notkin.
Gandalf software development environments.
IEEE Transactions on Software Engineering, December, 1986.
Software development environments help programmers perform tasks related to the software development process. Different programming projects require different environments. However, handcrafting a separate environment for each project is not economically feasible. Gandalf solves this problem by permitting environment designers to generate families of software development environments semiautomatically without excessive cost.
Environments generated using Gandalf address both programming environments, which help ease the programming process, and system development environments, which reduce the degree to which a software project is dependent on the good will of its members. Gandalf environments integrate programming and system development, permitting interactions not available in traditional environments.
The paper covers several topics including the basic characteristics of Gandalf environments, our method for generating these environments, the structure and function of several existing Gandalf environments, and ongoing and planned research of the project.

[Herlihy 85a]
Herlihy, M.
Atomicity vs. availability: concurrency control for replicated data.
Technical Report CMU-CS-85-108, Carnegie Mellon University Computer Science Department,
February, 1985.
Data managed by a distributed program may be subject to consistency and availability requirements that must be satisfied in the presence of concurrency, site crashes, and network partitions. This paper proposes two integrated methods for implementing concurrency control and replication for data of abstract type. Both methods use quorum consensus. The Consensus Locking method minimizes constraints on availability, and the Consensus Scheduling method minimizes constraints on concurrency. These methods systematically exploit type-specific properties of the data to provide better availability and concurrency than methods based on the conventional read/write classification of operations. Necessary and sufficient constraints on correct implementations are derived directly from the data type specification. These constraints reveal that an object cannot be replicated in a way that simultaneously minimizes constraints on both availability and concurrency.

[Herlihy 85b] Herlihy, M.
Using type information to enhance the availability of partitioned data.
Technical Report CMU-CS-85-119, Carnegie Mellon University Computer Science Department,
April, 1985.

A partition occurs when functioning sites in a distributed system are unable to communicate. This paper introduces a new method for managing replicated data in the presence of partitions. A novel aspect of this method is that it systematically exploits type-specific properties of the data to support better availability and concurrency than comparable methods in which operations are classified only as reads or writes. Each activity has an associated level, which governs how it is serialized with respect to other activities. Activities at the same level are serialized systematically, but higher-level activities are serialized after lower-level activities. A replicated data item is a typed object that provides a set of operations to its clients. A quorum for an operation is any set of sites whose co-operation suffices to execute that operation, and a quorum assignment associates a set of quorums with each operation. Higher-level activities executing 'in the future' may use different quorum assignments than lower-level activities executing 'in the past.' Following a failure, an activity that is unable to make progress using one quorum assignment may switch to another by restarting at a different level.

[Herlihy 85c] Herlihy, M.
Comparing how atomicity mechanisms support replication.
Technical Report CMU-CS-85-123, Carnegie Mellon University Computer Science Department,
May, 1985.

Most pessimistic mechanisms for implementing atomicity in distributed systems fall into three broad categories: two-phase locking schemes, timestamping schemes, and hybrid schemes employing both locking and timestamps. This paper proposes a new criterion for evaluating these mechanisms: the constraints they impose on the availability of replicated data.

A replicated data item is a typed object that provides a set of operations to its clients. A quorum for an operation is any set of sites whose co-operation suffices to execute that operation, and a quorum assignment associates a set of quorums with each operation. Constraints on quorum assignment determine the range of availability properties realizable by a replication method.

This paper compares the constraints on quorum assignment necessary to maximize concurrency under generalized locking, timestamping, and hybrid concurrency control

mechanisms. This comparison shows that hybrid schemes impose weaker constraints on availability than timestamping schemes, and locking schemes impose constraints incomparable to those of the others. Because hybrid schemes permit more concurrency than locking schemes, these results suggest that hybrid schemes are preferable to the others for ensuring atomicity in highly available and highly concurrent distributed systems.

[Hisgen 85] Hisgen,A.
Optimization of user-defined abstract data types: a program transformation approach.
Technical Report CMU-CS-85-166, Carnegie Mellon University Computer Science Department,
September, 1985.

This dissertation introduces a programming language facility for optimizing user-defined abstract data types. Current optimizing compilers have concentrated on the optimization of built-in, predefined types, for example, the integers. This work investigates the possibility of extending the benefits of program optimization to user-defined abstract data types. The programmer of an abstract data type writes transformations that state when one operation of the type (or sequence of operations) may be replaced by another operation (or sequence of operations). A transformation may have an enabling precondition, which says that it is legitimate only in contexts in which the enabling precondition can be shown to be true. When compiling a program that is a client of the type, the compiler analyzes the client's calls on the operations of the type and attempts to apply the transformations to particular calls (or sequences of calls).

This dissertation presents a language for writing transformations between the operations of an abstract data type. The transformation language also includes facilities for writing specifications for the type in a manner that caters to the task of optimization. Examples of data types that can exploit the transformation language are given. Techniques for compiling client programs are described.

[Kaiser 85a] Kaiser, G.E.
Semantics for structure editing environments.
Technical Report CMU-CS-85-131, Carnegie Mellon University Computer Science Department,
May, 1985.

This thesis addresses the processing of semantics by structure editor-based programming environments. This processing is performed incrementally while the user writes and tests her programs. The semantics processing involves the

manipulation of two kinds of properties, static and dynamic. The implementor of a programming environment describes the semantics processing in terms of these properties.

Recent research in structure editing environments has focused on the generation of programming environments from description. Several mechanisms have been proposed, and the most successful of these have been action routines and attribute grammars. Using action routines, written as a collection of imperative subroutines, it is difficult to anticipate all possible interactions that may result in adverse behavior. Attribute grammars are written in a declarative style and the implementor need not be concerned with subtle interactions because all interactions among attribute grammar rules are handled automatically. Unfortunately, attribute grammars have hitherto seemed unsuited to the description of dynamic properties.

This thesis describes a very large extension to attribute grammars that solves this problem. The extended paradigm is called *action equations*. Action equations are written in a declarative notation that retains the flavor of attribute grammars but adds an easy means to express both dynamic properties and static properties. The extensions include attaching particular attribute grammar-style rules to events that represent user commands; supporting propagation both of events and of change with respect to attribute values; limited support for non-applicative mechanisms, allowing attributes to be treated as variables and permitting both modification of and replacement for changes to attribute values. Together, these extensions are sufficient to support dynamic properties.

[Kaiser 85b] Kaiser, G.E. and E. Kant.
Incremental parsing without a parser.
The Journal of Systems and Software 5(2):121-144, May, 1985.

This article describes an algorithm for incremental parsing of expressions in the context of syntax-directed editors for programming languages. Since a syntax-directed editor represents programs as trees and statements and expressions as nodes in trees, making minor modifications in an expression can be difficult. Consider, for example, changing a '+' operator to a '*' operator or adding a short subexpression at a syntactically but not structurally correct position, such as inserting ') * (d' at the # mark in ' (a + b # + c)'. To make these changes in a typical syntax-directed editor, the user must understand the tree structure and type a number of tree-oriented construction and manipulation commands. This article describes an algorithm that allows the user to think in terms of the syntax of the expression as it is displayed on the screen (in infix notation)

rather than in terms of its internal representation (which is effectively prefix), while maintaining the benefits of syntax-directed editing. This algorithm is significantly different from other incremental parsing algorithms in that it does not involve modifications to a traditional parsing algorithm or the overhead of maintaining a parser stack or any data structure other than the syntax tree. Instead, the algorithm applies tree transformations, in real-time as each token is inserted or deleted, to maintain a correct syntax tree.

[Kaiser 86] Kaiser, G.E.
Generation of run-time environments.
In *SIGPLAN '86 Symposium on Compiler Construction*, June, 1986.
Attribute grammars have been used for many years for automated compiler construction. Attribute grammars support the description of semantic analysis, code generation and some code optimization in a formal declarative style. Other tools support the automation of lexical analysis and parsing. However, there is one large part of compiler construction that is missing from our toolkit: run-time environments. This paper introduces an extension of attribute grammars that supports the generation of run-time environments. The extension also supports the generation of interpreters, symbolic debugging tools, and other execution-time facilities.

[Kaiser and Garlan 87]
Kaiser, G.E. and D. Garlan.
Composing software systems from reusable building blocks.
In *The Twentieth Hawaii International Conference on System Sciences (HICSS-20)*, Kona, HA, January, 1987.
Current approaches to software reuse have had little effect on the practice of software engineering. Among the reasons that most existing approaches have been so limited is the fact that they result in software that is highly tied to linguistic and/or functional context. A software building block can be reused only in a manner envisioned by the original programmer. A generic stack module written in Ada can only be used for manipulating stacks, and only within an Ada environment. A window manager written in any programming language can only be used as a window manager.
We argue that to achieve an order of magnitude improvement in software production, we need to support software reusability that has three important characteristics: (a) language-independence, (b) support for component reuse through composition, and (c) the ability to reuse a component in a way not anticipated by the original programmer. We describe a framework for achieving these three

goals. The important components of the famework are *features*, a unit for modularity that can be composed in a manner similar to the multiple inheritance of object-oriented languages and *action equations*, a declarative notation for specifying the behavior of software building blocks.

[Newton 86a] Newton, T.D.
An implementation of Ada generics.
Technical Report CMU-CS-86-125, Carnegie Mellon University Computer Science Department,
May, 1986.

This paper describes the technique used for implementing generics in the Ada+ compiler. It involves performing semantic analysis on generic units, producing code for instantiations by generic expansion, and preserving the results of semantic analysis on a template in its copies.

One of the more interesting features of the Ada programming language is the capability to define *generic* subprograms and packages which can be parameterized by types and subprograms as well as by objects. By allowing the reuse of code, generic units can save programming time and increase reliability. However, while generic units are a nice tool from a programmer's point of view, they pose an added burden for a compiler both in terms of semantic analysis and in terms of code generation. This paper is an attempt to describe how the Ada+ compiler deals with the problems posed by generic units.

[Newton 86b] Newton, T.D.
A survey of language support for programming in the large.
Technical Report CMU-CS-86-124, Carnegie Mellon University Computer Science Department,
May, 1986.

The support provided by a number of programming languages for the activity of programming in the large is examined, and their features are categorized with respect to decomposition of a system, import/export mechanisms, separate compilation, and version/configuration control. A comparison is made using this categorization. Eighteen languages are surveyed; ranging from Simula-67 to Modula-2 to Ada to BCPL, they exhibit a number of design philosophies.

[Newton et al. 85] Newton, T.D., W.H. Maddox, and R.G. Stockton.
User's guide to the Ada+ compiler
1985.

[Notkin 85] Notkin, D.
The Gandalf project.
The Journal of Systems and Software 5(2):91-106, May, 1985.

The Gandalf project is concerned with the automated generation of software development environments. In particular, the project has considered project management environments, system version control environments, and incremental programming environments. The artifacts surrounding these environments are described. Later versions of these environments have been constructed as structure editors. The processes and tools involved in generating structure editors for software development environments are also discussed. Future plans of the project are briefly mentioned.

[Saraswat 86] Saraswat, V.A.
Problems with concurrent Prolog.
Technical Report CMU-CS-86-100, Carnegie Mellon University Computer Science Department,
January, 1986.

In this paper I argue that pure Horn logic does not provide the correct conceptual framework for concurrent programming. In order to express any kind of useful concurrency some extra-logical apparatus is necessary. The semantics and proof systems for such languages must necessarily reflect these control features, thus diluting the essential simplicity of Horn logic programming.

In this context I examine Concurrent Prolog as a concurrent and as a logic programming language, highlighting various semantic and operational difficulties. My thesis is that Concurrent Prolog is best thought of as a set of control features designed to select some of the many possible execution paths in an inherently non-deterministic language. It is perhaps not a coherent set of control and data-features for the ideal concurrent programming language. It is not a Horn logic programming language because it does not distinguish between derivations and refutations, because of its commitment to don't care indeterminism. As a result, soundness of the axioms does not guarantee a natural notion of partial correctness and the failure-as-negation rule is unsound. Because there is no don't know determinism, all search has to be programmed, making it a much more procedural rather than declarative language.

Moreover, we show that its proposed '?' (read-only) annotation is under-defined and there does not seem to be any consistent, reasonable way to extend its definition. We propose and justify alternate synchronization and commitment annotations.

[Scherlis 86] Scherlis, W.L.
Abstract data types, specialization, and program reuse.
In *International Workshop on Advanced Programming Environments*, ACM SIGPLAN/SIGSOFT, April, 1986.

It is often asserted that our ability to reuse programs is limited primarily by the power of programming language abstraction mechanisms. We argue that, on the basis of performance considerations, this is just not the case in practice -- these generalization mechanisms must be complemented by techniques to adapt the generalized structures to specific applications. Based on this argument, we consider a view of programming experience as a network of programs that are generalizations and specializations on one another and that are interconnected by appropriate program derivation fragments. We support this view with a number of examples. These examples illustrate the important role of abstract data type boundaries in program derivation.

[Scherlis and Jorring 86]
Scherlis, W.L. and U. Jorring.
Compilers and staging transformations.
In *Proceedings of the Thirteenth POPL Conference*, ACM, January, 1986.

Computations can generally be separated into stages, which are distinguished from one another by either frequency of execution or availability of data. *Precomputation* and *frequency reduction* involve moving computation among a collection of stages so that work is done as early as possible (so less time is required in later steps) and as infrequently as possible (to reduce overall time).

We present, by means of examples, several general transformation techniques for carrying out precomputation transformations. We illustrate the techniques by deriving fragments of simple compilers from interpreters, including an example of Prolog compilation, but the techniques are applicable in a broad range of circumstances. Our aim is to demonstrate how perspicuous accounts of precomputation and frequency reduction can be given for a wide range of applications using a small number of relatively straightforward techniques.

Related work in partial evaluation, semantically directed compilation, and compiler optimization is discussed.

[Shombert 85] Shombert,L.A.
Using redundancy for testable and repairable systolic arrays.
Technical Report CMU-CS-85-157, Carnegie Mellon University Computer Science Department,
August, 1985.

This thesis presents a method of using spares to enhance the reliability and testability of systolic arrays. The method, called roving spares, provides fault detection and fault isolation without interrupting array operation, essentially providing a self testing array. Systolic arrays are defined and the design space of systolic arrays is identified. The methodology for roving spares on the simplest, but still very powerful, type of systolic array is then derived. Several detailed designs are generated to provide sample data points for the analysis that follows. The analysis shows that reliability is increased by factors to two to ten, over a nonredundant array, and that this improvement is achieved at low cost. The testing capability of roving spares does not significantly decrease the reliability benefits of spares. A brief analysis of a more complex systolic array indicates that the benefits achievable for the simple array can be expected for all types of systolic arrays.

[Staudt 86] Staudt, B.J., C.W. Krueger, A.N. Habermann, and V. Ambriola.
The Gandalf system reference manuals.
Technical Report CMU-CS-86-130, Carnegie Mellon University Computer Science Department,
May, 1986.

The Gandalf System is a workbench for the creation and development of interactive programming environments. The system consists of several components that an implementor uses for designing and fine tuning a user environment with task-specific tools and facilities. This report is a collection of three documents describing the use of the Gandalf System: *The Gandalf Editor Generator Reference Manuals*, *The Aloe Action Routine Language Manual*, and *The Implementor's Guide to Writing Daemons for Aloe.*

[Stockton 85] Stockton, R.G.
Overload resolution in Ada+.
Technical Report CMU-CS-85-186, Carnegie Mellon University Computer Science Department,
December, 1985.

This paper describes one technique for performing Ada overload resolution. It involves a bottom-up scan of an attributed syntax tree which examines all possible interpretations of an expression and filters out all invalid interpretations.

One of the many useful features of the Ada programming language is the capability to overload various symbols. Although this can contribute immensely to the readability of programs, it places a much greater burden upon the compiler, since the meaning of a symbol can not always be

uniquely determined based upon its name. In fact, in some cases there might be several equally valid interpretations of the given symbol. The compiler must determine, based on the context of the symbol, which of the possible interpretations is the correct one. This document is an attempt to describe the way in which the Ada+ compiler accomplishes this task.

[Wadler 84] Wadler,P.L.
Listlessness is better than laziness.
Technical Report CMU-CS-85-171, Carnegie Mellon University Computer Science Department,
August, 1984.
The thesis is about a style of applicative programming, and a program transformation method that makes programs written in the style more efficient. It concentrates on a single, important source of clarity and inefficiency in applicative programs: the use of structures to communicate between components of a program.

[Wilber 84] Wilber, R.
White pebbles help.
Technical Report CMU-CS-85-101, Carnegie Mellon University Computer Science Department,
December, 1984.
A family of directed acyclic graphs of vertex in degree 2 is constructed for which there are strategies of the black-white pebble game that use asymptotically fewer pebbles than the best strategies of the black pebble game. This shows that there are straight-line programs that can be evaluated.

6. DISTRIBUTED SENSOR NETWORKS

The long term goal of our research is to seek problem solutions that would influence both the long-term design goals of DSNs and distributed intelligence systems in general. Systems for building DSNs are complex. Their design raises a number of difficult issues including:

- Suitable physical structures and system architectures for fault-tolerant computation.
- Languages and tools to assist in creating and debugging programs for a distributed environment.
- Techniques for distributed signal processing.
- Construction and maintenance of distributed knowledge structures.

Though our research is particularly relevant to various aspects of the distributed sensor network problem, it more generally relates to any system that has distributed data, which are concurrently read and updated by a potentially large collection of processes. Much of this work was done using results from our earlier research on Accent and TABS [Spector et al. 85a].

Our objective has been to develop a distributed transaction facility and associated linguistic support to simplify the construction of and interoperability of databases of all types, particularly those that require continued access despite the occurrence of failures. We also began to develop example applications that use such databases, and continue to function despite failures. All of our work is machine independent and designed to use Mach and both uni- and multi-processors. These facilities simplify programming by freeing the application programmer from many reliability and concurrency concerns. With the facilities we are developing, application programs appear to run sequentially, despite other concurrently-executing programs. Programs also can more easily deal with failures, because failures are guaranteed to never leave programs in inconsistent states.

Our work divided into two components: In one component, we were winding down our experimentation with the Accent operating system, but learning as many lessons as possible from that testbed. In the other, we began the major effort to design and implement the Camelot distributed transaction facility and the Avalon language extensions to C++, Common Lisp, and Ada. As part of our integrational and systems development work, there were numerous opportunities to innovate in the development of algorithms. We have developed efficient algorithms for restoring the state of computers after failures, managing disk storage, coordinating multiple computers that are involved with a transaction, and storing replicated data. Overall, our research efforts in DSN generally divide into three categories: developing algorithms, developing systems, and analyzing systems.

6.1. Algorithm development

During 1986 we analyzed the trade-offs between concurrency and availability. In particular, we described how two-phase locking, multi-version timestamping, and hybrid synchronization techniques affect the availability of objects using quorum consensus protocols [Herlihy 86]. For example, we showed that hybrid schemes permit more quorum assignments than timestamping schemes , and thus could provide for higher availability in some instances. In related work, we also described new synchronization techniques. These techniques can use increased semantic knowledge to gain greater concurrency and can be used in conjunction with standard techniques such as two-phase locking.

We also enhanced our theoretical work on a replicated directory algorithm we had previously developed and implemented. This resulted in a better set of correctness proofs [Bloch et al. 87]. (See section 6.4.4.)

We concluded the design and implementation of a technique for demand-driven transfer of data across a network. This technique is useful for transferring program images intended for execution on another machine [Zayas 87].We performed the work on Accent but expect it to be portable to Mach.

We completed the development of a replication algorithm for storing log data, that is, data that describes the essential state transitions that occur on nodes of a distributed system. This algorithm permits recovery of nodes even after they are themselves physically destroyed [Daniels et al. 87]. We had previously done a simplified implementation of the algorithm in the TABS system, and we proceeded with its reimplementation and enhancement for use in Camelot. The reimplementation required the development of a more efficient communication protocol, and a complete implementation of the low-level storage structures. (See section 6.4.2.)

We began development of a new commit algorithm that coordinates multiple nodes that may be involved in committing a transaction. This algorithm has the reliability benefits of so-called Byzantine algorithms, with the many performance benefits of traditional algorithms. The reliability benefits are most useful in systems that are less tolerant while nodes or networks are repaired.

We began developing a new replication technique that should be practical for replicating a wide variety of abstract data types, such as queues, stacks, directories, sets, and typical database storage structures such as multiply indexed data sets.

We enhanced our previous TABS techniques for manipulating long-lived data. These techniques will be useful in Camelot and now support up to 2^{48} bytes of data; permit rapid streaming of data into memory; interact more efficiently with recovery mechanisms; require a reduced number of messages during normal processing; and help systems recover after node crashes.

The recovery algorithms that we developed support nested transactions with the ef-

ficiency usually associated with large, commercial database systems. In addition, the recovery algorithms permit efficient log usage for short transactions, the type that are most commonly used. We began the implementing these techniques in Camelot.

6.2. System development

Because we believe that researchers today need usable systems and languages on which to base their own work, we have twin goals in constructing Camelot and Avalon. First, we are constructing the Camelot and Avalon facilities so they can be used by others in the DARPA internet community. This requires balancing complexity, functionality, performance, and ease of implementation so we can produce working systems in a short period of time. Second, Camelot and Avalon must develop higher risk features, constructs, or algorithms in order to produce a system with sufficient flexibility and performance. Examples of areas in which Camelot and Avalon should be useful include the development of various types of database systems, command and control systems, messaging systems, and various near real-time control tasks.

This work on Camelot and Avalon was influenced by Accent, but more importantly, by TABS [Spector et al. 85b]. With TABS we demonstrated that general purpose distributed transactions are a valuable tool for structuring highly reliable distributed systems, and sufficiently efficient to be practical.

After a series of intensive group-wide sessions, we determined that a system like TABS was required, but that it should have more flexibility, higher performance, an easier to use interface, and relatively wide distribution. The latter required that the system run on the Berkeley 4.3 UNIX-compatible Mach operating system. We determined the other requirements were feasible given careful design and coding [Spector et al. 86a].

Thus, we functionally specified the Camelot distributed transaction facility and the Avalon programming language extensions. We specified Camelot as supporting nested transactions, very large user-defined objects, distributed operation, distributed logging, and compatibility with Mach. We specified Avalon as being built on standard programming languages, such as C++, Ada, and Common Lisp, and providing linguistic support for developing highly reliable distributed applications. These functional specifications were just a beginning, but they have led to large efforts [Spector et al. 86b, Herlihy and Wing 87a] and the current DARPA Reliable Distributed Systems Effort.

6.2.1. Camelot

The Camelot Project divided into two parts: Camelot implementation and Camelot testing/demonstration. Camelot implementors developed the low-level functions in the system including the disk manager, recovery manager, transaction coordinator, node server, node configuration application, communication manager, and Camelot library. Substantial time was devoted to evaluating the performance of the initial system func-

tions. Camelot testers developed applications to both test and demonstrate the utility of Camelot. These applications include an X-based, graphical room reservation system sufficient to support the CMU Campus; a document ordering system for use over the ARPAnet; a system to support the Department's cheese cooperative; and abstract data type libraries for shared recoverable hash tables and B-trees. These applications were chosen because they are representative of many problems in both industry and military applications, and because they will be widely used at Carnegie Mellon. A substantial amount of effort was put into documenting the system, resulting in the *Guide to the Camelot Distributed Transaction Facility Release 1.0.* Edition 0.4(23).

Camelot implements the synchronization, recovery, and communication mechanisms needed to support the execution of distributed transactions and the definition of shared abstract objects. It runs with the underlying support of the UNIX-compatible Mach operating system and its programming tools and provides flexible support for distributed transactions by accessing a wide variety of user-defined objects. Internally, Camelot uses many efficient algorithms to reduce the overhead of transaction execution including write-ahead logging, careful management of recoverable data, and highly tuned commit protocols. In June 1987 we had Camelot running on MicroVAXes, IBM RT PCs, and Multimaxes.

We completed the release of a system that is a sufficient base for our internal users: the Camelot test group and the Avalon project. We have also completed a release of our user's guide and the evaluation of our distributed logging package. As presented at SIGMOD '87, our logging package proved capable of supporting the execution of 70 transactions per second on a 2 MIPS IBM RT PC. We also evaluated the base Camelot system. The performance numbers show that the Camelot's overhead is not high and that it will not interfere with the intended uses of the system.

6.2.2. Avalon

We began a detailed design of the Avalon language in the first half of 1987. Avalon is a language interface for building reliable distributed applications on top of Camelot and Mach. Initially, it will be a set of extensions for C++, but there will also be extensions for use with Common Lisp and Ada. The Avalon effort has been divided into three parts:

- *Design:* We are tailoring our language extensions to maintain the spirit of each base language.
- *Implementation:* The Avalon runtime system exploits the Camelot transaction management facility.
- *Tool Support:* The Avalon/C++ preprocessor provides the interface to an Avalon programmer. The Avalon type library will contain a set of built-in types and user-extensible types. Avalon and Camelot share other version control and administrative tools.

We also began to design the first of the Avalon language extensions, which are to be made to the C++ language [Herlihy and Wing 87a]. In general terms, these extensions

permit programmers to implement permanent, synchronized abstract data objects with little or no more work than they use for traditional objects. The concepts are applicable to Common Lisp and Ada, and we intend to extend them for use in these systems.

The Avalon work necessarily lags behind the Camelot work since it depends on underlying Camelot facilities, and there are no completed portions of Avalon at this time. The Avalon group is aiming toward having the first Avalon demonstration by September 1987.

6.3. Accent

Many of the innovative ideas that were being incorporated in the Mach operating system come from our experience with Accent. For example, the Mach message system with its emphasis on copy-on-write data transfer is directly based on the Accent message system [Rashid 86]. Also, Mach's ultimate goal of producing a system comprising a relatively small operating system kernel with many servers derives from the structure of the Accent system. As a final example, the Mach Interface Generator(MIG) is based on the Matchmaker interface tool we previously wrote for Accent. (See Chapter 2.)

6.3.1. Systems evaluation

In 1986 Rashid and Fitzgerald concluded a long-term (multi-year) effort to evaluate the implementation of the Accent message passing mechanism. This message passing mechanism substantially reduces message passing costs by permitting the efficient transfer of very large messages. The technique Accent uses is called *copy-on-write data transfer.*

Our evaluation work, as detailed in [Fitzgerald 86], demonstrated that Accent substantially reduced the data copying costs associated with message passing while still retaining its other benefits. For example, after comparing Accent with the conventionally organized UNIX 4.1 BSD, we showed that Accent's file system performance is comparable, despite the fact that files were transferred using messages and copy-on-write mapping. This evaluation work substantially influenced the development of the Mach operating system [Accetta et al. 86], which is on-going work under the DARPA Strategic Computing Initiative.

The work that we performed involved developing a performance evaluation methodology for precisely measuring certain very short events in an operating system. This methodology and associated tools are also described in [Fitzgerald 86]; we expect the measurement techniques to be useful in future projects.

6.3.2. Operating system constructs

During the second half of 1985 we developed and evaluated operating system constructs that make it easier to write distributed programs—programs that must execute on multiple nodes of a distributed system. Such programs are growing more important because of needs for increased performance, reliability, and availability despite failures. They are also essential to solving parts of the distributed sensor network problem.

We completed the design of a message-passing mechanism, implementable on Accent, for doing demand-driven data transfer. In conventional message passing approaches, a network processing node that wishes to send a message dispatches the entire message at once, regardless of its size. The recipient/remote node must wait for the entire message to arrive before it can begin processing it. In a demand-driven approach, the sender node dispatches an "IOU" to the remote node. Then, as the remote node accesses portions of the message, it demands the rest of the message/data only as needed. This "lazy" transfer approach permits the recipient node to begin processing before all the data has arrived. In common situations where not all the data in a message will be accessed, this approach can substantially reduce system delays.

In other kernel work, we formalized and analyzed a mechanism for accessing objects in *recoverable storage*. Recoverable storage refers to the areas in virtual memory which contain objects that persist across program invocations and are not destroyed by processor, memory, or disk failures. With such recoverable storage, programmers can declare program variables normally and be assured the variables will always have their most recent values. Our implementation technique for recoverable storage is based on a generalization of the database technique of *write-ahead* logging. Like all logging techniques, write-ahead logging records state transitions of objects in a highly reliable log. The log is used to redo or undo changes following failures. However, unlike less efficient techniques, our write-ahead strategy permits most log writes to be done asynchronously; hence, they do not slow down the execution of application programs. We implemented a variant of this recoverable storage implementation technique as part of TABS [Spector et al. 85a]. This work lays the groundwork for a more flexible, production-quality implementation.

6.4. Reliability

We completed three major tasks in our work on reliability: We designed a distributed algorithm to efficiently record the state transitions of processing nodes across a network on remote *log servers*, we finished implementing a system whereby transaction-based applications can more effectively communicate with human users, and we concluded the performance evaluation of the TABS system.

6.4.1. Distributed transaction facility

We built the TABS distributed transaction facility to demonstrate that general purpose facilities that support distributed transactions are feasible to implement and useful in simplifying the construction of reliable distributed applications. Although there is room for diversity in its exact functions, a distributed transaction facility must make it easy to initiate and commit transactions, to call operations on objects from within transactions, and to implement abstract types that have correct synchronization and recovery properties.

To date, transactions have been useful in the restricted domain of commercial systems. Our research has been based on the notion that transactions provide properties that are essential to many other types of distributed applications. *Synchronization properties* guarantee that concurrent readers and writers of data do not interfere with each other. *Failure atomicity* simplifies the maintenance of invariants on data by ensuring that updates are not partially done despite the occurrence of failures. *Permanence* provides programmers the luxury of knowing that only catastrophic failures will corrupt or erase previously made updates.

Overall, the most important contributions of the TABS system were:

- The development of a system architecture in which transactions can be used for a wide variety of applications, including the maintenance of distributed and replicated data that is useful in distributed sensor networks.
- The integration of virtual memory management and recovery to provide very efficient, yet easy to use memory structures for programmers using TABS.
- An efficient set of communication mechanisms to track the nodes that are involved in a transaction and permit them to come to an agreement that the transaction has successfully completed.
- The first implementation of two recovery algorithms that provide decreased recovery time after system crashes and increased flexibility in using complex data types.

6.4.2. Distributed logging

This work is based on the following idea: If a processing node's fundamental state transitions are recorded on one or more other nodes, then even after a catastrophic failure, that processing node's state can be reconstructed by reapplying the state transitions, one after another. In other words, if a processing node N_1 is destroyed, there exists sufficient information on other nodes to allow N_1 to be reconstituted. We amplified this basic idea by developing the concept of a *replicated log service* [Daniels et al. 87]:

1. Logically, a replicated log service provides primitives to append new data records to the end of a logically infinite log. Depending on its precise configuration, the replicated log guarantees that these data records will be available in the future, despite a high number of failures.

2. A replicated log service is implemented through the use of a collection of networked, dedicated log servers. To write a log record to the replicated log service, the log records are written in parallel to some of the remote log servers using a special-purpose, high-performance protocol. To read a previously written record, it is necessary only to read the information on a single log server having up-to-date information. The algorithm makes it easy to learn which log servers are current and does some rather complex bookkeeping to handle all possible failure conditions.

The log service is tuned to support transaction-based systems but could be used wherever people want to record state transitions reliably. In some transaction-based environments, the use of a replicated log service could offer survival, operational, performance, and cost advantages. Survival is likely to be better for a replicated log service because it can tolerate the destruction of one or more entire processing nodes. Operation could be better because it is easier to manage high volumes of log data at a small number of logging nodes, rather than at all transaction processing nodes. Performance might be better because shared facilities can have faster hardware than could be afforded for each processing node. Finally, providing a shared network logging facility would be less costly than dedicating highly reliable storage (e.g., duplexed disks) to each processing node, particularly in some distributed systems environments.

6.4.3. Interaction with users

Reliability projects typically concentrate on maintaining the consistency of data internally stored at multiple processing nodes. Our work on user-interaction focused on how human users should interact with such systems. One major question is how to present *tentative*, *committed*, or *aborted* information to a user in the presence of unreliable communication and display equipment. Another is how to reexecute a user's commands so as to automatically retry transactions that have aborted.

We developed a user interaction system in which users can count on receiving correct information about the status of transactions they have initiated. Additionally, system output to the user is stable across crashes; that is, a display's output can be viewed as a type of database which has the same integrity guarantees as any other database. Even though there may be failures that temporarily delay a user from seeing his output, the failures can be repaired, and the user is guaranteed not to lose any information.

Human input, too, is stored reliably in a database. If a program reading from that database aborts due to a failure, the human input remains in the database and can be automatically reused when the work resumes.

In our prototype implementation of this, we stored both types of data in TABS data servers which are stored as reliably as any other system data. The system's performance shows the idea will work satisfactorily on machines that can deliver 2 or more MIPS. We gave our I/O subsystem an interface similar to the standard I/O system to make it easy for programmers to use.

6.4.4. Replicated directory demonstration

One way to ensure reliability is to replicate data—store it redundantly at multiple locations. Replicated data can enhance data availability in the presence of failures and increase the likelihood that data will be accessible when needed. Researchers at CMU have developed algorithms that exploit knowledge about the semantics of the replicated data to provide more effective replications than traditional approaches such as disk mirroring provide. In particular, Bloch, Daniels, and Spector developed an algorithm to replicate directory objects having operations on data such as *Lookup*, *Update*, *Insert*, and *Delete* [Bloch et al. 87].

We implemented this algorithm on top of TABS, both to demonstrate the correctness and performance of the algorithm, and also to demonstrate the completeness of TABS. We ran tests in which machines were turned off to show that data remains accessible despite failures. Then, the machines were turned on to show that they would automatically reconnect to the network after failures. This demonstration was one of the few demonstrations of replication algorithms that has actually been implemented.

6.5. Bibliography

[Accetta et al. 86] Accetta, M., R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Young.
Mach: a new kernel foundation for UNIX development.
In *Proceedings of Summer Usenix*, USENIX, July, 1986.

Mach is a multiprocessor operating system kernel and environment under development at Carnegie Mellon University. Mach provides a new foundation for UNIX development that spans networks of uniprocessors and multiprocessors. This paper describes Mach and the motivations that led to its design. Also described are some of the details of its implementation and current status.

[Bhandari et al. 87]
Bhandari, I.S., H.A. Simon, and D.P. Siewiorek.
Optimal diagnosis for causal chains.
Technical Report CMU-CS-87-151, Carnegie Mellon University Computer Science Department,
September, 1987.

Probe Selection (PS) is an important facet of any diagnostic program. The problem solved here is to find an optimal algorithm for PS in a causal chain. The word "optimal" has been used in the literature on diagnosis to designate both locally optimal and globally optimal algorithms. Locally optimal algorithms use some best-first technique to choose *next* the probe that optimizes some metric and they are not generally optimal, although they do provide good heuristics. Globally optimal algorithms choose that *sequence* of probes that optimizes some metric and they are truly optimal. In this work, optimal will be used to refer to the latter.

An optimal algorithm to do probe selection in causal chains is presented. Probes may have different costs of measurement. The algorithm runs in polynomial time.

[Bloch et al. 87] Bloch, J.J., D.S. Daniels, and A.Z. Spector.
A weighted voting algorithm for replicated directories.
JACM 34(4), October, 1987.
Also available as Technical Report CMU-CS-86-132.

Weighted voting is used as the basis for a replication technique for directories. This technique affords arbitrarily high data availability as well as high concurrency. Efficient algorithms are presented for all of the standard directory operations. A structural property of the replicated directory that permits the construction of an efficient algorithm for deletions is proven. Simulation results are presented and the system is modeled and analyzed. The analysis agrees well with the simulation, and the space and time perfor-

mance are shown to be good for all configurations of the system.

[Daniels et al. 87] Daniels, D.S., A.Z. Spector, and D.S. Thompson.
Distributed logging for transaction processing.
In *Sigmod '87 Proceedings*, ACM, May, 1987.
Also available as Technical Report CMU-CS-86-106.

Increased interest in using workstations and small processors for distributed transaction processing raises the question of how to implement the logs needed for transaction recovery. Although logs can be implemented with data written to duplexed disks on each processing node, this paper argues there are advantages if log data is written to multiple *logserver* nodes. A simple analysis of expected logging loads leads to the conclusion that a high performance, micro-processor based processing node can support a log server if it uses efficient communication protocols and low latency, non-volatile storage to buffer log data. The buffer is needed to reduce the processing time per log record and to increase throughput to the logging disk. An interface to the log servers using simple, robust, and efficient protocols is presented. Also described are the disk data structures that the log servers use. This paper concludes with a brief discussion of remaining design issues, the status of a prototype implementation, and plans for its completion.

[Detlefs et al. 87] Detlefs, D.L., M.P. Herlihy, and J.M. Wing.
Inheritance of synchronization and recovery properties in Avalon/C++.
In *Proceedings of Hawaii International Conference on System Sciences*, ACM, August, 1987.
Also available as Technical Report CMU-CS-87-133. Also published in Proceedings of OOPSLA 87, March 1987.

We exploit the inheritance mechanism of object-oriented languages in a new domain, fault-tolerant distributed systems. We use inheritance in Avalon/C++ to transmit properties, such as serializability and crash resilience, that are of specific interest in distributed applications. We present three base classes, RESILIENT, ATOMIC, and DYNAMIC, arranged in a linear hierarchy, and examples of derived classes whose objects guarantee desirable fault-tolerance properties.

[Durham 86] Durham, I.
Abstraction and the methodical development of fault-tolerant software.
PhD thesis, Carnegie Mellon University Computer Science Department, February, 1986.
Also available as Technical Report CMU-CS-86-112.

The reliable operation of software is a factor of increasing importance with the use of computers for critical functions. Software in general is demonstrably unreliable, particularly in the presence of external failures. Software that continues to provide reliable, if degraded, service in spite of external failures is termed *Fault-Tolerant*. Fault-tolerant software uses redundancy in code and data to recover from failures. Because few tools are available to guide the introduction of redundancy for the most cost-effective improvement in reliability, an *ad hoc* approach is commonly used. Unfortunately, such an approach cannot guarantee that the most serious potential failures have even been recognized. There is, therefore, a need for a methodical approach to deciding where to introduce redundancy. Abstraction has provided a foundation for the methodical development of correct software. As a conceptual tool, it simplifies the structure of software and supports both the precise specification of its behavior in the absence of failures and the ease of reasoning about it. This thesis provides a foundation for the methodical development of fault-tolerant software using abstraction as the basis for describing both failures and the behavior of software in the presence of those failures.

[Fitzgerald 86] Fitzgerald, R.P.
A performance evaluation of the integration of virtual memory management and interprocess communications in Accent.
PhD thesis, Carnegie Mellon University Computer Science Department, October, 1986.
Also available as Technical Report CMU-CS-86-158.

All communication-oriented operating systems need a way to transfer data between processes. The Accent network operating system addresses this need by integrating copy-on-write virtual memory management with inter-process communication. Accent provides a flat, 32-bit, sparsely allocatable, paged virtual address space to each process. It uses *mapping*, the manipulation of virtual memory data structures, to transfer large data objects between processes and to provide mapped access to files and other data objects. It uses *copy-on-write* protection to prevent accidental modification of shared data, so that mapping transfers data by value.

Although by-value data transfer and mapped file access have been considered desirable on methodological grounds, experience with previous systems such as RIG and CAL raised serious questions about the performance possible in such systems. This dissertation examine the impact of the Accent approach on the design, implementation, performance and use of Accent.

[Fitzgerald and Rashid 85]
Fitzgerald, R. and R.F. Rashid.
The integration of virtual memory management and interprocess communication in Accent.
Technical Report CMU-CS-85-164, Carnegie Mellon University Computer Science Department,
September, 1985.
The integration of virtual memory management and interprocess communication in the Accent network operating system kernel is examined. The design and implementation of the Accent memory management system is discussed and its performance, both on a series of message-oriented benchmarks and in normal operation, is analyzed in detail.

[Grizzaffi 85] Grizzaffi, A.M.
Fault-free performance validation of fault-tolerant multiprocessors.
Technical Report CMU-CS-86-127, Carnegie Mellon University Computer Science Department,
November, 1985.
By the 1990's, aircraft will employ complex computer systems to control flight-critical functions. Since computer failure would be life threatening, these systems should be experimentally validated before being given aircraft control.
Over the last decade, Carnegie Mellon University has developed a validation methodology for testing the fault-free performance of fault-tolerant computer systems. Although this methodology was developed to validate the Fault- Tolerant Multiprocessor (FTMP) at NASA-Langley's AIRLAB facility, it is claimed to be general enough to validate any ultrareliable computer system.
The goal of this research was to demonstrate the robustness of the validation methodology by its application on NASA's Software Implemented Fault-Tolerance (SIFT) Distributed System. Furthermore, the performance of two architecturally different multiprocessors could be compared by conducting identical baseline experiments.
From an analysis of the results, SIFT appears to have a better overall performance for instruction execution than FTMP. One conclusion that can be made is thus far the validation methodology has been proven general enough to apply to SIFT, and has produced results that were directly comparable to previous FTMP experiments.

[Herlihy 86] Herlihy, M.P.
Optimistic concurrency control for abstract data types.
In *Fifth ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, ACM SIGACT-SIGOPS, August, 1986.
A concurrency control technique is optimistic if it allows transactions to execute without synchronization, relying on

commit-time validation to ensure serializability. This paper describes several new techniques for objects in distributed systems, proves their correctness and optimality properties, and characterizes the circumstances under which each is likely to be useful. These techniques have the following novel aspects. First, unlike many methods that classify operations only as reads or writes, these techniques systematically exploit type-specific properties of objects to validate more interleavings. Necessary and sufficient validation conditions are derived directly from an object's data type specification. Second, these techniques are modular: they can be applied selectively on a per-object (or even per-operation) basis in conjunction with standard pessimistic techniques such as two-phase locking, permitting optimistic methods to be introduced exactly where they will be most effective. Third, when integrated with quorum-consensus replication, these techniques circumvent certain trade-offs between concurrency and availability imposed by comparable pessimistic techniques. Finally, the accuracy and efficiency of validation are further enhanced by some technical improvements: distributed validation is performed as a side-effect of the commit protocol, and validation takes into account the results of operations, accepting certain interleavings that would have produced delays in comparable pessimistic schemes.

[Herlihy 87] Herlihy,M.
Extending multiversion time-sharing protocols to exploit type information.
In *IEEE Transactions on Computers*, IEEE, April, 1987.
Atomic transactions are a widely accepted approach to implementing and reasoning about fault-tolerant distributed programs. This paper shows how multiversion time-stamping protocols for atomicity can be extended to induce fewer delays and restarts by exploiting semantic information about objects such as queues, directories, or counters. This technique relies on static preanalysis of conflicts between operations, and incurs no additional run-time overhead. This technique is deadlock-free, and it is applicable to objects of arbitrary type.

[Herlihy and Wing 86]
Herlihy, M.P. and J.M. Wing.
Avalon: language support for reliable distributed systems.
Technical Report CMU-CS-86-167, Carnegie Mellon University Computer Science Department,
November, 1986.
Avalon is a set of linguistic constructs designed to give programmers explicit control over transaction-based

processing of atomic objects for fault-tolerant applications. These constructs are to be implemented as extensions to familiar programming languages such as C++, Common Lisp, and Ada; they are tailored for each base language so the syntax and spirit of each language are maintained.

This paper presents an overview of the novel aspects of Avalon/C++: (1) support for testing transaction serialization orders at run-time, and (2) user-defined, but system-invoked, transaction commit and abort operations for atomic data objects. These capabilities provide programmers with the flexibility to exploit the semantics of applications to enhance efficiency, concurrency, and fault-tolerance.

[Herlihy and Wing 87a]
Herlihy, M.P. and J.M. Wing.
Avalon: language support for reliable distributed systems.
In *17th Symposium on Fault-Tolerant Computer Systems*, IEEE, July, 1987.
Also available as Technical Report CMU-CS-86-167.

Avalon is a set of linguistic constructs designed to give programmers explicit control over transaction-based processing of atomic objects for fault-tolerant applications. These constructs are to be implemented as extensions to familiar programming languages such as C++, Common Lisp, and Ada; they are tailored for each base language so the syntax and spirit of each language are maintained.

This paper presents an overview of the novel aspects of Avalon/C++: (1) support for testing transaction serialization orders at run-time, and (2) user-defined, but system-invoked, transaction commit and abort operations for atomic data objects. These capabilities provide programmers with the flexibility to exploit the semantics of applications to enhance efficiency, concurrency, and fault-tolerance.

[Herlihy and Wing 87b]
Herlihy,M.P., and J.M. Wing.
Axioms for concurrent objects.
In *Conference Record of the 14th Annual ACM Symposium on Principles of Programming Languages*, ACM, January, 1987.

Specification and verification techniques for abstract data types that have been successful for sequential programs can be extended in a natural way to provide the same benefits for concurrent programs. We propose an approach to specifying and verifying concurrent objects based on a novel correctness condition, which we call linearizability. Linearizability provides the illusion that each operation takes effect instantaneously at some point between its in-

vocation and its response, implying that the meaning of a concurrent object's operations can still be given by pre- and post- linearizability, and then give examples of how to reason about concurrent objects and verify their implementations based on their (sequential) axiomatic specifications.

[Herlihy and Wing 87c]
Herlihy, M.P. and J.M. Wing.
Reasoning about atomic objects.
Technical Report CMU-CS-87-176, Carnegie Mellon University Computer Science Department,
November, 1987.

Atomic transactions are a widely-accepted technique for organizing activities in reliable distributed systems. In most languages and systems based on transactions, atomicity is implemented through atomic objects, which are typed data objects that provide their own synchronization and recovery. This paper describes new linguistic mechanisms for constructing atomic objects from non-atomic components, and it formulates proof techniques that allow programmers to verify the correctness of such implementations.

[Herlihy et al. 87] Herlihy, M.P., N.A. Lynch, M. Merritt, and W.E. Weihl.
On the correctness of orphan elimination algorithms.
In *17th Symposium on Fault-Tolerant Computer Systems*, IEEE, July, 1987.
Abbreviated version of MIT/LCS/TM-329.

Nested transaction systems are being explored in a number of projects as a means for organizing computations in a distributed system. Like ordinary transactions, nested transactions provide a simple mechanism for coping with concurrency and failures. In addition, nested transactions extend the usual notion of transactions to permit concurrency within a single action and to provide a greater degree of fault-tolerance, by isolating a transaction from a failure of one of its descendants.

In a distributed system, however, various factors, including node crashes and network delays, can result in *orphaned* computations: computations that are still running but whose results are no longer needed. Even if a system is designed to prevent orphans from permanently affecting shared data, orphans are still undesirable, for two reasons. First, they waste resources. Second, they may see inconsistent information.

Several algorithms have been designed to detect and eliminate orphans before they can see inconsistent information. In this paper we give formal descriptions and correctness proofs for the two orphan elimination algorithms in [7] and

[10]. Our analysis covers only orphans resulting from aborts of actions that leave running descendants; we are currently working on modeling crashes and describing the algorithms that handle orphans that result from crashes. Our proofs are completely rigorous, yet quite simple. We show formally that the algorithms work in combination with any concurrency control protocol that ensures serializability of committed transactions, thus providing formal justification for the informal claims made by the algorithms' designers. Separating the orphan elimination algorithms from the concurrency control algorithms in this way contributes greatly to the simplicity of our results, and is in marked contrast to earlier work on similar problems.

[Jones and Rashid 87]
Jones, M.B. and R.F. Rashid.
Mach and Matchmaker: Kernel and language support for object-oriented distributed systems.
Technical Report CMU-CS-87-150, Carnegie Mellon University Computer Science Department,
September, 1987.
This paper also appeared in *Proceedings of the First Annual ACM Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA*, September, 1986.
Mach, a multiprocessor operating system kernel providing capability-based interprocess communication, and Matchmaker, a language for specifying and automating the generation of multi-lingual interprocess communication interfaces, are presented. Their usage together providing a heterogeneous, distributed, object-oriented programming environment is described. Performance and usage statistics are presented. Comparisons are made between the Mach/Matchmaker environment and other related systems. Possible future directions are examined.

[Jones et al. 85] Jones, M.B., R.F. Rashid, and M.R. Thompson.
Matchmaker: an interface specification language for distributed processing.
In *Proceedings of the 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1985.
Matchmaker, a language used to specify and automate the generation of interprocess communication interfaces, is presented. The process of and reasons for the evolution of Matchmaker are described. Performance and usage statistics are presented. Comparisons are made between Matchmaker and other related systems. Possible future directions are examined.

[Liskov et al. 86] Liskov, B., M. Herlihy, and L. Gilbert.
Limitations of synchronous communication with static process structure in languages for distributed computing.
In *13th ACM Symposium on Principles of Programming Languages*, ACM, January, 1986.
Also available as Technical Report CMU-CS-85-168.

Modules in a distributed program are active, communicating entities. A language for distributed programs must choose a set of communication primitives and a structure for processes. This paper examines one possible choice: synchronous communication primitives (such as rendezvous or remote procedure call) in combination with modules that encompass a fixed number of processes (such as Ada tasks or UNIX processes). An analysis of the concurrency requirements of distributed programs suggests that this combination imposes complex and indirect solution to common problems and thus is poorly suited for applications such as distributed programs in which concurrency is important. To provide adequate expressive power, a language for distributed programs should abandon either synchronous communication primitives or the static process structure.

[McKendry and Herlihy 86]
McKendry, M.S. and M. Herlihy.
Time-driven orphan elimination.
In *Proceedings of the Fifth Symposium on Reliability in Distributed Software and Database Systems*, IEEE, January, 1986.

An orphan in a transaction system is an activity executing on behalf of an aborted transaction. This paper proposes a new method for managing orphans created by crashes and by aborts. The method prevents orphans from observing inconsistent states, and ensures that orphans are detected and eliminated in a timely manner. A major advantage of this method is simplicity: it is easy to understand, to implement, and to prove correct. The method is based on timeouts using clocks local to each site. The method is fail-safe: although it performs best when clocks are closely synchronized and message delays are predictable, unsynchronized clocks and lost messages cannot produce inconsistencies or protect orphans from eventual elimination.

[Rashid 86] Rashid, R.F.
Threads of a new system.
Unix Review 4(8):37-49, 1986.

The Department of Defense, anxious for better multithreaded application support, has funded the development of Mach, a multiprocessor operating system for UNIX applications.

[Rashid et al. 87] Rashid, R.F., A. Tevanian, M. Young, D. Golub, R. Baron, D. Black, W. Bolosky, and J. Chew.
Machine-independent virtual memory management for paged uniprocessor and multiprocessor architectures.
Technical Report CMU-CS-87-140, Carnegie Mellon University Computer Science Department,
July, 1987.
This paper describes the design and implementation of virtual memory management within the CMU Mach Operating System and the experiences gained by the Mach kernel group in porting that system to a variety of architectures. As of this writing, Mach runs on more than half a dozen uniprocessors and multiprocessors including the VAX family of uniprocessors and multiprocessors, the IBM RT PC, the Sun 3, the Encore Multimax, the Sequent Balance 21000 and several experimental computers. Although these systems vary considerably in the kind of hardware support for memory management they provide, the machine-dependent portion of Mach virtual memory a consists of a single code module and its related header file. This separation of software memory management from hardware support has been accomplished without sacrificing system performance. In addition to improving portability, it makes possible a relatively unbiased examination of the pros and cons of various hardware memory management schemes, especially as they apply to the support of multiprocessors.

[Spector 85] Spector, A.Z.
The TABS project.
Database Engineering 8(2), 1985.
To simplify the construction of reliable, distributed programs, the TABS Project is performing research in the construction and use of general purpose, distributed transactions facilities. As part of this work, it has constructed a prototype and data objects that are built onto it. The goals of the work are to show that a distributed transaction facility can simplify programming some types of distributed applications and that its performance can be satisfactory.

[Spector 87] Spector, A.Z.
Distributed transaction processing and the Camelot system,
In Paker, Y. et al., *Distributed Operating Systems.* Springer-Verlag, 1987.
Also available as Technical Report CMU-CS-87-100.
This paper describes distributed transaction processing, a technique used for simplifying the construction of reliable distributed systems. After introducing transaction processing, the paper presents models describing the structure of dis-

tributed systems, the transactional computations on them, and the layered software architecture that supports those computations. The software architecture model contains five layers, including an intermediate layer that provides a common set of useful functions for supporting the highly reliable operation of system services, such as data management, file management, and mail. The functions of this layer can be realized in what is termed a *distributed transaction facility*. The paper then describes one such facility - Camelot. Camelot provides flexible and high performance commit supervision, disk management, and recovery mechanisms that are useful for implementing a wide class of abstract data types, including large databases. It runs on the Unix-compatible Mach operating system and uses the standard Arpanet IP communication protocols. Presently, Camelot runs on RT PC's and Vaxes, but it should also run on other computers including shared-memory multiprocessors.

[Spector and Daniels 85]
Spector, A.Z. and D.S. Daniels.
Performance evaluation of distributed transaction facilities.
Presented at the Workshop on High Performance Transaction Processing, Asilomar, September, 1985.

[Spector and Swedlow 87]
Spector, A.Z. and K.R. Swedlow, eds.
Guide to the Camelot distributed transaction facility: release 1 0.92(38) Edition, Carnegie Mellon University, 1987.
No abstract appeared with the paper.

[Spector et al. 85a]
Spector, A.Z., D.S. Daniels, D.J. Duchamp, J.L. Eppinger, and R. Pausch.
Distributed transactions for reliable systems.
In *Proceedings of the Tenth Symposium on Operating System Principles*, ACM, December, 1985.
Also available as Technical Report CMU-CS-85-117.
Facilities that support distributed transactions on user-defined types can be implemented efficiently and can simplify the construction of reliable distributed programs. To demonstrate these points, this paper describes a prototype transaction facility, called TABS, that supports objects, transparent communication, synchronization, recovery, and transaction management. Various objects that use the facilities of TABS are exemplified and the performance of the system is discussed in detail. The paper concludes that the prototype provides useful facilities, and that it would be feasible to build a high performance implementation based on its ideas.

[Spector et al. 85b]
Spector, A.Z., J. Butcher, D.S. Daniels, D.J. Duchamp, J.L. Eppinger, C.E. Fineman, A. Heddaya, and P.M. Schwarz.
Support for distributed transactions in the TABS prototype.
IEEE Transactions on Software Engineering SE-11(6), June, 1985.
Also available as Technical Report CMU-CS-84-132.
The Tabs prototype is an experimental facility that provides operating system-level support for distributed transactions that operate on shared abstract types. The facility is designed to simplify the construction of highly available and reliable distributed applications. This paper describes the TABS system model, the TABS prototype's structure, and certain aspects of its operation. The paper concludes with a discussion of the status of the project and a preliminary evaluation.

[Spector et al. 86a]
Spector, A.Z., D. Duchamp, J.L. Eppinger, S.G. Menees, and D.S. Thompson.
The Camelot interface specification.
Camelot Working Memo 2.
We present a performance evaluation methodology for general purpose distributed transaction processing facilities. This methodology extends previous techniques in two ways: First, it provides more insight into a transaction facility's internal operation and makes it possible to predict the effects of algorithmic and architectural changes. Second, it permits the performance of many types of transactions to be understood. We illustrate the methodology by applying it to a prototype transaction facility called TABS.

[Spector et al. 86b]
Spector, A.Z., J.J. Bloch, D.S. Daniels, R.P. Draves, D. Duchamp, J.L. Eppinger, S.G. Menees, and D.S. Thompson.
The Camelot project.
Database Engineering 9(4), December, 1986.
Also available as Technical Report CMU-CS-86-166.
This paper is an early discussion of Camelot. It broadly discusses its key functions and some aspects of its implementation. For perspective, it was written just after the project measured the performance of local write transactions without stable storage. A pretty early paper.

[Spector et al. 87] Spector, A.Z., D. Thompson, R.F. Pausch, J.L. Eppinger, D. Duchamp, R. Draves, D.S. Daniels, and J.J. Bloch.
Camelot: A flexible and efficient distributed transaction processing facility for Mach and the Internet.
In *IEEE Transactions on Computers, special issue on reliability*, IEEE, June, 1987.
Also available as Technical Report CMU-CS-87-129.

Camelot is a distributed transaction facility that runs on top of the Berkeley Unix 4.3 compatible Mach operating system. Camelot runs on a variety of different hardware and supports the execution of distributed transactions on shared user-defined objects, and hence the operation of distributed network services. The Camelot library, akin to the Unix Man 3 library, provides about 30 macros and procedure calls to simplify the development of applications and distributed services. To achieve good performance, Camelot is implemented using a combination of multithread tasks, shared memory, messages, and datagrams. This paper reports on a number of latency experiments to show the overhead of Camelot Release 0.4(22).

[Tevanian and Rashid 87]
Tevanian, A. Jr. and R.F. Rashid.
MACH: A basis for future UNIX development.
Technical Report CMU-CS-87-139, Carnegie Mellon University Computer Science Department,
June, 1987.
Computing in the future will be supported by distributed computing environments. These environments will consist of a wide range of hardware architectures in both the uniprocessor and multiprocessor domain. This paper discusses Mach, an operating system under development at Carnegie Mellon University, that has been designed with the intent to integrate both distributed and multiprocessor functionality. In addition, Mach provides the foundation upon which future Unix development may take place in these new environments.

[Tevanian et al. 87]
Tevanian, A. Jr., R.F. Rashid, D.B. Golub, D.L. Black, E. Cooper, and M.W. Young.
Mach threads and the Unix kernel: The battle for control.
Technical Report CMU-CS-87-149, Carnegie Mellon University Computer Science Department,
August, 1987.
This paper examines a kernel implementation lightweight process mechanism built for the Mach operating system. The pros and cons of such a mechanism are discussed along with the problems encountered during its implementation.

[Tygar and Wing 87]
Tygar,J.D., and J.M. Wing.
Visual specification of security constraints.
In *1987 Workshop on Visual Languages*, IEEE and the Swedish Defense Organization, February, 1987.
Also available as Technical Report CMU-CS-87-122.

We argue and demonstrate that the security domain naturally lends itself to pictorial representations of security constraints. Our formal model of security is based on an access matrix that traditionally has been used to indicate which users have access to which files, e.g., in operation systems. Our formal visual notation borrows from and extends Harel's statechart ideas, which are based on graphs and Venn diagrams. We present a tour of our visual language's salient features and give examples from the security domain to illustrate the expressiveness of our notation.

[Wendorf 87] Wendorf, J.W.
OS/application concurrency: A model.
Technical Report CMU-CS-87-153, Carnegie Mellon University Computer Science Department,
April, 1987.

A model of the processing performed on a computer system is presented. The model divides processing into two types: OS processing and application processing. It then defines what it means to have OS/application concurrency, and enumerates the different forms such concurrency can take. Examples are presented to illustrate the model's analytic and predictive capabilities. The model provides a common framework for describing the concurrency in different systems, and it aids in identifying the areas where increased concurrency may be possible. The potential performance improvements resulting from increased OS/application concurrency can also be predicted from the model.

[Wendorf and Tokuda 87]
J.W. Wendorf and H. Tokuda.
An interprocess communication processor: Exploiting OS/application concurrency.
Technical Report CMU-CS-87-152, Carnegie Mellon University Computer Science Department,
March, 1987.

The efficiency of the underlying interprocess communication facility is often one of the key determinants of the overall performance (and success) of a message-based operating system. Because of its importance, IPC has frequently been a target for extra hardware support, through the addition of special machine instructions or specialized IPC coprocessors. In this paper we propose and evaluate *software-level functional specialization* within a tightly-coupled multiple-processor system, as a "hardware" support technique for improving the performance of communicating processes. Our experiments, conducted on a VAX-11/784 shared-memory multiprocessor, show that per-

formance is significantly improved by the overlapped execution of IPC and application processing. We analyze under what conditions a software-specialized IPC processor will be effective, and we indicate how remote IPC support can be easily integrated with local IPC in our design.

[Wing 87] Wing, J.M.
A study of twelve specifications of the library problem.
Technical Report CMU-CS-87-142, Carnegie Mellon University Computer Science Department,
July, 1987.

Twelve workshop papers include an informal or formal specification of Kemmerer's library problem. The specifications range from being knowledge-based to logic-based to Prolog-based. Though the statement of the informal requirements is short and "simple," twelve different approaches led to twelve different specifications. All twelve, however, address many of the same ambiguities and incompletenesses, which we describe in detail, present in the library problem. We conclude that for a given set of informal requirements, injecting domain knowledge helps to add reality and complexity to it, and formal techniques help to identify its deficiencies and clarify its imprecisions.

The purpose of this paper is to summarize and compare the twelve different papers that address the same set of informal requirements--Kemmerer's library problem--as assigned to participants of the *Fourth International Workshop on Software Specification and Design* held in Monterey, California in April 1987.

[Zayas 87] Zayas, E.R.
The use of copy-on-reference in a process migration system .
Technical Report CMU-CS-87-121, Carnegie Mellon University Computer Science Department,
April, 1987.

Process migration is a valuable tool in a distributed programming environment. Two factors have conspired to discourage efficient implementations of this facility. First, it has been difficult to design systems that offer the necessary name and location transparency at a reasonable cost. Also, it is often prohibitively expensive to copy the large virtual address spaces found in modern processes to a new machine, given the narrow communication channels available in such systems.

This dissertation examines the use of *lazy* address space transfers when processes are migrated to new sites. An *IOU* for all or part of the process memory is transferred to the remote location. Individual memory pages are copied over the network in response to attempts by the transplanted

process to touch areas for which it holds an IOU. The SPICE environment developed at Carnegie Mellon has been augmented to provide a process migration facility that takes advantage of such a copy-on-reference scheme. The underlying Accent kernel's location-independent IPC mechanism is integrated with its virtual memory management to supply the necessary transparency and the ability to transmit data in a lazy fashion.

Study of the testbed system reveals that copy-on-reference address space transmission improves migration effectiveness (performance). Relocations occur up to a thousand times faster, with transfer times independent of process size. Since processes access a small portion of their memory in their lifetimes, the number of bytes transferred between machines drops by up to 96%. Message-handling costs are lowered by up to 94%, and are more evenly distributed across the remote execution. Without special tuning, faulting in a remote page took only 2.8 times longer on average than accessing a page on the local disk. Page prefetch and explicit transfer of resident pages are shown to be helpful in certain situations.

7. GRACEFUL INTERACTION

In the timesharing era, the great challenge was to use every machine cycle as efficiently as possible. In the new world of workstations, user time is increasingly expensive, while machine cycles are cheap and plentiful. The challenge now is to use these cycles to facilitate the flow of information between user and computer and promote effective, graceful man-machine interaction. This will make the computer users more comfortable and productive, and should thus be a help in both the development effort and usefulness of any major piece of software.

The target environment for our work in Graceful Interaction is a heterogeneous network of workstations and multiprocessors. We tailored our work specifically for large, heterogeneous computer systems because such systems have become predominant in the academic, military, and industrial environment, but not enough is known about providing effective and understandable user interfaces for them.

One of our goals was to facilitate the creation of a new generation of user interface systems by developing a prototype interface environment with some of the features we believed would be available in the future. We developed the notion of the Uniform Workstation Interface (UWI), i.e., a homogeneous interface system that integrated the results of previous research in man-machine communication, and formed the Dante project.

An important strategic decision for the Dante Project was to develop the Uniform Workstation Interface on top of the Mach operating system. This decision serves several purposes:

- The Mach operating system provides us with a large community of users (both at Carnegie Mellon and at external sites) and with a very diversified range of computer systems.
- The community of Mach users includes very experienced and sophisticated users and software developers, and thus represents an ideal testbed for our system. The Mach community will provide us with extensive feedback and thus contribute significantly to the ultimate success of the system.
- By taking advantage of the best possible development environment, we were able to minimize the effort we spent on implementation details and concentrate instead on the conceptual issues of designing user interface systems.
- We expect that the early availability of the UWI will in turn significantly enhance the appeal of Mach as an operating system and will contribute to its widespread acceptance.
- Finally, the Mach operating system gives us a powerful programming environment that supports efficient communication among local or remote processes, possibly written in different programming languages. Such an environment is the ideal layer upon which an interface system for a heterogeneous computing environment can be built.

We are developing the UWI as an interface manager written in Common Lisp. This is based on our experience with Lisp-based systems and on the clear trend towards computers that can run Lisp as efficiently as more traditional languages like C or Pascal. The tools we built to accompany it were intended to be as standard as possible so they would be accessible to as wide a community as possible.

7.1. Components of the UWI

7.1.1. The Lisp Shell

The Lisp Shell, which we started designing in the first half of 1986, is a Lisp-based command language interpreter that constitutes one of the interaction mechanisms in the UWI. The Lisp Shell is completely customizable, both through the customary user-level mechanisms and through an embedded programming language. This programming language, Common Lisp, makes the Lisp Shell an extremely powerful tool, allowing it to execute commands written as Lisp programs or as external processes written, for instance, in C or Pascal. This gives the user the full functionality of the UNIX programming environment without having to leave Lisp [Giuse 85].

During the second half of 1986, we implemented the first version of the Lisp Shell and ported it to an external environment, i.e. to the Warp project. A member of the Warp group did the porting which consisted of moving the Lisp Shell from CMU Common Lisp under Mach to Lucid Common Lisp under UNIX 4.2 (on a Sun workstation). The port also required converting the system's editor from Hemlock to Sun Emacs. The resulting system, called the Warp Shell, constitutes the top-level user interface to the Warp systolic multiprocessor. A first version of the Warp Shell was released to users outside the CMU community (see section 2.3 in the Distributed Processing Chapter).

Based partly on our experience with porting the Lisp Shell to the Warp environment, we recently redesigned the Lisp Shell. We added an operating system independent layer and clarified the boundary between the editor and the shell. This increased its portability and functionality. The increased functionality includes support for multiple command interpreters which can be active simultaneously, thus allowing application programs to define and use their own customized command languages as necessary.

7.1.2. The Viewers system

Inspired by the ZOG project and the Xerox Notecards project, we built Viewers, a frame-based interaction system that allows a user to navigate through a network of interconnected frames of information. The Viewers system represents both a browsing mechanism and a knowledge representation system. Knowledge is represented as frames of information that are connected into complex networks. The user can employ a simple, menu-like interaction to browse through the network and to perform actions such as running programs or manipulating system resources. We released the Viewers system to the Spice Lisp community on the Perq workstation and later completed a ver-

sion for the IBM RT PC, running under Mach. During the first half of 1987, we revised and extended the Viewers system. Now the interface can be customized and is dynamic; the application can change how Viewers interacts with the user as it runs.

7.1.3. Mirage

In the Spring of 1987, we implemented the graphical component of UWI called Mirage. Several of our tools are already using the meta-device for all their graphics input and output; moreover, we have made the system available as a stand-alone tool for developers of Lisp systems within the Mach environment [Busdiecker 86]. Mirage is written in Spice Lisp (CMU Common Lisp) and provides a powerful object-oriented programmable interface. This interface allows application programmers to develop complex applications more easily, since the system can be developed incrementally. One of the components of Mirage is a device- and window system-independent graphics layer, which provides complete insulation between the application program and the underlying hardware and software. The graphics layer is not itself a window manager, but rather uses whatever window manager is provided by the underlying machine. This first implementation uses the X window manager as the underlying window system, thus providing very powerful graphics capabilities on a large number of different machines. The object-oriented interface is built in Common Lisp and uses Portable Common Loops, a public-domain object system that is evolving into the standard object system for all Common Lisp implementations.

7.1.4. MetaMenu

We have completed the design of MetaMenu, a powerful menu-based system that will be another of the basic building blocks of the Uniform Workstation Interface. This system will be integrated with Mirage, the graphic component described above. We have started implementing MetaMenu, based on the X window manager.

7.1.5. Griffin

During the Spring of 1987, we completed the first implementation of Griffin, a new interface tool that implements a form-filling paradigm. This paradigm lets a user interact with a program through *forms*, which contain slots that can be individually manipulated by the user. Griffin is based on our past experience with the Cousin system and represents a considerable extension to that system, both in terms of power and in terms of performance. Griffin can be modified by the user, and it provides extensive type checking, a built-in help facility, and on-line documentation [Engelson 87].

7.2. Chinese Tutor

To test and validate some of the tools we have created so far, we developed the Chinese Tutor, an intelligent computer-based tutoring system for beginner-level

Chinese. The Chinese Tutor represents the first prototype of an application program that uses the Uniform Workstation Interface to allow the user to interact with the system through diverse styles of interaction. The application program itself is totally unaware of exactly how the user is interacting with it, which translates into a greatly simplified application program. The Chinese Tutor makes extensive use of the Viewers system and uses KR, described in section 7.3.1, to represent knowledge about the language. It also uses some of the preliminary work we have done on Mirage to provide graphical support for the display of Chinese characters. Work on the Chinese Tutor has provided us with invaluable feedback about the interactions among some of the subsystems we are developing, and we have already used some of this feedback to improve the design of those subsystems.

7.3. A knowledge-based system

Early form-based systems, including our own, were limited to a static model of the user and a single interface paradigm that forced the user to use a rigidly defined interaction style. To overcome the limitations, we began pursuing a different approach: By giving the interface system sufficient knowledge, we could enable it to make some decisions without explicit user intervention.

7.3.1. Representing knowledge

In the beginning of 1985 we analyzed what knowledge was required and how it could be represented. The required knowledge included a user model, i.e. a description of the user's expertise level, particular preferences, and goals, as well as a system model that described the environment and available resources and how to best utilize some of those resources. We chose to represent the knowledge using semantic networks and in 1986 constructed a simple semantic network prototype system. This prototype, called KR, provides very flexible knowledge representation without the performance overhead normally associated with full-blown semantic network systems. We designed it specifically for efficiency, since we felt that traditional frame-based systems would have a significant performance impact on a user interface environment like the one we are building. KR will constitute the central representation mechanism for the UWI; we have already converted the Lisp Shell to use it, and will convert the other system components to use it too. We have released the first version of KR to the CMU Common Lisp community as a stand-alone knowledge representation mechanism.

7.3.2. A knowledge-based interface

As one example of our knowledge-based interface strategy, we designed and implemented CoalSORT, a prototype intelligent interface to a large bibliographic database in the coal technology domain [Monarch 86]. The system's knowledge resides in a frame-based semantic network that represents a domain expert's knowledge, particularly in its organizational aspects.

Our CoalSORT research attacks consistency problems that plague more conventional document access strategies, specifically statistical and manual indexing. The system's network representation minimizes guesswork in the indexing task. Both users seeking information and those cataloging documents can browse through the concept net. Searchers and indexers thus select concept keywords from the same organization. Consistency between query and catalog views reflects the network's ability to represent the meaning underlying relevant terms.

By applying its understanding of the domain's conceptual structure, the system can guide and progressively restrict the search through a large document collection. Through menus and multiple window displays, CoalSORT cooperates with a user to formulate and refine partial concept descriptors. Descriptors express the information content match between a topical query and documents that the system knows about. Query formulation proceeds by recognition rather than recall or guesswork and the search employs a weighted key-concept mechanism. CoalSORT displays different kinds of information in separate windows and currently works with several terminal types, including Concepts and the DEC VT series, a Macintosh personal computer, and a Perq workstation. Relationships among network nodes appear as verbal diagrams and the system offers both global views and local context cues.

Our prototype system demonstrates the feasibility of combining a network knowledge representation with a browsing interface. Preliminary studies with engineers experienced in coal liquefaction technology proved encouraging. Users found the system relatively easy to learn and our network design adequately captured the meanings of key domain concepts. Our work opens the way for more powerful systems that can automatically parse queries and abstracts into a uniform semantic representation.

7.4. Bibliography

[Busdiecker 86] Busdiecker,R.
Mirage user's manual.
Dante Project internal working document.

[Engelson 87] Sean P. Engelson.
Griffin — a form-filling interface system.
Dante Project internal working document.

This document contains a description of Griffin, the Dante form-based user-interface system. Griffin is designed to work with clisp, and more specifically, with Hemlock, the meta-device, and the Dante knowledge structures.

Form-based communication between a user and an application is analogous to a person filling in a form for the purposes of conveying information to the readers of the form. The form provides slots, or fields, which either contain information to be communicated between the user and the application, or perform actions for the user. The user will fill in the fields with various values, which the application can then read and use, when the user signals events on action oriented fields.

[Giuse 85] Giuse, D.
Programming the Lisp Shell.

This document contains a description of how to program the Lisp Shell, both in terms of writing new Shell commands and in terms of creating programs that use the Shell as one of their resources to achieve a higher level of control.

The document is still under development, since the Lisp Shell itself is under very active development. At this stage, this document should not be interpreted as a cast-in-concrete specification yet, but rather as an overview of the current functionality.

[Giuse 87] Giuse, D.
KR: An efficient knowledge representation system.
Technical Report CMU-RI-TR-87-23, Carnegie Mellon University
Robitics Institutes,
October, 1987.

KR is a very efficient semantic network knowledge representation language implemented in Common Lisp. It provides basic mechanisms for knowledge representation which include user-defined inheritance, relations, and the usual repertoire of knowledge manipulation functions. The system is simple and compact and does not include some of the more complex functionality often found in other knowledge representation systems. Because of its simplicity, however, KR is highly optimized and offers good performance. These qualities make it suitable for many ap-

plications which require a mixture of good performance and flexible knowledge representation.

[Hayes et al. 85] Hayes, P.J., P.A. Szekely, and R.A. Lerner.
Design alternatives for user interface management systems based on experience with Cousin.
In *CHI '85 Proceedings*, April, 1985.

User interface management systems (UIMSs) provide user interfaces to application systems based on an abstract definition of the interface required. This approach can provide higher-quality interfaces at a lower construction cost. In this paper we consider three design choices for UIMSs which critically affect the quality of the user interfaces built with a UIMS, and the cost of constructing the interfaces. The choices are examined in terms of a general model of a UIMS. They concern the sharing of control between the UIMS and the application it provides interfaces to, the level of abstraction in the definition of the sequencing of the dialogue. For each choice, we argue for a specific alternative. We go on to present Cousin, a UIMS that provides graphical interfaces for a variety of applications based on highly abstracted interface definitions. Cousin's design corresponds to the alternative we argued for in two out of three cases, and partially satisfies the third. An interface developed through, and run by Cousin is described in some detail.

[Monarch 86] Monarch, I.
Abstract: intelligent information retrieval interfaces and a new configuration of text.
In *AAAI-86 Workshop on Intelligence in Interfaces*, AAAI, August, 1986.

The focus of this abstract is to summarize the approach taken to provide an intelligent interface for a bibliographic database. However, at the peripheries, the implications of this approach for a new conception of textual communication and a textual boundaries will be noted. Such an approach has these implications because an index at the back of a book is similar to the subject index in a library card catalogue. The linear organization of a book as specified by its table of contents is always capable of being supplemented by the multiple access points indicated by its index. The nonlinear reading and also writing of texts made possible by indexing can become a more central feature of textual communication in the emerging context of knowledge-based browsing interfaces to bibliographic databases.

8. VERY LARGE SCALE INTEGRATION

The ultimate goal of our research in VLSI is to make it practical to use VLSI routinely as a tool in the design of experimental computer systems. Our principal focus in the past three years has been constructing working systems that apply VLSI in several key areas. This approach allows us to evaluate algorithms and architectures that have attractive theoretical properties when actually applied to a particular problem and designed into circuitry. It can suggest new ideas for tools and in the meantime allow us to test and refine existing tools. It also induces innovations in the applications themselves. Our work during this period has been in three general areas: systolic building-blocks, design and testing tools, and VLSI systems and applications.

8.1. Systolic Building-blocks

8.1.1. Building Crossbar Switches

One of our goals is to develop methods that permit fabricating chips with good performance, even while working in a silicon-broker environment. The XBAR chip, with its simple structure, allowed us to concentrate on path optimization and gate sizing. The use of a second layer of aluminum provided a further challenge.

We have completed the layout of XBAR, a building block for implementing large crossbars. XBAR is a high-bandwidth, 16×16, two-bit crossbar chip implemented in double-metal, CMOS P-well technology with 3μm feature size. The chip is about 7.8×8.8 mm^2 and will be housed in a 144-pin grid array package. Speed has been the major goal and this explains the chip's large size despite its simple structure and functionality. Our target speed is a delay of less than 60 nanoseconds (data-in, data-out) on a 100pF load over the commercial temperature range. We have performed accurate SPICE simulations, achieving a 35 nanoseconds delay in the critical path (pads included, 100pF load).

8.1.2. Intermodule Communication

LINC is a custom chip whose function is to serve as an efficient link between system functional modules such as arithmetic units, register files, and I/O ports. In this respect, LINC is a "glue" chip for powerful system construction: It can provide efficient hardware support to connect high-speed, high-density building-block chips, provide physical communications and data buffering between functional system units, and implement some complicated data shuffling operations. During this contract period we have explored two applications that can use this processor array: the fast Fourier transform (FFT) and a simulated annealing algorithm for chip placement and wire-length minimization.

In June 1985 we completed LINC's layout, with General Electric's cooperation, and simulated its functionality on a Daisy workstation. GE simulated the low-level cells, verified timing of critical paths through the chip, and tested the fabricated chips using

test vectors designed and simulated at CMU. LINC will be demonstrated in the WarpJr systolic array being designed at CMU.

WarpJr is a 32-bit, floating-point, linear systolic processor of our design. Its functionality resembles that of the Warp processor, but it uses LINC to implement the data path and the AM29325 floating point chip as the ALU. To simplify programming the systolic array and to test the feasibility of exploiting LINC's pipelined registers efficiently, we wrote a compiler specifically for WarpJr. The high-level language is W2, just as in the Warp compiler, and its front end is also the same as the Warp compiler's.

8.2. Tools for VLSI Design and Testing

8.2.1. Yield Simulation

With partial support from the Semiconductor Research Corporation, we have built a catastrophic-fault yield simulator for integrated circuits. Our VLASIC simulator employs a Monte Carlo strategy and statistical defect models to hypothesize random catastrophic point defects on a chip layout and then determine what circuit faults, if any, have occurred. The defect models are described in tables, and therefore readily extend to new processes or defect types. The defect statistical model is based on actual fabrication line data and has not appeared before in the literature. The circuit fault information generated by VLASIC can be used to predict yield, optimize design rules, generate test vectors, evaluate redundancy, etc. We are extending VLASIC to handle larger designs and to improve its redundancy analysis system.

The process tables assume a single-level metal NMOS technique. The simulator takes as inputs a circuit and wafer layout, and defect statistics. The simulator uses a Monte Carlo method to generate potentially-faulty circuit instances with the correct statistical distribution. A back end is currently being implemented for use in fault-tolerant circuit design. We are currently fitting the process models to real data from a two-micron double-metal NMOS process. We will then use the simulator to predict the yield of a 16K SRAM containing 100,000 transistors, and compare our results with actual fabrication results.

8.2.2. Testing by Simulation

As VLSI circuit sizes have increased, researchers have found that testing manufactured circuits for defects proves at least as challenging as designing and manufacturing them. Simulation offers one fundamental means of debugging and gaining confidence in designs, and research at CMU is exploring several techniques that promise significant advances in fault simulation and test generation. Our work includes both fault simulators and test data pattern generators for MOS circuits.

Several aspects of CMU's research program stand out among related efforts in both industry and academia. First, much of our work is based on *switch-level* models, an abstract representation of MOS circuits in which each transistor is modeled as a parameterized switch. The switch-level representation of a system can capture many aspects of circuit behavior that more traditional gate-level models cannot, while providing a level of abstraction that allows efficient validation of very large circuits operating over long input sequences. Second, our work applies *symbolic* methods, in which an abstract representation of a circuit is created, describing its behavior for many possible input and timing conditions. Symbolic methods are used both to enhance the performance of conventional simulators, as well as to provide new capabilities in circuit verification and automatic test pattern generation. Finally, our work emphasizes an *algorithmic* approach that spans the entire range from theory to practice. On the theoretical side, we explore and develop new algorithms and verification methodologies. On the practical side, we implement production quality programs for a variety of validation tasks. These programs receive widespread use by VLSI designers nationwide.

Switch-level Models

A switch-level simulator models a logic circuit as a network of nodes connected by transistors. This allows a detailed modeling of many of the phenomena associated with MOS circuits such as bidirectional transistors, dynamic memory, precharged logic, and various bus structures. In contrast, simulators based on more traditional gate-level models cannot model these features accurately and will often fail to detect major design errors. Unlike detailed circuit level simulators, however, switch-level simulators abstract away many details of the circuit electronics in the interest of performance. Node voltages are represented by three states 0, 1, and X (for uninitialized or invalid), and transistors are modeled as discrete switches. Consequently, switch-level simulators can simulate circuits containing 100,000 or more transistors for thousands of clock cycles in a timely fashion.

Symbolic Methods

A symbolic simulator computes a circuit's behavior for a sequence of user-supplied input patterns. The user can then interactively examine and manipulate the computed symbolic representations to gain insight into circuit operation. Alternatively, a user might attempt to prove circuit correctness by testing the representation for equivalence with another derived from the circuit specification. Such a program opens up a totally new way for designers to analyze circuits. A conventional simulator forces the designer to try out a small number of test cases one at a time. Serious design errors often remain undetected. Once improper behavior has been detected, the designer must engage in the laborious process of hypothesizing the source of the error, devising test patterns to validate the hypothesis, and simulating these patterns. In contrast, a symbolic simulator helps the user understand how the circuit processes arbitrary Boolean data. A single simulation sequence determines the circuit behavior for many test cases. Using the symbolic manipulation capabilities of the program, the user can examine the function at different points in the circuit to more quickly identify the source of an error.

In a conventional simulator, the description of a system is read in, the user specifies a series of input patterns, and the simulator computes the behavior of the system for

these patterns. In a symbolic simulator, however, the input patterns consist of Boolean variables in addition to the constants 0 and 1, and the behavior computed by the simulator consists of Boolean functions over the present and past input variables. These functions can then be tested for equivalence with functions representing the desired behavior of the system, thereby verifying the correctness of the system for the set of all possible input data represented by the input patterns. Although a complete symbolic verification of a large circuit may be impractical, the user can adopt a hybrid approach with some inputs set to variables and others to constants.

Our MOSSYM simulator simulates MOS circuits represented as switch-level networks [Bryant.85.Symbolic]. MOSSYM implements the same circuit model as the our previous simulator MOSSIM II and can accurately model such MOS circuit structures as ratioed, complementary and precharged logic, dynamic storage, bidirectional pass transistors, and busses. Experimental results with MOSSYM are quite promising [Bryant&.85.Performance] . The availability of a symbolic simulator raises a new set of problems on how to rigorously verify a circuit based on observations of its input-output behavior. This task is related to the "machine identification" problem of classical finite state machine theory, but with some new twists that allow more positive results. (For related discussion, see section 8.2.5.)

8.2.3. A Compiled Simulator for MOS Circuits

The COSMOS project, begun in early 1986, addresses the issue of algorithmic improvements to switch-level simulation [Bryant.86.Collection]. COSMOS, a compiled simulator for MOS circuits, will replace MOSSIM II as the leading switch-level simulator at CMU. It will operate approximately an order of magnitude faster than MOSSIM II while providing additional capabilities including fault simulation. Furthermore, it can easily be implemented on any special purpose simulation accelerator that supports Boolean evaluation. Features of COSMOS include both logic and concurrent fault simulation, mechanisms to interface user-written C code to implement new simulation commands as well as behavioral models, and the ability to simulate up to 32 sets of data simultaneously. Programs are provided to translate circuit descriptions produced by the Berkeley Magic circuit extractor into the network format required by the symbolic analyzer.

Unlike switch-level simulators that operate directly on the transistor level description during simulation, COSMOS transforms the transistor network into a Boolean description during a preprocessing step. This Boolean description, produced by a symbolic analyzer, captures all aspects of switch-level networks including bidirectional transistors, stored charge, different signal strengths, and indeterminate (X) logic values. Most significantly, for all but a small class of dense, pass transistor networks (e.g. barrel shifters), the size of the Boolean description grows linearly with the size of the transistor network. Even for these exceptional cases, the growth is no worse than quadratic. This compares favorably to the exponentially sized results produced by all previous symbolic analyzers.

For execution on a general purpose computer, COSMOS generates a procedure for each channel-connected subnetwork. When called, this procedure computes the new states of the subnetwork nodes as a function of the initial transistor and node states. The Boolean description produced by the symbolic analyzer maps directly into machine-level logical instructions, and hence execution time is very short. COSMOS compiles these sub-network procedures together with an event-driven scheduler and user interface code. The resulting program appears to the user much like an ordinary simulator, except that the network is already loaded at the start of execution. This program is being written in C and will be made available to the DARPA VLSI community.

For execution on special purpose hardware, COSMOS maps the output of the symbolic analyzer into a set of Boolean elements. Whatever methods are provided to support logic gate simulation are then used to perform switch-level simulation. COSMOS requires no special hardware to support switch-level simulation. In fact, many of the costly features found in existing simulation accelerators such as bidirectional elements and multi-valued logic modeling are not needed. Preprocessors such as ours encourage a RISC approach to hardware design where only basic operations on a limited set of data types are implemented. The preprocessor must perform the mapping between the complex models required by the simulator and the simple operations implemented by the hardware. For switch-level simulation, the payoff in terms of greater flexibility and performance clearly favors this approach.

COSMOS provides a combination of high simulation performance and a variety of simulation features. It simulates between 10 and 200 times faster than other switch-level simulators such as MOSSIM II. COSMOS achieves this performance by preprocessing the transistor network using a symbolic Boolean analyzer, converting the Boolean description into procedures describing the behavior of subnetworks plus data structures describing their interconnections, and then compiling this code into an executable simulation program.

An earlier bottleneck caused by the long time required to preprocess a circuit into an executable simulation program has been solved by a combination of hierarchy extraction, incremental analysis, and assembly code generation. The preprocessor takes a flat network description and extracts a two-level hierarchy consisting of transistor subnetworks as leaves, and their interconnection as root. This extraction utilizes graph coloring/isomorphism-testing techniques similar to those used by wirelist comparison programs. To avoid ever repeating the processing of isomorphic subnetworks, it maintains a directory of subnetworks and their compiled code descriptions with file names derived from a hash signature of the transistor topology. Finally, the code generation program can generate assembly language declarations of the data structures rather than C code. The data structure formats for all UNIX assemblers are sufficiently similar that the assembly code generator for a new machine type can be produced with minimal effort. As an example, a 1600 transistor circuit that earlier required 23 minutes to preprocess on a VAX-11/780 now requires only 2.9 minutes to preprocess the first time, and only 2.3 minutes subsequently.

8.2.4. System Design Tools

Standard Frame Configuration

When planning chip fabrication via MOSIS, designers must conform to the MOSIS Standard Frame I/O pad conventions. To facilitate placing and connecting input/output pads around the edges of a chip design, we have written a useful tool that we call a "standard frame instantiator". The user gives our Frame program a pad frame description identifying the parameters of the frame itself, characteristics of the pad cells used, the sorts of wires connecting the frame's cells, and which particular pad cell goes at each pin. The program provides a completely laid-out frame, including the "glue" between pad cells. It takes only minutes to generate an error-free pad frame. We are currently extending Frame to include labels in the generated frame cells in such a way that Magic's router can automatically route the frame to the chip's internals. We have already used the tool to generate frames for two chips.

Asynchronous Building Blocks

We have designed a chip that contains a collection of asynchronous circuit building blocks that are unavailable as standard IC parts. Without these parts, building real asynchronous circuits is extremely difficult. The chip has eight different configurations, each offering a different set of asynchronous parts to the user. The parts include Muller C-elements, two and four input transition call modules, two and four way select modules, transition toggles, asynchronous arbiters, asynchronous FIFO, and Q-Flops (for building internally clocked delay-insensitive modules). With only a few exceptions, the parts use a two-phase transition-sensitive communication protocol.

Our Parts-R-Us chip for building self-timed circuits returned from fabrication in the fall of 1986 [Brunvand.87.Parts-R-Us]. Thirteen chips were delivered and, although yield was quite low, we found at least one functional example of every design element. Two chips were completely working and three were completely non-functional, indicating that the yield problem was probably the fabricator's and not due to the design. We have developed test software that allows the same Common Lisp program that was used to drive a design's simulation to be used to test the completed chip. This allows easy comparison of the simulated behavior to that of the chip, and also means that test programs need only be written once. After testing, we built two small demonstration circuits in the Spring, revised the design slightly to permit building more complex circuits, and returned the chip to MOSIS for refabrication.

Our next step will be to use the Parts-R-Us building block designs to construct a silicon compiler for automatically generating asynchronous circuits from programs. The compiler will map algorithms to logic designs and then apply a series of optimizing transformations to derive efficient hardware realizations. We have already begun work on tools and methods that will take behavioral descriptions written in Occam and translate them into self-timed circuits. Occam is a CSP-like language used for describing concurrent communicating objects. It turns out to provide a very natural medium for describing one class of self-timed circuits.

An Asynchronous Multiplier

Rockoff completed the layout of an asynchronous multiplier with a recursive symbol structure. This chip project has been engineered for delays using Sutherland's "logical-effort" delay model. Switch-level simulation has been performed on the entire chip. Additionally, the layout has been verified including boundary circuitry such as pads. Included on the chip is a set of test structures that will allow us to characterize both the speeds of devices fabricated on the die as well as the DC transfer effects of various gates' logical efforts.

8.2.5. Formal Verification by Simulation

We have also investigated methods for formally verifying digital circuits by logic simulation [Bryant.86.Can]. A simulator can prove correctness if it can be shown that only circuits implementing the system specification will produce a particular response to a sequence of simulation commands. This style of verification has advantages over other proof methods in being readily automated and requiring less attention to the low-level details of the design, as well as advantages over other approaches to simulation in providing more reliable results, often at a lower cost.

Our work has explored two verification methods: The first, called "black-box" simulation, involves simply observing the output produced by the simulator in response to a sequence of inputs with no consideration of the internal circuit structure. In contrast to the machine identification problem of classical sequential systems theory, however, the simulator is assumed capable of modeling a signal as having an unknown or X value, where the simulator must produce responses that are monotonic for the partial ordering $X < 0$ and $X < 1$. In addition to supplying input sequences, the user can command the simulator to set all internal signals to X. It has been shown that a circuit can be verified by black-box simulation if and only if the specified behavior is that of a *definite* system, i.e. the output of the system at any time depends only on the most recent k inputs for some constant k. The second method, called "transition" simulation, requires the user to specify the relation between states in the circuit and the specification. The simulator is then used to prove that each state transition in the specification is implemented correctly. Arbitrary systems can be verified by this method, but the simulation sequences depend on the implementation.

In general the circuit verification problem is NP-hard. However, in some cases the ability of the simulator to model unknown signals can be exploited to reduce the number of patterns simulated. A variety of memory circuits can be verified by simulation sequences that are linear or near-linear in the memory size.

State-transition verification has been applied to two circuits, the dynamic stack in the Mead-Conway text, and various sizes of a 3-transistor dynamic RAM. The number of simulation steps necessary to verify a n-bit RAM is asymptotically proportional to n log n. However, as each simulation step involves simulation of the entire circuit, the duration of a simulation step also increases with the size of the circuit; thus, the time required to verify an n-bit RAM is more nearly proportional to n**2 log n. Measurements bear this

out; for example, verifying a 256-bit RAM on a lightly loaded VAX using MOSSIM II as the logic simulator took 8 hours. Though these measurements show that state-transition verification of large circuits is not yet feasible, a symbolic simulator such as MOSSYM should significantly reduce the number of simulation steps required. As MOSSYM matures, state-transition verification should become more promising.

8.2.6. Formal Hardware Description Semantics

Another research focus centers around applying concepts from the domain of programming systems to hardware design. We are particularly interested in the problem of assuring the correctness of hardware implementations with respect to specifications. We are addressing this problem in two ways: verification and compilation.

We have been developing temporal logic based tools for specifying and verifying sequential machines. Recently, we have used our state machine verifier to debug the design for a complicated DMA controller. This was the most ambitious verification effort that we have attempted so far. We have also developed a prototype version of an automatic verifier for asynchronous circuits. Circuit specifications are given as finite state machines and a very flexible timing model for circuit behavior is supported. In addition, the system makes it possible to verify a circuit in parts, following the modular structure of the circuit. Finally, we have also developed techniques for analyzing sequential circuits that are composed of many identical processing elements. These techniques permit a reduced version of the circuit with one or two processing elements to be considered instead of the entire circuit. The "state explosion" phenomenon (See section 8.2.7), which had previously hindered the analysis of such circuits, is avoided by these techniques.

Verification methods can be used to check whether an existing design satisfies a specification. We have concentrated on *automatic* techniques for verifying sequential circuits. This work has proceeded in two directions: verifying *asynchronous* sequential circuits described at the gate level, and verifying *synchronous* sequential circuits described by a program in a high-level hardware description language (SML).

In both asynchronous and synchronous cases: the circuit is specified by formulas in temporal logic, a state-graph describing the possible circuit behaviors is automatically extracted from the circuit description, and then the formulas are automatically compared with the state graph. The last step is performed by a program called the *model-checker*. If the state graph (and hence the circuit) satisfies the specification, the model-checker reports that all is well. If the circuit fails to meet the specification, the model-checker tries to produce an example of an execution (sequence of circuit actions) which can be used to find and correct the problem.

Another way to guarantee the correctness of a hardware implementation is to compile it directly from its specification. We are doing this in two different ways. First, we have developed means for compiling *path expressions* into asynchronous controllers. Path expressions were originally devised as a way to specify the desirable interactions

among "loosely-coupled" systems of concurrent programs. We believe that they are also a good way to specify the interactions among loosely-coupled hardware devices. Our technique isolates the concurrency control in a separate circuit which interacts with the circuits it controls via request/acknowledge signals. The controller circuit allows the controlled circuits to execute in only those orderings allowed by the path expression.

Second, an SML program can be regarded as a specification of a correct circuit once it has been verified. We can translate SML programs into a form suitable for input to existing programs that translate state machines into hardware. For example, it is possible to write a high-level program for a device controller, check that it is correct with respect to a specification in temporal logic, and then translate it automatically into various hardware implementations.

Programming Language Issues

We have continued experimenting with our SML language, using it to describe a commercial UART interface. We were able to verify some temporal properties of our description for this common and nontrivial device. We also produced a description of SML's syntax and semantics and illustrated its usefulness in designing small finite state machines [Browne&.85.SML].

Although most formal models of asynchronous circuits avoid making assumptions about the relative speeds of the circuit components, many practical designs rely on such assumptions for their correct operation. Therefore we extended the theory and implementation of our asynchronous verifier to include simple timing assumptions.

We have also begun to investigate the formal foundations of asynchronous circuit operation as a basis for more powerful methods of verifying and synthesizing them. "Fairness" and "liveness" properties play critical roles in concurrent systems. An example of a fairness property is "if any user requests a resource, he will eventually be granted it"; a liveness property might be "a circuit always acknowledges a request". Recently, others have proposed trace theory as an appropriate formal semantics theory for asynchronous circuits. Previously, trace theory had only considered finite circuit executions, and describing fair or live behavior requires infinite executions in general. We extended trace theory to include infinite executions, and showed that it is possible to describe a fair, delay-insensitive arbiter (a circuit whose existence previous researchers had questioned) [Black.85.Existence].

Finally, many apparently complex circuits have a fairly simple recursive structure. We have implemented a system, Escher, that allows the user to describe the recursive structure using a graphical interface. The circuit structure is expanded when some parameters are provided (such as how many bits wide it should be). The result is a layout of the primitive cells and their connecting wires. We have applied the system to a variety of examples, including a sorting network, FFT, and a recursive hardware multiplier [Clarke&.85.Escher].

8.2.7. Automatic Hardware Verification

Many hardware systems can be viewed at some level of abstraction as communicating finite state machines. The dream of somehow using this observation to automate the verification of such programs can be traced all the way back to the early papers on Petri nets in the 1960's. The temporal logic *model checking procedure* also attempts to exploit this observation. The model-checking algorithm determines whether the global state transition graph associated with some concurrent program satisfies a formula in the *CTL* temporal logic. The algorithm is linear in both the size of the global state graph and the length of the specification. Researchers have successfully used it to find subtle errors in many sequential circuit designs. Several other researchers have extended the basic model checking algorithm or proposed alternative algorithms. Although these algorithms differ significantly in the type of logic they use and in the way they handle issues like fairness, they all suffer from one apparently unavoidable problem: In analyzing a system of *N* processes, the number of states in the global state graph may grow exponentially with *N*. We call this problem the *state explosion problem*. Our approach to this problem is based on another observation about distributed programs. Although a given program may involve a large number of processes, it is usually possible to partition the processes into a small number of classes so that all of the processes in a given class are essentially identical. Thus, by devising techniques for automatically reasoning about systems with many identical processes, it may be possible to make significant progress on the general problem.

We have devised a means of reducing the problem of checking the correctness of large networks of identical finite state machines to the problem of checking very small networks. This allows us to apply our automatic verification tools to large systems with vast numbers of states.

To understand how our method works, consider a distributed mutual exclusion algorithm for processes arranged in a token ring network. How can we determine that such a system of processes is correct? Our first attempt might be to consider a reduced system with one or two processes. If we can show that the reduced system is correct and if the individual processes are really identical, then we are tempted to conclude that the entire system will be correct. In fact, this type of informal argument is used quite frequently by designers in constructing systems that contain large numbers of identical processing elements. Of course, it is easy to contrive an example in which some pathological behavior only occurs when, say, 100 processes are connected together. By examining a system with only one or two processes it might even be quite difficult to determine that this behavior is possible. Nevertheless, one has the feeling that in many cases this kind of intuitive reasoning does lead to correct results. The question that we address is whether it is possible to provide a solid theoretical basis that will prevent fallacious conclusions in arguments of this type.

We have addressed the problem of devising an appropriate logic for reasoning about networks with many identical processes. The logic that we propose is based on computation trees and is called *Indexed Temporal Logic*. Typical operators include

GLOBALLYf, which will hold in a state provided that f holds globally along all computation paths starting from that state and *INEVITABLY*f, which will hold in a state provided that f eventually holds along all computation paths. In addition, our logic permits formulas of the form $\bigwedge_i f(i)$ and $\bigvee_i f(i)$ where $f(i)$ is a formula of our logic. Intuitively, the formula $\bigwedge_i f(i)$ will be true in a global state of some concurrent system, provided that the formula $f(i)$ holds for each component process i. $\bigvee_i f(i)$ is explained similarly.

Since a closed formula of our logic cannot contain any atomic propositions with constant index values, it is impossible to refer to a specific process by writing such a formula. Hence, changing the number of processes in a family of identical processes should not affect the truth of a formula in our logic. We make this intuitive idea precise by introducing a new notion of equivalence between networks of finite state processes. We prove that if two systems of processes correspond in this manner, a closed formula of our logic will be true in the initial state of one if and only if it is true in the initial state of the other. We have devised a procedure that can be used in practice to find a network with a small number of processes that is equivalent to a much larger network with many identical processes. We call this result the *collapsing theorem for networks with many identical processes.*

To see how the collapsing theorem might be used, consider the distributed mutual exclusion algorithm discussed above. We assume that the atomic proposition c_i is true when the i-th process is in its critical region, and that the atomic proposition d_i is true when the i-th process is delayed waiting to enter its critical region. A typical requirement for such a system is that a process waiting to enter its critical region will eventually enter the critical region. This condition is easily expressed in our logic by the formula

$$\bigwedge_i GLOBALLY(d_i \supset INEVITABLY\, c_i).$$

By using our results it is possible to show that exactly the same formulas of our logic hold in a network with 1000 processes as hold in a network with two processes! We can use one of the temporal logic model checking algorithms to automatically check that the above formula holds in networks of size two and conclude that it will also hold in networks of size 1000.

8.3. VLSI Systems and Applications

8.3.1. A Scan Line Array Processor

The SLAP project is developing a highly parallel (100-1000 processor) SIMD linear array architecture for image computation and similar applications [Fisher.86.Scan]. The scan line array processor (SLAP) architecture devotes a processor to each pixel of an image scan line. Processing elements are connected in a linear array by both nearest-neighbor bidirectional paths and a specialized video-rate shift register, and are con-

trolled in SIMD fashion. This architecture has a number of advantages for VLSI implementation, and appears to be well-suited to a wide variety of image processing tasks. We also expect it to be of use in graphics and specialized numeric processing, and preliminary investigation suggests that it can also be fruitfully applied to the simulation of connectionist architectures for artificial intelligence. We estimate that a 512 processor SLAP can be built from 128 MOSIS 3 micron CMOS chips, and provide 2 billion operations per second using 8 bit pixels and 16-20 bit results.

Most recently, our work has concentrated on hardware implementation and on the development of programming paradigms and tools. The SLAP PE datapath layout has been completed, and a test chip has been submitted for fabrication in 3 micron CMOS. We have used a conservative design style that should allow us to get useful chips from both N well and P well runs. In anticipation of the availability of MOSIS standard frames with higher pincounts, we are redesigning the SLAP interprocessor communication path to handle a full word in a single cycle, yielding improvements in latency, ease of programming, and system construction. We expect to submit a 4-PE layout for fabrication in 2 micron CMOS near the end of 1987.

We are also engaged in the design of a prototype machine. We expect a 512-processor machine to include a one-board microprogrammed global controller and two or three array boards. We plan to implement the controller as a triple height VME wirewrap board, and the array boards as triple height VME PC boards. The controller will be interfaced to a Sun-3 host, while the array boards, using the VME cage only for mechanical support and power, will receive data over point to point cables from commercial image capture and storage components. Our schedule calls for the prototype to become operational in the summer of 1988. When fully populated with 128 SLAP chips, the system should yield a peak throughput of some four billion 20 bit operations per second.

Our goal has been to refine the architecture of the 20-bit processing element for efficiency and ease of code generation, while laying the groundwork for the chip layout.

On the architecture front, we have settled on a two-bus datapath with separate ALU (with support for 40 bit Booth multiplication and division) and barrel shifter, specialized circuitry for a video shift register and neighbor communication, and an unusual two-port register file. Instructions are pipelined three cycles deep, with a simple fixed layout and timing scheme. Supporting flexible programming within the SIMD paradigm are a novel context nesting mechanism, local addressing of the register files, and support for global voting and fast combinational signal propagation.

Now that the hardware effort is well underway, we have also begun to work on programming support. We have designed a two-level programming language (one level for the array controller, and one for the SIMD array itself) that has two options for programming the array: a high-level assembler with full expressive power, and a high-level expression language that allows straightforward expression of data-parallel programs. In support of this expression language, we have developed an abstract treatment of com-

munication in fine-grain machines that unifies it with computation, and allows the automatic generation of code that maximizes sharing of intermediate results among neighboring processors. Our preliminary experiments with a handful of examples have yielded high-level code of handcrafted quality.
Fisher and Highnam have made further progress in mapping computer vision algorithms onto such machines, and Highnam is developing a flexible coding and simulation facility for further algorithm studies. Rockoff is investigating functional block designs in CMOS in order to provide good speed and area estimates for the detailed design.

8.3.2. A Coprocessor Design Environment

The coprocessor design environment project, started in Fall 85, is developing a suite of hardware and software tools aimed at assisting the process of designing and deploying custom coprocessors within an existing application environment. The tools provide early feedback on eventual system performance as well as assistance in hardware and software interfacing.

We have completed the logic design of a MC68020-compatible coprocessor design frame. We are currently designing an example coprocessor with raster graphics and data structure applications that exercises the most commonly used features of the frame. We expect the design frame to use the pending MOSIS 108 pin standard; in case the standard is not set by the time the example design is ready for fabrication, we plan to disable some signals and fabricate it in an 84 pin package.

We are also using the example design to test out our initial design and implementation of a coprocessor interface compiler that produces code generation facilities, and a performance simulator, which runs actual application code, emulates the coprocessor, and provides estimates of system performance before the device is built.

We have completed a register-transfer and transistor-level logic design of a 68020-compatible coprocessor design frame at the detailed timing level. We have tested the logic design by running it against a functional simulator, and we have begun to lay out the chip [Chatterjee&.86.Specialized]. We plan to insert coprocessors in systems through the use of a daughterboard holding the CPU, the coprocessor and any peripheral circuitry. Our initial design will support slave-mode and DMA operation. We had initially expected that DMA, which our preliminary studies show can yield speedups on many tasks in the 68020 environment, would require the use of a support chip, since we have limited ourselves to MOSIS standard packages with an eye to making the design frame and associated software available to other MOSIS users. Now, however, we plan to use a soon-to-be-announced MOSIS package with a higher pincount.

Specialized Coprocessors

In this area we have begun a study of single-chip coprocessors, linked closely to general-purpose hosts as accelerators for inner loops of programs. As a first step, we have begun to measure the expected performance of coprocessors that use system memory rather than local memory, and hence are essentially bandwidth limited. We

plan to carry out similar estimates for more algorithms, and to do more detailed estimates of coprocessors with and without local memories. We also plan to design and implement a CMOS design frame, a la Katz, for the production of coprocessors for the Motorola 68020.

8.3.3. Pipelined Architectures for Data-dependent Algorithms

We have been working on architectures for data dependent applications, such as encountered in speech recognition. First, we evaluated a class of custom architectures in which the processor, the processor memory interconnection, and the synchronization supported by the shared memory could be customized to the task, using custom VLSI chips. One particular task, the Harpy speech recognition system, was fully simulated on a number of architecture configurations, and speed ups of above 8000 compared to the VAX-780 were obtained assuming MOSIS level NMOS technology (100 ns clock cycle time, 400ns memory cycle time). The speed up as a function of the number of MOS transistors (excluding memory) was found to be superlinear, even though the speed up in terms of number of processors was sublinear. The explanation lies in the reduction in number of MOS transistors per processor that can be achieved by customizing each processor individually to more and more specialized tasks.

Currently, we are investigating a general-purpose architecture for data dependent programs. Our aim is to provide performance similar to that of the full custom architecture with full programmability while eliminating hardware utilization (speedup/MOS transistors). We expect to use VLSI technology to implement a memory controller which supports efficient synchronization primitives including HEP style bit/word semaphores, Cedar style synchronization keys, atomic operations on shared queues and stacks, and blocking/unblocking of processors waiting on synchronization events.

Bisiani and Anantharaman have completed simulations of both a pipelined and a parallel version of the search accelerator. The simulations have been done using real speech data. The simulations indicate that both versions of the accelerator can speed up search by about three orders of magnitude [Anantharaman&.86.Hardware].

We are not planning to build a VLSI device because of the manpower involved and because the details of the search algorithm (e.g. size of some data elements, pruning heuristics, etc.) are still changing. Since the device would be a definite plus for the speech project we are now planning to build an emulator out of off-the-shelf hardware. Because of the nature of this project and since it interacts closely with all the other "software" work done for speech we have proposed to build the accelerator as part of the next Strategic Computing contract.

Continuing our work toward designing an architecture suitable for a large set of speech recognition search algorithms, we have analyzed in detail various algorithms for searching Hidden Markov model graphs. We have described all the algorithms as variations of a single basic algorithm and compared their individual requirements. Given this unified description, we can recast each algorithm as a pipeline of simple register-to-

register machines that can be tailored to attain some speed/cost or speed/size trade-off. We are now building a simulator to test various trade-offs using (or with) real data. The other issue we have been working on is how to use a collection of identical pipeline machines to achieve higher performance. Load balancing is a challenge because all the algorithms we are dealing with are data dependent. As soon as we have a reasonable design, we will build and test it within the distributed speech system. We expect to be able to do that easily since most of the hardware/software interfacing has already been done for a signal processing accelerator and we can take advantage of this environment.

8.3.4. Chess Machine

Chess programs and chess machines that search a large space of possibilities in a very simple way have established dominance over chess programs that try to bring a lot of knowledge to bear in guiding the search and evaluating each situation.

A Parallel Chess Machine

The CMU Chess Machine, which features a move generator built from 64 custom VLSI chips designed by Ebeling and fabricated through MOSIS, came to life in 1985. Programmed by Berliner's chess group, it has since attained a provisional USCF rating of 2170 (higher than that of any current computer program) and is expected to achieve Master level (2200) before long. The machine comprises a microprogrammed processor that controls a variety of chess-specific hardware, including the move generator that produces and orders moves at the rate of 200,000 moves/s.

Most successful chess programs utilize a brute-force α-β search with as much positional evaluation at the leaf positions as can be afforded. Since positional evaluation can be done very fast incrementally while move generation requires a lot of time, we chose to investigate whether we could use VLSI to implement a fast move generator. The result is an array of 64 (identical) custom VLSI chips, each of which generates a subset of the legal moves. The chips also perform move ordering based on capture information and square safety and maintain the search context for searches up to 40 ply.

Each chip comprises about 14,000 transistors, divided about equally between the legal move computation, move ordering and the context stack. The time required to perform the different operations ranges from 120 nanoseconds for making moves to 300 nanoseconds for performing a distributed arbitration cycle. The time required to process each position in the search with the current hardware implementation is about 5 microseconds and includes the time for evaluation, handling the transposition and repetition tables, and the α-β search.

After about 20 tournament games with a variety of human players, the machine has yet to lose to a player rated under 2100 and has a win and a draw in 5 games against master players. While the speed of the hardware has brought us this far, we have implemented the position evaluation as programmable hardware that is not yet being fully utilized. We expect that the speed of the move generator combined with more intelligent evaluation will take us well into the Master category.

A 3-Chip Set for Chess

We are now applying our Hitech and Chess Machine experience to develop a three-chip set: move generator, controller, and evaluator. The three-chip set is expected to search at around one million positions/second, or about 5-10 times faster than the current generation of chess machines. While the chip set should be able to reach well into Master level performance searching in single-processor configuration, the real goal is to operate the chip set in a multiprocessor configuration. A new parallel α-β algorithm that has shown "asymptotically optimal" speedup in preliminary analytical results is now been examined. The controller chip itself is also being designed along with the study of the parallel algorithm. The three-chip set coupled with a commercial static RAM chip will form a self-contained chess machine. Inside the controller will be the move stack, various status registers, communications interface to the parent processor, and board repetition detection logic. The repetition detection logic is based on a new incremental algorithm instead of the usual hash table implementation. The evaluator is based on a new pipelined evaluation scheme and will implement a mixture of the Belle evaluation function and the Hitech evaluation function.

Our first chip was a 40 pin chess move generator that essentially implemented the Belle move generation algorithm. The chip incorporates several circuit refinements and measures 6912x6812 microns (MOSIS standard die size) in three-micron p-well CMOS process. We simulated the ~36K-transistor chip in its entirety on a Daisy workstation at the transistor switch level. SPICE simulation of the critical path circuitry indicated a maximum throughput of about two million moves/second.

The initial fabrication yield was good (up to 50%), with the exception of a run from one vendor apparently having fabrication problems. We built a chip tester and a simple chess machine around the chip to evaluate its performance. The chip operates at a raw speed of about 2,000,000 moves/second—about three times as fast as the Belle TTL move generator and 10 times as fast as the Hitech 64-chip move generator. In terms of speed over chip count, the single chip move generator is about three orders of magnitude better than either of the previous designs. We have built one test machine capable of searching around 1,000,000 nodes/second (less than the raw speed because of controller overhead). We are currently constructing a machine using three move generators that will search some 3,000,000 nodes/second, about 20 times faster than any existing chess machine.

ChipTest, the Sun-based system built around one of our move generator chips, was recently crowned the new ACM North American Computer Chess Champion. A new two processor chess machine is currently being constructed. The new machine is expected to run at around 2,000,000 nodes/second. This represents about a factor of four increase in raw speed over the retiring champion. Because of algorithm improvement, the actual speed increase should be around a factor of five. The new machine will be completed before the end of the 1987. The new machine will be a single VME triple height, full depth board that plugs directly into a Sun workstation. Based on test results between ChipTest and Hitech, we expect that once the chess knowledge in Hitech is merged with the new machine, a computer grandmaster will become a reality.

We have also made progress on new algorithms for minimax searching. Our "selective extension" method of following promising lines of play has shown success in preliminary experiments and will receive a full-scale trial when the new hardware is ready.

We have started work on a companion controller/evaluator. Because of the good yield achieved for the move generator, we decided to design the controller and the evaluator as a single chip. Preliminary simulation results on parallel α-β algorithms have been extremely promising; speedup on the order of hundreds seems to be readily achievable. The new parallel algorithms have also been shown theoretically to dominate the weaker form of sequential α-β algorithm that does not use deep cutoff.

8.4. Bibliography

[Anantharaman and Bisiani 86]
Anantharaman, T., and R. Bisiani.
A hardware accelerator for speech recognition algorithms.
In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, June, 1986.
Two custom architectures tailored to a speech recognition beam search algorithms are described. Both architectures have been simulated using real data and the results of the simulation are presented. The design of the custom architectures is described, and a number of ideas are presented on the automatic design of custom systems for data-dependent computations

[Black 85]
Black, D.L.
On the existence of delay-insensitive fair arbiters: Trace theory and its limitations.
Technical Report CMU-CS-85-173, Carnegie Mellon University Computer Science Department,
October, 1985.
In this paper, we attempt to settle the controversy over whether delay-insensitive fair arbiters exist. We examine Udding's (1985) claim that they do not exist and find that this result is theoretically correct but of no practical significance because it relies on an inappropriate notion of fairness. We show that for the relevant notions of fairness, the existing trace theory of finite traces lacks sufficient expressive power to adequately specify a fair delay-insensitive arbiter--the existing specification of a fair arbiter is also satisfied by an unfair arbiter. Based on this reasoning, we extend trace theory to include infinite traces, and show by example the importance of including liveness in such a theory. The extended theory is sufficiently expressive to distinguish fair arbiters from unfair ones. We use this theory to establish the existence of a delay-insensitive fair arbiter. In the process of formulating the extension we develop a more general trace-theoretic composition operator that does not require the domain constraints (composability restrictions) used by other authors. Finally, we introduce wire modules as an abstraction to capture the important role transmission media properties play in circuit behavior.

[Browne and Clarke 85]
Browne, M.C. and E.M. Clarke.
SML - a high level language for the design and verification of finite state machines.
Technical Report CMU-CS-85-179, Carnegie Mellon University Computer Science Department,
November, 1985.

In this paper, we describe a finite state language named SML (State Machine Language) and illustrate its use with two examples. Although the compilation procedure is exponential, the compiler is fast enough that we believe that SML can still be a useful tool for the design of small (< 1000 state) finite state machines. Furthermore, we have interfaced our SML compiler with a temporal logic theorem prover that can assist in the debugging and verification of SML programs. In addition to being useful for design, SML can also be a documentation aid, since it provides a succinct notation for describing complicated finite state machines. A program written in SML can be compiled into a state transition table that can then be implemented in hardware using an appropriate design tool. The output of the SML compiler can also be used by the Berkeley VLSI design tools to layout the finite state machines as either a ROM, PLA, or PAL.

[Browne et al. 85a]
Browne,M., E. Clarke, D. Dill, and B. Mishra.
Automatic verification of sequential circuits.
In *Proceedings of the 7th International Conference on Computer Hardware Description Languages*, August, 1985.
Also available as Technical Report CMU-CS-85-100.

Verifying the correctness of sequential circuits has been an important problem for a long time. But lack of any formal and efficient method of verification has prevented the creation of practical design aids for this purpose. Since all the known techniques of simulation and prototype testing are time-consuming and not very reliable, there is an acute need for such tools. In this paper we describe an automatic verification system for sequential circuits in which specifications are expressed in a propositional temporal logic. In contrast to most other mechanical verification systems, our system does not require any user assistance and is quite fast -- experimental results show that state machines with several hundred states can be checked for correctness in a matter of seconds!

The verification system uses a simple and efficient algorithm, called a *Model Checker*. The algorithm works in two steps: in the first step, it builds a labeled state-transition graph; and in the second step, it determines the truth of a tem-

poral formula with respect to the state-transition graph. We discuss two different techniques that we have implemented for automatically generating the state-transition graphs: The first involves extracting the state graph directly from the circuit by simulation. The second obtains the state graph by compilation from an HDL specification of the original circuit. Although these approaches are quite different, we believe that there are situations in which each is useful.

[Browne et al. 85b]
Browne, M.C., E.M. Clarke, and D.L. Dill.
Automatic circuit verification using temporal logic: two new examples.
In *1985 IEEE International Conference on Computer Design: VLSI in Computers*, IEEE, October, 1985.
In this paper provide further evidence for the usefulness of our approach by describing enhancements to the basic verifier that automates the extraction of state transition graphs from circuits. We discuss two different techniques. The first approach involves extracting the state graph from a wire-list description of the circuit and is for asynchronous circuits. The second obtains the state diagram by compilation from an HDL specification of the original circuit. Although these approaches are quite different, we believe that there are situations in which each is useful.

[Brunvand 87]
Brunvand, E.
Parts-R-Us: A chip apart(s)....
Technical Report CMU-CS-87-119, Carnegie Mellon University Computer Science Department,
May, 1987.
Parts-R-Us is a chip that contains a collection of building block parts for asynchronous circuit design. The parts contained on the chip are either not available as standard commercial components, or are standard gates combined into small modules that are particularly useful for building asynchronous control circuits. There are eight different configurations of Parts-R-Us, each offering a different set of asynchronous parts to the user. The parts contained on the chip include: C-elements, transition call modules, transition selectors, transition toggles, transition arbiters, a four phase mutual exclusion element, an asynchronous register, two phase Q-registers, and four phase Q-registers.
This document is both a description of Parts-R-Us, and a user's manual for designers using the chip.

[Bryant 85a] Bryant, R.E.
Symbolic manipulation of Boolean functions using a graphical representation.
In *22nd Design Automation Conference*, IEEE, June, 1985.

In this paper we describe a data structure for representing Boolean functions and an associated set of manipulation algorithms. Functions are represented by directed, acyclic graphs in a manner similar to the representations of Lee and Akers, but with further restrictions on the ordering of decision variables in the graph. Although a function requires, in the worst case, a graph of size exponential in the number of arguments, many of the functions encountered in typical applications have a more reasonable representation. Our algorithms are quite efficient as long as the graphs being operated on do not grow too large. We present performance measurements obtained while applying these algorithms to problems in logic design verification.

[Bryant 85b] Bryant, R.E.
Symbolic verification of MOS circuits.
In *1985 Chapel Hill Conference on VLSI*, Computer Science Press, Inc., March, 1985.

The program MOSSYM simulates the behavior of a MOS circuit represented as a switch-level network *symbolically*. That is, during simulator operation the user can set an output to either 0, 1, or a Boolean variable. The simulator then computes the behavior of the circuit as a function of the past and present input variables. By using heuristically efficient Boolean function manipulation algorithms, the verification of a circuit by symbolic simulation can proceed much more quickly than by exhaustive logic simulation. In this paper we present our concept of symbolic simulation, derive an algorithm for switch-level symbolic simulation, and present experimental measurements from MOSSYM.

[Bryant 86a] Bryant, R.
A collection of papers about a symbolic analyzer for MOS circuits.
Technical Report CMU-CS-86-114, Carnegie Mellon University Computer Science Department,
March, 1986.

COSMOS, a compiled Simulator for MOS Circuits, aims to perform accurate switch-level simulation at least an order of magnitude faster than previous simulators such as MOSSIM II. Unlike programs that operate directly on the transistor level description during simulation, COSMOS transforms the transistor network into a Boolean description during a preprocessing step. This Boolean description, produced by a symbolic analyzer, captures all aspects of

switch-level networks including bidirectional transistors, stored charge, different signal strengths, and indeterminate (X) logic values. These papers give a brief overview of COSMOS as well as a detailed presentation of the theory and algorithms behind the symbolic analyzer.

[Bryant 86b] Bryant, R.
Can a simulator verify a circuit?,
In Milne, G.J., *Formal Aspects of VLSI Design*. North-Holland, 1986.
A logic simulator can prove the correctness of a digital circuit if only circuits implementing the system specification can produce a particular response to a sequence of simulation commands. This paper explores two methods for verifying circuits by a three-valued logic simulator where the third state X indicates an indeterminate value. The first, called black-box simulation, involves simply observing the output produced by the simulator in response to a sequence of inputs with no consideration of the internal circuit structure. This style of simulation can verify only a limited class of systems. The second method, called transition simulation, requires the user to specify the relation between states in the circuit and the specification. The simulator is then used to prove that each state transition in the specification is implemented correctly. Arbitrary systems may be verified by this method.

[Bryant 86c] Bryant, R.E.
Graph-based algorithms for Boolean function manipulation.
*IEEE Transactions on Computers*C-35(8):677-691, 1986.
Also available as CMU-CS-85-135.
In this paper we present a new data structure for representing Boolean functions and an associated set of manipulation algorithms. Functions are represented by directed, acrylic graphs in a manner similar to the representations introduced by Lee and Akers, but with further restrictions on the ordering of decision variables in the graph. Although a function requires, in the worst case, a graph of size exponential in the number of arguments, many of the functions encountered in the typical applications have a more reasonable representation. Our algorithms have time complexity proportional to the sizes of the graphs being operated on, and hence are quite efficient as long as the graphs do not grow too large. We present experimental results form applying these algorithms to problems in logic design verification that demonstrate the practicality of our approach.

[Bryant 87a] Bryant, R.
Graph-based algorithms for Boolean function manipulation.
In *IEEE Transactions on Computers*, IEEE, August, 1987.

In this paper we present a new data structure for representing Boolean functions and an associated set of manipulation algorithms. Functions are represented by directed, acyclic, graphs in a manner similar to the representations introduced by Lee and Akers, but with further restrictions on the ordering of decision variables in the graph. Although a function requires, in the worst case, a graph of size exponential in the number of arguments, many of the functions encountered in typical applications have a more reasonable representation. Our algorithms have time complexity proportional to the sizes of the graphs being operated on, and hence are quite efficient as long as the graphs do not grow too large. We present experimental results from applying these algorithms to problems in logic design verification that demonstrate the practicality of our approach.

[Bryant 87b] Bryant, R.E.
Two papers on a symbolic analyzer for MOS circuits.
Technical Report CMU-CS-87-106, Carnegie Mellon University Computer Science Department,
February, 1987.

This report contains two papers describing a set of algorithms to extract the logical behavior of a digital metal-oxide semiconductor (MOS) from its transistor representation. Switch-level network analysis, applied symbolically, performs the extraction. The analyzer captures all aspects of switch-level networks including bidirectional transistors, stored charge, different signal strengths, and indeterminate (X) logic values. The output is a set of Boolean formulas, where the behavior of each network node is represented by a pair of formulas. In the worst case, the analysis of an n node network can yield a set of formulas containing a total of $O(n^3)$ Boolean operations. However, all but a limited set of dense, pass transistor networks give formulas with $O(n)$ total operations.

The analyzer can serve as the basis of efficient programs for a variety of logic design tasks, including : logic simulation (on both conventional and special purpose computers), fault simulation, test generation, and symbolic verification.

These papers have been accepted for publication in *IEEE Transactions on Computer-Aided Design of Integrated Circuits.*

[Bryant 87c] Bryant, R.E.
A methodology for hardware verification based on logic simulation.
Technical Report CMU-CS-87-128, Carnegie Mellon University Computer Science Department,
June, 1987.

A logic simulator can prove the correctness of a digital circuit if it can be shown that only circuits implementing the system specification will produce a particular response to a sequence of simulation commands. This style of verification has advantages over other proof methods in being readily automated and requiring less attention to the low-level details of the design. It has advantages over other approaches to simulation in providing more reliable results, often at a comparable cost.

This paper presents the theoretical foundations of several related approaches to circuit verification based on logic simulation. These approaches exploit the three-valued modeling capability found in most logic simulators, where the third value *X* indicates a signal with unknown digital value. Although the circuit verification problem is NP-hard as measured in the size of the circuit description, several techniques can reduce the simulation complexity to a manageable level for many practical circuits.

[Bryant and Schuster 85]
Bryant, R.E. and M.D. Schuster.
Performance evaluation of FMOSSIM, a concurrent switch-level fault simulator.
In *22nd Design Automation Conference*, IEEE, June, 1985.

This paper presents measurements obtained while performing fault simulation of MOS circuits modeled at the switch level. In this model the transistor structure of the circuit is represented explicitly as a network of charge storage nodes connected by bidirectional transistor switches. Since the logic model of the simulator closely matches the actual structure of MOS circuits, such faults as stuck-open and closed transistors as well as short and open-circuited wires can be simulated. By using concurrent simulation techniques, we obtain a performance level comparable to fault simulators using logic gate models. Our measurements indicate that fault simulation times grow as the product of the circuit size and number of patterns, assuming the number of faults to be simulated is proportional to the circuit size. However, fault simulation ties depend strongly on the rate at which the test patterns detect the faults.

[Chatterjee and Fisher 86]
Chatterjee, S., and A.L. Fisher.
Specialized coprocessor chips: fast computation with slow memory.
In *Platinum Jubilee Conference on Systems and Signal Processing*, December, 1986.

Advances in VLSI technology, computer aided design and design automation, and rapid turnaround fabrication have

made the use of special-purpose architectures more and more practical. Our focus in this paper is on the prospects of using special-purpose devices to speed up the inner loops of programs, as an extension of the usual techniques of code tuning and vertical migration. In particular, we consider the use of single-chip coprocessors that do not contain large local memories, and hence operate on data stored in system memory. We show that the intrinsic efficiency of specialized hardware can lead in some cases to dramatic speedups over software implementations, despite the processor-memory "bottleneck."

[Clarke and Feng 85]
Clarke, E.M., and Y. Feng.
Escher-- a geometrical layout system for recursively defined circuits.
Technical Report CMU-CS-85-150, Carnegie Mellon University Computer Science Department,
July, 1985.
An Escher circuit description is a hierarchical structure composed of cells, wires, connectors between wires, and pins that connect wires to cells. Cells may correspond to primitive circuit elements, or they may be defined in terms of lower level subcells. Unlike other geometrical layout systems, a subcell may be instance of the cell being defined. When such a recursive cell definition is instantiated, the recursion is unwound in a manner reminiscent of the procedure call copy rule in Algol-like programming languages. Cell specifications may have parameters that are used to control the unwinding of recursive cells and to provide for cell families with varying numbers of pins and other internal components. We illustrate how the Escher layout system might be used with several nontrivial examples, including a parallel sorting network and a FFT implementation. We also briefly describe the unwinding algorithm.

[Dally and Bryant 85]
Dally, W.J., and R.E. Bryant.
A hardware architecture for switch-level simulation.
In *IEEE Transactions on Computer-Aided Design*, IEEE, July, 1985.
The Mossim Simulation Engine (MSE) is a hardware accelerator for performing switch-level simulation of MOS VLSI circuits (1), (2). Functional partitioning of the MOSSIM algorithm and specialized circuitry are used by the MSE to achieve a performance improvement of ~300 over a VAX 11/780 executing the MOSSIM II program. Several MSE processors can be connected in parallel to achieve additional speedup. A virtual processor mechanism allows the MSE to simulate large circuits with the size of the circuit limited only by the amount of backing store available to hold the circuit description.

[Fisher 85a] Fisher, A.L.
Design synthesis and practical considerations for bit-level arithmetic arrays.
In *Second International Symposium on VLSI Technology, Systems and Applications*, Pages 274-277. May, 1985.

Bit-serial implementations of systolic and other array algorithms are often found attractive because of their potential for ease of design, high clock rates and flexible word length. The author deals with two aspects of serial implementations: their design and their cost and performance. In the first section, the author shows a new approach to deriving serial arrays, in two steps, from word-parallel arrays of inner product cells. In the second section, he considers the costs and benefits of such arrays, and presents a list of factors that determine the best degree of serialization in a given system.

[Fisher 85b] Fisher, A.L. and P.T. Highnam.
Real-time image processing on scan line array processors.
In *Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, IEEE, November, 1985.

[Fisher 85c] Fisher, A.L. and H.T Kung.
Synchronizing large VLSI processor arrays.
In *IEEE Transactions on Computers*, Pages 734-740. IEEE, 1985.

Highly parallel VLSI computing structures consist of many processing elements operating simultaneously. In order for such processing elements to communicate among themselves, some provision must be made for synchronization of data transfer. The simplest means of synchronization is the use of a global clock. Unfortunately, large clocked systems can be difficult to implement because of the inevitable problem of clock skews and delays, which can be especially acute in VLSI systems as feature sizes shrink. An alternative means of enforcing necessary synchronization is the use of self-timed asynchronous schemes, at the cost of increased design complexity and hardware cost. Realizing that different circumstances call for different synchronization methods, this paper provides a spectrum of synchronization models; based on the assumptions made for each model, theoretical lower bounds on clock skew are derived, and appropriate or best possible sychronization schemes for large processor arrays are proposed.

One set of models is based on assumptions that allow the use of a pipelined clocking scheme where more than one clock event is propagated at a time. In this case, it is shown that even assuming that physical variations along clock lines can produce skews between wires of the same length, any one-dimensional processor array can be correctly

synchronized by a global pipelined clock while enjoying desirable properties such as modularity, expandability, and robustness. This result cannot be expanded to two-dimensional arrays, however; the paper shows that under this assumption, it is impossible to run a clock such that the maximum clock skew between two communicating cells will be bounded by a constant as systems grow. For such cases, or where pipelined clocking is unworkable, a synchronization scheme incorporating both clocked and asynchronous elements is proposed.

[Fisher 85d] Fisher, A.L.
Memory and modularity in systolic array implementations.
In *International Conference on Parallel Processing*, August, 1985.
Although systolic array algorithms are usually pictured in terms of processing elements with very little if any local storage, practical implementations can often make good use of sizable local memories. This paper explores some of the cost, performance and modularity issues involved in memory-intensive systolic array implementations.
The paper is divided into two main sections. The first presents a clarification and uniform summary of two known applications of local memory in systolic arrays, and derives some new design criteria and improves on some existing designs. The second section proposes a structure which allows the efficient separation of dataflow from computation in a systolic implementation, providing some benefits of flexibility and modularity.

[Fisher 86] Fisher, A.L.
Scan line array processors for image computation.
In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, June, 1986.
The scan line array processor (SLAP), a new architecture designed for high-performance, low-cost image computation, is described. A SLAP is an SIMD linear array of processors, and hence is easy to build and scales well with VLSI technology. At the same time, appropriate special features and programming techniques make it efficient for a wide variety of low- and medium-level computer vision tasks. The basic SLAP concept and some of its variants are described, a particular planned implementation is discussed, and its performance for computer vision and other applications is indicated.

[Fisher et al. 85] Fisher, A.L., H.T. Kung, and K. Sarocky.
Experience with the CMU programmable systolic chip,
In P. Antognetti, F. Ancheau, and J. Vimillemin, *Microarchitecture of VLSI Computers*, Pages 209-222. Martinus Nijhoff Publishers, 1985.
Also available as Technical Report CMU-CS-85-161.

The CMU *programmable systolic chip* (PSC) is an experimental, microprogrammable chip designed for the efficient implementation of a variety of systolic arrays. The PSC has been designed, fabricated, and tested. The chip has about 25,000 transistors, uses 74 pins, and was fabricated through MOSIS, the DARPA silicon broker, using a 4 micron nMOS process. A modest demonstration system involving nine PSCs is currently running. Larger demonstrations are ready to be brought up when additional working chips are acquired.

The development if the PSC, from initial concept to silicon layout, took slightly less than a year, but testing, fabrication, and system demonstration took an additional year. This paper reviews the PSC, describes the PSC demonstration system, and discusses some of the lessons learned from the PSC project.

[Gupta 86] Gupta, A.
Parallelism in production systems.
Technical Report CMU-CS-86-122, Carnegie Mellon University Computer Science Department,
March, 1986.

Production systems (or rule-based systems) are widely used in Artificial Intelligence for modeling intelligent behavior and building expert systems. Most production system programs, however, are extremely computation intensive and run quite slowly. The slow speed of execution has prohibited the use of production systems in domains requiring high performance and real-time response. This thesis explores the role of parallelism in the high-speed execution of production systems.

On the surface, production system programs appear to be capable of using large amounts of parallelism- it is possible to perform match for each production in a program in parallel. The thesis shows that in practice, however, the speed-up obtainable from parallelism is quite limited, around 10-fold as compared to initial expectation of 100-fold to 1000-fold. The main reasons for the limited speed-up are: (1) there are only a small number of productions that are affected (require significant processing) per change to working memory; (2) there is a large variation in the processing requirement of these productions; and (3) the number of changes made to working memory per recognize-act cycle

is very small. Since the number of productions affected and the number of working-memory changes per recognize-act cycle are not controlled by the implementor of the production system interpreter (they are governed mainly by the author of the program and the nature of the task), the solution to the problem of limited speed-up is to somehow decrease the variation in the processing cost of affected productions. The thesis proposes a parallel version of the Rete algorithm which exploits parallelism at a very fine grain to reduce the variation. It further suggests that to exploit the fine-grained parallelism, a shared-memory multiprocessor with 32-64 high performance processors is desirable. For scheduling the fine-grained tasks consisting of about 50-100 instructions, a hardware task scheduler is proposed.

[Hsu 86] Hsu, F.H.
Two designs of functional units for VLSI based chess machines.
Technical Report CMU-CS-86-103, Carnegie Mellon University Computer Science Department,
January, 1986.

Brute force chess automata searching 8 plies (4 full moves) or deeper have been dominating the computer chess scene in recent years and have reached master level performance. One interesting question is whether 3 or 4 additional piles coupled with an improved evaluation scheme will bring forth world championship level performance. Assuming an optimistic branching ratio of 5, speedup of at least one hundredfold over the best current chess automaton would be necessary to reach the 11 to 12 plies per move range.

One way to obtain such speedup is to improve the gate utilization and then parallelize the search process. In this paper, two new designs of functional units with higher rate efficiency than previous designs in the literature will be presented. The first design is for move generation only, and is essentially a refinement of the move generator used in the Belle chess automation, the first certified computer chess master. The second design is a general scheme that can be used for evaluating a class of chess-specific functions, besides generating moves. A move generator based on the second design will be described. Applications of the same general scheme will be briefly discussed.

[Hsu et al. 85] Hsu, F.H., H.T. Kung, T. Nishizawa, and A. Sussman.
Architecture of the link and interconnection chip,
In Fuchs, H., *1985 Chapel Hill Conference on Very Large Scale Integrated Systems.* Computer Science Press, 1985.

The link and interconnection chip (LINC) is a custom chip

whose function it is to serve an efficient link between system functional modules, such as arithmetic units, register files and I/O ports. This paper describes the architecture of LINC, and justifies it with several application examples.

LINC has 4-bit datapaths consisting of an 8x8 crossbar interconnection, a FIFO or programmable delay for each of its inputs, and a pipeline register file for each of its outputs. Using pre-stored control patterns LINC can configure an interconnection and delays on-the-fly. Therefore the usual functions of busses and register files can be realized with this single chip.

LINC can be used in a bit-sliced fashion to form interconnections with datapaths wider than 4 bits. Moreover, by tri-stating the proper data output pins, multiple copies of LINC can for crossbar interconnections larger than 8x8.

Operating at the target cycle time of 100 ns, LINC makes it possible to implement a variety of high-performance processing elements with much reduced package counts.

[Kung 85] Kung, H.T.
Memory requirements for balanced computer architectures.
Technical Report CMU-CS-85-158, Carnegie Mellon University Computer Science Department,
June, 1985.

In this paper, a processing element (PE) is characterized by its computation bandwidth, I/O bandwidth, and the size of its local memory. In carrying out a computation, a PE is said to be *balanced* if the computing time equals the I/O time. Consider a balanced PE for some computation. Suppose that the computation bandwidth of the PE is increased by a factor of α relative to its I/O bandwidth. Then when carrying out the same computation the PE will be imbalanced, i.e., it will have to wait for I/O. A standard method to avoid this I/O bottleneck is to reduce the overall I/O requirement of the PE by increasing the size of its local memory. This paper addresses the question of by how much the PE's local memory must be enlarged in order to restore balance.

[Lam and Mostow 85]
Lam, M.S. and J. Mostow.
A Transformational Model of VLSI Systolic Design.
*Computer*18(2):42-52, February, 1985.
An earlier version appears in *Proc. 6th International Symposium on Computer Hardware Description Languages and Their Applications*, May, 1983.

[Mishra 86] Mishra, B.
Some graph theoretic issues in VLSI design.
Technical Report CMU-CS-86-117, Carnegie Mellon University Computer Science Department,
May, 1986.

It is often said that VLSI design is the ultimate batch job! The statement succinctly characterizes several problems that a VLSI designer must face: many man-months of design effort, high turn-around time, long hours spent in testing and the difficulty of correcting a design error. In order to alleviate these problems, a VLSI design must be equipped with powerful, efficient and automated design tools. In addition, such tools, if properly designed, can help in eliminating the errors that hand-designs are prone to.

The first part of the thesis describes a graph theoretic problem (called All-Bidirectional-Edges Problem) that arises naturally in the context of the simulation of an MOS transistor network. The algorithm can be used to quickly detect all the pass-transistors in the network that can behave as bilateral devices. The knowledge of such transistors in the network can be used profitably in several existing simulation algorithms to obtain a significant speed-up in their performance. In addition, the algorithm can also be used to find sneak paths in the network and hence detect functional errors.

The second part of the thesis studies the design of tools for verifying the correctness of sequential circuits. Though the problem of verifying asynchronous circuits is considered to be rather important, there is a severe lack of practical design aids for this purpose. Since all the known techniques of simulation and prototype testing are time-consuming and not very reliable, the need for verification tools becomes more acute. Moreover, as we build larger and more complex circuits, the cost of a single design error is likely to become even higher. We investigate several algorithm design issues involved in an automatic verification system for (asynchronous) sequential circuits, in which the specifications are expressed in a propositional temporal logic of branching-time, call CTL. We also study how to tackle a large and complex circuit by verifying it hierarchically.

[Nowatzyk 85] Nowatzyk, A.
Advanced design tools for programmable logic devices.
Technical Report CMU-CS-86-121, Carnegie Mellon University Computer Science Department,
November, 1985.

Programmable Logic Devices (PLD's) such as field programmable logic arrays based on fusable link or floating gate

technology have become a viable alternative to random logic designs. Increasing device complexity and decreasing gate transfer delays allow PLD's to replace large fractions of circuits previously implemented with LSI and MSI chips. This trend has permitted designs that use fewer device types and achieve much higher logic densities.

PLD's are not only a direct replacement for conventional gate and flip flop level designs, but they support systematic design styles that were considered to be too expensive with conventional logic. The cost function for PLD designs can be radically different from those of gate-level implementations because PLD's come in standard sizes, so the actual number of gates used matters little as long as it does not exceed the available resources in a given package.

The complexity of PLD's and the unique constraints that a PLD imposes on a logic design requires new design tools to describe, encode, verify, and implement PLD based circuits. This paper provides a brief overview of available PLD design aid software and describes a more general method of using a general purpose language (here C) to specify and implement complex functions efficiently.

[Walker 85] Walker, H. and S.W. Director.
VLASIC: A Yield Simulator for Integrated Circuits.
In *IEEE International Conference on Computer-Aided Design*, IEEE, November, 1985.

APPENDIX I
GLOSSARY

Accent	network operating system for the Spice project that influenced the development of Mach
ALOEGEN	A Language-Oriented Editor GENerator, structure editor for syntax description
ANAMOS	a symbolic analyzer for MOS circuits that captures all aspects of switch-level networks
ARL	Action Routine Language
Avalon	a set of linguistic constructs designed to give programmers explicit control over transaction-based processing of atomic objects for fault-tolerant applications
Camelot	CArnegie MEllon Low Overhead Transaction facility, implements the synchronization, recovery, and communication mechanisms needed to support the execution of distributed transactions and the definition of shared abstract objects
Canny	an algorithm that finds high-contrast edges in smoothed images
Chinese Tutor	an intelligent computer-based tutoring system for beginner-level Chinese
ChipTest	a chess system built around one of our move-generator chips
Chunker	a program that applies the notion of chunking to pawn endings in chess
CIL	Calibrated Imaging Laboratory
CLX	a standard interface between Common Lisp and X
CMU-CSD	Carnegie Mellon Computer Science Department
CoalSORT	a prototype intelligent interface to a large bibliographic database in the coal technology domain
COSMOS	COmpiled Simulator for MOS circuits that combines the capabilities of MOSSIM II and FMOSSIM and is nearly an order of magnitude faster
Cypress-Soar	a Soar implementation of the Cypress system which designs divide-and-conquer algorithms
DBgen	a system for linking environment descriptions to form a working editor
Designer-Soar	a Soar implementation and redesign of Designer, an automatic algorithm designer
DML	Digital Mapping Laboratory
DP	Distributed Processing
DSN	Distributed Sensor Network
EBL	Explanation-Based Learning module within the Prodigy system

EBS	Explanation-Based Specialization module within the Prodigy system
Escher	a system that allows the user to describe the recursive structure of complex circuits using a graphical interface
Flamingo	a system for managing the interface between users and programs that run in large, distributed, heterogeneous environments
FMOSSIM	a concurrent switch-level fault simulator for MOS circuits
FOG	a distributed object system in Mach
Gandalf	project that investigates ways to facilitate and automate the construction of programming environments
Griffin	GRaceful Interface Featuring Form-based INteraction, an interface tool that implements a form-filling paradigm
Hemlock	a text editor written in Common Lisp for Spice
Hitech	a chess machine with programmable pattern recognition hardware
IDL	Interface Description Language
IPC	InterProcess Communication
ITC	Information Technology Center
IU	Image Understanding
KR	a prototype tool for building simple semantic networks to provide flexible knowledge representation
LGCC	a program that translates the Boolean representation produced by ANAMOS into a set of C evaluation procedures, as well as a set of initialized arrays representing the network structure
Lisp Shell	a customizable Lisp-based command language interpreter
Mach	a multiprocessor operating system kernel and environment
MAPS	Map-Assisted Photointerpretation System
Matchmaker	interprocess interface generator
MetaMenu	a menu-based system that will be one of the building blocks of the UWI
MIG	Mach Interface Generator
Mirage	the device- and window system-independent graphical component of the UWI
MOSSIM II	a software implementation of an algorithm for switch-level simulation of MOS circuits
MOSSYM	a simulator that simulates MOS circuits represented as switch-level networks
NFS	Network File Server
Oracle	SUPREM's primary knowledge repository
Parts-R-Us	a chip that contains a collection of asynchronous circuit building blocks that are unavailable as standard IC parts

Prodigy	an artificially intelligent "learning apprentice" intended to facilitate the acquisition of problem solving expertise
PT	Programming Technology
R1-Soar	a Soar implementation of DEC's VAX configuring expert system R1
RFS	Remote File Server
RISC	Reduced Instruction Set Computer
Rulekit	a production system package that combines frame-like data structures with priority-order control rules
SAR	Synthetic Aperture Radar
Sear	a knowledge acquisition system for R1-like expert systems
Sesame	a distributed file system that demonstrated the feasibility of using ports as tokens of identity
SIMD	Single Instruction, Multiple Data stream
SLAP	Scan-Line Array Processor
SML	State Machine Language
Soar	a general architecture for problem solving and learning
SPAM	System for Photointerpretation of Airports using MAPS
Spice	a large-scale distributed computing environment
SUP	a program for automatically distributing and updating network software
SUPREM	Search Using Pattern Recognition as an Evaluation Mechanism
TABS	TransAction Based Systems
3D Mosaic	a photointerpretation system which combines domain knowledge with image processing
3DFORM	3 Dimensional Frame-based Object Recognition Modeling system
UWI	Uniform Workstation Interface
Vice/Virtue	a distributed file system, similar to Sesame, developed by the CMU ITC
Viewers system	a frame-based interaction system that allows a user to navigate through a network of interconnected frames of information
VIZ	a language for describing flexible unparsing
VLASIC	a catastrophic-fault yield simulator for integrated circuits
VSLI	Very-Large Scale Integration
Warp	Carnegie Mellon's systolic array machine
Warp shell	the top-level user interface to the Warp systolic multiprocessor. The Warp shell is based on the Lisp Shell.
World Modelers	project that explores machine learning within a simulated reactive

environment that facilitates designing, implementing, and testing integrated learning systems

XBAR a high-bandwidth, 16x16 two-bit crossbar chip implemented in double-metal, CMOS P-well technology with 3μm feature size

INDEX