Basic Research in Computer Science
Final Report: 1990-1993

Edited by Dale A. James and C. Roy Taylor
Research Documents Group

22 December 1993

CMU-CS-93-229

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

This report documents a broad program of basic and applied information processing research conducted by Carnegie Mellon's School of Computer Science. The Information Processing Technology Office of the Advanced Research Projects Agency (ARPA) supported this work during the period 1 August 1990 through 2 August 1993.

Chapters 1 through 6 present in detail our six major research areas: Image Understanding, VLSI, Object Management, Integrated Architectures for Intelligent Systems, Creating Graphical Applications, and Types in Programming. Each chapter briefly describes the significant results of one research area and provides references for more detailed descriptions in the published literature.

Sponsored by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the United States Government.

Keywords: Image understanding, very large scale integration, object management, integrated architectures for intelligent systems, creating graphical applications, types in programming.

INTRODUCTION

This report documents a broad program of basic and applied information processing research conducted by Carnegie Mellon's School of Computer Science. The Information Processing Technology Office of the Advanced Research Projects Agency (ARPA) supported this work during the period 1 August 1990 through 2 August 1993.

Chapters 1 through 6 present in detail our six major research areas: Image Understanding, VLSI, Object Management, Integrated Architectures for Intelligent Systems, Creating Graphical Applications, and Types in Programming. Each chapter briefly describes the significant results of one research area and provides references for more detailed descriptions in the published literature.

Work Statement

Our six interrelated projects and their major objectives, as originally formulated in our proposal, follow next.

Image Understanding

Perform basic research in image understanding, emphasizing knowledge representation and algorithm acquisition for vision systems. Our principal tasks include:

- Develop novel methodologies, theories, and computational tools for *physical methods for computer vision*, that is, use of physical models that include optical modeling for reflection, light sources, and material types, and geometrical and material modeling for shapes and deformation.
- Develop and demonstrate a collection of new algorithms for analyzing, extracting features from, and matching two-dimensional and three-dimensional images—based on physical modeling. This includes color image segmentation algorithms with interreflection analysis, and scale-space continuation algorithms using stereo, motion, and boundary information.
- Develop *flying continuous stereo* techniques that can obtain dense and accurate depth maps from an image sequence taken by a moving camera with a very small interframe displacement. Demonstrate the capabilities on sequences of both calibrated images and real or simulated aerial images. Develop specifications for airborne cameras and computational architectures for developing a realtime operational prototype.
- Develop theories and tools for a *vision algorithm compiler* that can exploit sensor and object models to create recognition algorithms for an object under a sensor. Demonstrate vision algorithm compilation for multiple tasks including bin-picking, recognizing airplanes and other objects in SAR images, and metal surface inspection.

VLSI

Develop methodologies and tools for rapidly building and validating VLSI systems. Specific subtasks include:

- Extend COSMOS, our switch level simulator, to handle state of the art VLSI chips with approximately one million transistors. We will achieve this through algorithmic improvements, including the use of hierarchical techniques, and through the reimplementation of COSMOS on parallel machines.
- Develop a practical system for the formal verification of complex VLSI systems, based on symbolic simulation. Our basic approach has already been developed and tested; we need to extend its scope of application, improve its performance, and provide a user interface that makes it easy and safe to use. This work will thus involve both theoretical and engineering components.
- Develop a practical system to automatically generate test vectors for arbitrary MOS VLSI circuits. As for formal verification, we have implemented a proof of concept, but we need to improve both performance and usability to demonstrate the practicality of our approach. We will develop principles for the design of parallel programming languages, along with a specific language design, that promote both ease of programming and efficient compilation across disparate parallel architectures. As part of this work, we will also develop an extensive suite of benchmarks that can be used to assess not only the usefulness and performance of our language, but also the machines on which it runs.
- Develop techniques and tools for the representation, analysis, and compilation of parallel programs, again with the goal of portability across architectures. We will measure the effectiveness of these methods by comparing the resulting code against native code for our benchmarks on several real machines.

Object Management

Develop a highly scalable, distributed system that provides secure and reliable, yet efficient access to persistent objects spanning a broad range of types. The principal tasks of our work will be to:

- Design and build a Common Lisp *framework* that will support multiple, computationally intensive satisfaction engines along with their type-specific query languages and object repositories.
- Design and build *components* that fit into the framework in support of *a software development laboratory*. These components include theorem proving engines, some of which may be implemented on parallel machines for performance speedup. The laboratory will also include an object repository that stores first-order theories and one that stores program modules.
- Design, build, and evaluate an experimental prototype of an architecture that supports *large-scale object distribution*.

- Use the framework and its components in the above distributed prototype. This use will include demonstrating a distributed version of the software development application.

Integrated Architectures for Intelligent Systems

Conduct a broad program of basic research in artificial intelligence in support of the construction of integrated intelligent systems. The principal tasks of our work will be to:

- Design, construct, test and analyze integrated architectures that are capable of supporting the full range of activities required by an intelligent system.
- Study and explore the methods and techniques used by intelligent systems to:
    - Perform a variety of basic reasoning tasks such as for searching, evaluating, abstracting, planning, discovering, etc.
    - Solve specific problems such as jobshop scheduling, robotic exploration, robotic assembly, and logistics.
    - Learn about their environment and learn from their experience.
    - Interact with their external environment via language and robotic means.
- Integrate the above methods and techniques into the candidate integrated architectures so they can be effectively utilized together as an intelligent system.
- Analyze and explore task environments that provide appropriate challenges as tasks for integrated intelligent systems.
- Design, conduct and analyze extended experiments with the candidate architectures that call upon the integrated use of the component processes.
- Develop algorithms and software technologies to support the candidate integrated architectures which are efficient, scalable and operate on both uniprocessor and parallel computer configurations.
- Package and document the underlying software systems and the total candidate architectures to permit their use by other researchers.

Creating Graphical Applications

Develop comprehensive tools that support rapid prototyping and implementation of graphical, highly interactive user interfaces. The principal tasks of our work will be to:

- Determine how to specify application-specific behaviors by demonstration, and then to implement these techniques in a graphical user interface editor.
- Determine the appropriate global layouts for application objects, and the useful parameters to those layouts. Then, to provide capabilities for the user interface designer to specify these layouts by demonstration.
- Design and implement a powerful constraint language that supports the relationships needed by all user interface objects. The constraints must be solved in real time (for each incremental mouse movement).

- Design and implement a very high-level language for describing the contents of the user interface.
- Design and implement the appropriate components of a library that will allow applications to incorporate demonstrational techniques in their user interfaces.
- Determine how to provide existing gesture recognition and touch tablet technology in an application-independent manner, and then to provide these capabilities from the graphical editor and high-level languages.
- Develop and document all parts of the system to a level where they are useful to other projects, and then work with those projects so that the system will be widely used.

Types in Programming

Conduct a broad program of basic research in types in programming. The principal tasks of our work will be to:

- Design, implement, and extend the Forsythe programming language, and study the impact of its conjunctive type system on program development.
- Design, implement, and extend the programming language LEAP, and study the impact of its explicit polymorphic type system on program development.
- Develop the basic proof theory and implementation support for the type and logic specification language LF.
- Consolidate work in category theory on models for type theories, and from this develop practical proof systems for reasoning about programs.

1. IMAGE UNDERSTANDING

The goal of our image understanding research is to develop *basic image understanding techniques* to significantly mitigate a wide spectrum of vision problems that autonomous intelligent systems must face in military and manufacturing applications. Our strategy addresses key issues arising from the interaction between basic vision technology and task-oriented vision systems.

1.1 Advanced Sensing

1.1.1 Camera Calibration and Control

Controlling both camera and lens is central to analyzing texture and image content because such control determines image resolution and hence the resolvable detail level. Automating this process poses a general problem involving modeling, calibration, and control of the camera and lens.

Camera systems with automated zoom lenses are inherently more useful than those with fixed parameter lenses. With a variable parameter lens a camera can adapt to changes or differences in the scenes being imaged or measure properties of the scene by noting how scene image changes as lens parameters are varied. But to be able to use these lenses effectively, we need to model the relationship between lens control parameters and the parameters of the resulting images. While camera calibration has been the subject of much research in machine vision and photogrammetry, for the most part the resulting models and calibration techniques have been for cameras with fixed parameter lenses where the lens imaging process is static. For cameras with automated lenses, the image formation process is a dynamic function of the lens control parameters. The complex nature of the relationships between the control parameters and the imaging process plus the need to calibrate them over a continuum of lens settings makes both the modeling and the calibration of cameras with zoom lenses fundamentally more difficult than that of cameras with fixed parameter lenses. At Carnegie Mellon's Calibrated Imaging Laboratory, we are developing new methods for modeling, calibrating, and controlling automated zoom lenses. Our work has led to the development of camera models that hold calibration across continuous ranges of focus and zoom and to high-precision 3-D shape recovery algorithms from lens focus.

In [Willson and Shafer 91a] we demonstrate the feasibility and effectiveness of using precise multiple-degree-of-freedom camera control to improve the performance of certain vision tasks. We show how traditional single-degree-of-freedom approaches to color imaging and range-from-focus run into significant problems with unmodeled lens behaviors—chromatic aberration in color imaging and focus magnification in range-from-focus. We also present two new multiple-degree-of-freedom approaches to color imaging and range-from-focus techniques that achieve significantly better results than previous methods.

The most important imaging property that we need to model for machine vision is the perspective projection of points from 3-D object space into the 2-D image plane. Critical to developing a perspective projection model for a variable parameter lens is an accurate model of the motion of image center. But before we can address this problem we have to ask the question, ''What is the image center?'' Nearly all methods for machine vision assume that the image center is considered to be the point of intersection of camera optical axis and camera sensing plane. In fact there are many possible definitions of image center, and most real lenses do not have the same coordinates. In [Willson and Shafer 93a] we have identified 16 different ways to define ''image center,'' and developed a taxonomy of image centers based on the number of different camera settings used and on the type of measurements that are made during calibration. By using an appropriate image center for the image property that we are trying to model, we have improved the precision of a standard (Tsai-Lenz) fixed parameter lens calibration from 0.23 $\pm$0.10 pixels RMS error to 0.06 $\pm$0.04 pixels.

In developing dynamic models for cameras with automated lenses, we face the problem that the lens control parameters do not have a one-to-one correspondence with the terms of the camera models. Indeed, we find that every model term is potentially a function on every lens control parameter. To deal with this situation we need to use an iterative empirical approach to modeling and calibration. Using this approach we have produced a dynamic perspective projection camera model that holds calibration across a continuous range of focal lengths (from 45mm to 130mm) and focussed distance (from 1.5m to 2.5m). The dynamic model is based on a standard 11 term (Tsai-Lenz) static camera model. Low order bivariate polynomials are used to characterize the influence of the lens control parameters on the individual terms of the static model. In the final analysis only six terms—the focal length, x and y components of the image center, radial lens distortion coefficient and Tz— need be variable functions of the lens control parameters, while the five remaining model terms are constant. To fit the resulting 96-coefficient dynamic camera model efficiently to the large sets of calibration data that are necessary, we have developed a method of alternating linear and nonlinear optimization. In stereo systems the necessity of an accurate 3-D to 2-D model of perspective projection has traditionally limited the systems to using fixed parameter lenses. With this dynamic camera model, it is now possible to use the versatility of automated zoom lenses in stereo vision applications. Preliminary results for the dynamic camera model appear in [Willson and Shafer 93b].

Based on our new lens models, we have developed new methods that dramatically improve 3-D shape recovery from lens focus and defocus. In the range-from-focus task, we obtain an accuracy of 1 part in 1000 at distances of 1.2m, a fivefold improvement over previously published results for this task. The improvement comes from smoothing the criterion function by applying a polynomial curve in the vicinity of the peak value, where noise becomes a limiting factor to precision. More significantly, for range-from-defocus we employed two improvements: We use an iterative method to overcome the effects of window size on the calculation; and we developed an improved blur model expressed in terms of motor control variables rather than abstract optical idealizations. These improvements have allowed us to make more accurate calibration. Taking just two images with differing foci, we have obtained dense depth maps with a precision better than 1 part in 200, compared to results of 1 part in 75 reported in the literature. With this precision, depth-from-defocus is becoming a viable complement to more established techniques for 3-D shape recovery such as stereo vision.

1.1.2 High-Speed VLSI

Range image information conveys a rich description of the physical world that makes it the leading sensing technology for robotic inspection, manipulation, and control tasks. In a range image, each pixel represents the 3-D distance to a point in the imaged scene. Many techniques for the direct acquisition of range images have been developed, and, of these, lightstripe triangulation methods have proven among the most robust and practical. However, conventional lightstripe sensors are very slow, typically producing less than ten frames of range data per second. This limitation makes them unsuitable for fast recognition, dynamic scene analysis, or real time control applications.

Our work in this area is developing a high-speed, 3-D, imaging system. At the heart of the system is an intelligent VLSI sensor that can acquire 100 to 1000 range frames per second — rates two orders of magnitude better than those achievable with conventional lightstripe rangefinding methods. The sensor uses a novel ‘‘cell-parallel’’ lightstripe algorithm in which a 32×32 array of cells acquires a 1024 pixel range image each millisecond. Each cell determines independently when it sees light reflected back from scene objects. We have measured the accuracy and repeatability of each pixel to be within 0.5mm at 500mm distances (0.1%).

The cell-parallel algorithm is practical only because we can integrate computation and memory functions in a single photoreceptive cell. Employing analog processing techniques has enabled us to build dense cell arrays that perform the rangefinding function.

Calibrating the system

Calibration of this cell-parallel range system poses a unique challenge. The sensor IC currently does not return intensity images of the scene as a byproduct of the range acquisition process. Existing methods for calibrating conventional camera-based lightstripe range systems rely on having intensity images available.

We have developed a calibration procedure that eliminates the need for intensity images. This new method relies on an accurate 3-D positioning device for determining the range system geometry. With the positioner reference objects are manipulated in the 3-D field of view of the range sensor IC in order to determine the sensor-to-world coordinate transform. A mapping between sensor data output values and object position is then measured, completing the calibration process.

We were able to reproduce the data generated by the range system. We positioned a stationary target in the working volume of the system. Output is produced by a single element of the sensor IC over many range image frames; this output is then converted into a histogram. These experiments show standard deviation values $\sigma \approx 34$ out of a total system dynamic range of 3000, $\approx 1\%$ worst-case repeatability for measurements made by the system.

Integrating sensing and processing on integrated circuit substrates holds great promise for developing sophisticated data-acquisition systems. Computation at the point of sensing allows tailoring in a parallel fashion raw sensor data to the needs of higher level system requirements. The ability to acquire data intelligently also means that new sensing methodologies can be developed. One of the most distinguishing features of this research is that we are trying to do more than achieve increased speed by simply implementing known algorithms with VLSI technology, as is the case with using VLSI chips for convolution. Rather, we are demonstrating that by integrating sensing and processing, we have enabled modifications in the basic operational principles of information acquisition (in our case, range imaging) that result in qualitative performance improvements.

Evolving the sensor system

The prototype cell-parallel range-imaging system, based on our smart sensor, employs a fully customized integrated circuit implemented in a 2µ CMOS process. The chip contains 896 smart cells arranged in a 28×32 array and measures 7.8×9.2mm. Each cell contains a photoreceptor, light stripe detection circuitry, and the circuitry to record and read out range data samples.

Our second-generation VLSI range sensor has become operational. The cells of the new design are 40% smaller and they yield increased range image spatial resolution. In addition, a true-peak detector has replaced the thresholding circuit previously used to identify the light stripe. The new peak detection scheme has two important advantages. First, the new design is more sensitive to the stripe. The sensor now operates in the presence of bright indoor ambient lighting, and the increased sensitivity permits is use on a wider variety of objects. In addition, the range sensor chip now provides reflectance information as an artifact of the new peak detection process. The reflectance image is read from the chip along with acquired range data. Pixels of the reflectance image are perfectly aligned with corresponding pixels in the range image.

One of the distinguishing features of this research is the very practical nature of the problem that has been solved. Our sensor technology provides the high frame rates required by the most demanding autonomous robotic systems. The advantages of VLSI computational sensing have been advocated by many. However, few practical sensors of this type have been developed. We are working to move this technology from the laboratory by replicating our cell-parallel range image systems. We hope to soon begin deploying these systems for use by those performing robotic applications research in the DARPA and NSF communities.

We successfully demonstrated the basic range-imaging technology and began adapting our sensor for robotic applications. We used our laboratory sensor to provide data for a 3-D pose estimation system. The application area is robotic assisted surgery. Range images provided by the sensor are used to determine, in real time, the pose of a section of femur bone with respect to a stored 3-D model of the bone. In practice, a hip-replacement procedure would be outlined beforehand by a physician using data from MRI scans. Data from the range-image sensor would then be used to precisely guide the bone milling required during surgery. Actual use of this next-generation sensor technology is crucial to the next phase of the research—facilitating the eventual transfer of the technology, providing feedback for the ongoing basic-sensor research, and demonstrating the practical value of this research program.

1.2 Physics-Based Vision

1.2.1 Understanding Color

Color is a rich carrier of image information and can potentially facilitate identifying and distinguishing each scene object. Yet, this promise is difficult to realize because each object surface can display many gradations in color due to shading, highlights, and illumination. This complexity has made traditional feature detection very difficult and error-prone, leading to later problems in object recognition and description by machine vision.

Low-level vision systems need to account for complex imaging phenomena that produce apparent color: shadows, highlights, interreflections, etc. Our research in color understanding aims to develop an analysis theory that relates camera, lighting, and object properties to image appearance, thereby allowing images to be analyzed so that the vision system can understand what is in the world around it. Our approach is based on modeling the actual illumination and reflection of light in the scene, the surface properties of objects in the scene, the nature of color camera and imaging devices, and the resulting color properties of image data. One of our primary thrusts in this area is modeling the physical laws of color reflection and how they determine color in an image.

We are now studying the importance of local illumination effects upon object appearances. The usual approach in image understanding research is to assume that a single light source illuminates a scene and that the light does not interact among objects. This model is appealing because of its simplicity and global approach. However, in reality, light from one object may reflect onto another, altering the second object appearance. This local effect can cause a dramatic color change over substantial areas of the image. We are also interested in how object properties such as surface roughness and material type affect appearance, especially where interreflection occurs.

We have investigated models of surface reflection to see how they might be extended to include interreflection. We are interested in how the amount of optical roughness affects the appearance of highlights and interreflected highlights. We've compared these extended models with images of interreflections to determine if it is useful for analyzing images. A useful model must be sophisticated enough to predict actual appearance well yet simple enough that we have some hope of recovering the model parameters from images. We took an image containing interreflections and generated a histogram from the brightness levels of its pixels. Previously, we found that such a histogram can be qualitatively related to object properties, including roughness. A very smooth object was observed to have a longer, narrower highlight cluster in the histogram whereas a rougher (optically) object was shown to have a shorter, fatter highlight cluster associated with it. Furthermore, smooth objects were observed to have short, narrow spikes corresponding to secondary highlights (interreflection of primary highlights). We also examined the Torrance-Sparrow model in order to establish a quantitative relationship between these image features and the amount of optical roughness.

Exploiting the physics

The physics of interreflection and color is relatively simple, and has been successfully exploited in computer graphics. However, the methods of computer graphics are not generally invertible for machine vision. Recently, some researchers have shown preliminary methods for identifying interreflected color based on qualitative heuristics, and others have shown quantitative analysis but only for idealized surfaces that have no shininess or highlights.

Our approach is to tackle the most complete model of interreflection, one that describes both matte and shiny reflections, and apply quantitative reflection models from the optics literature (such as Torrance and Sparrow's model). Using this strategy, we have shown that the shape, relative position, and surface roughness of each surface contribute to the distribution of colors seen in the color histogram. The color histogram has identifiable features that are directly related to the inherent color and roughness of each surface and to the geometrical arrangement of surfaces and lighting. Optical laws provide a precise formulation for the relationship between the histogram and the scene, giving a foundation for reliable image analysis based on physics rather than heuristics.

To develop this approach into an algorithm for machine vision, we are studying color images in the Calibrated Imaging Laboratory. This laboratory provides an environment for controlled experimentation with lighting and object properties, using high-precision imaging equipment supplemented by optical measurements from a spectroradiometer. These facilities provide high-quality image data so that careful analysis of color is possible. We need such precise data to validate the theoretical models we are developing, even though the goal is eventually to formulate algorithms to apply to general imaging scenarios.

Information from color variations

Although interreflection increases image complexity, it may also yield additional information about the properties of objects in that image. We want to be able to analyze pictures containing interreflection, instead of treating it as a source of noise to be avoided.

We investigated how our model of reflection and interreflection relates to the apparent distribution of color on an object. One of the key analytic techniques is to arrange color variations in a color histogram. In ''Anatomy of a Histogram'' [Novak and Shafer 91] we present a new way to analyze color appearance for image understanding.

We have found that the three-dimensional color histogram of an object contains many identifiable features, such as linear clusters and planar clusters. Our model shows that the shape and location of these clusters provide valuable information about such scene features as surface roughness, object color, illumination color, and illumination position. We have developed a mathematical framework to relate the histogram features to scene properties in a quantitative way.

Color histograms were first used for image segmentation by grouping similar-colored pixels. We have shown that the clusters in the histogram have a specific length, direction and orientation, which directly relate to many scene properties. A full analysis of the histogram leads to a description of surface roughness and imaging geometry, as well as an improved estimate of illumination color.

We refer to the two linear clusters in the color histogram of a single object as the body reflection cluster and the highlight cluster. The highlight cluster consists of those pixels that correspond to the highlight area of the object, whereas the body reflection cluster consists of those pixels that do not show an appreciable amount of highlight. Estimating illumination from highlights provides a method of color constancy.

However, Klinker's method, which calculated the direction of the highlight cluster and used that as an estimate of the illumination color, did not consider surface roughness, which has a considerable effect upon highlight appearance. For very smooth objects, the amount of body reflection does not vary significantly over the highlight region, so the highlight cluster direction is very close to the illumination color. However for rougher surfaces, the highlight is spread over a larger range of reflectance angles, causing a significant change in body reflection over the area of the highlight. This situation causes the highlight cluster to be skewed away from the illumination color in the direction of the object color. The amount of skewing is dependent upon the object's roughness, and to some degree the imaging geometry. Thus an estimate of these two characteristics is necessary for estimating illumination color from highlights.

Furthermore, an estimate of roughness is useful in itself, since it is an important object characteristic that may be related to manufacturing quality or the amount of wear an object has undergone. We have shown that the length of the highlight cluster is inversely related to the roughness of the object. This is because smoother objects concentrate their surface reflection over a narrow area and the resulting highlight is brighter. Furthermore, we have shown the intersection of the highlight cluster with the body reflection cluster is wider when the object is rougher. This is because the highlight covers a wider range of reflectance angles. The point farthest along the body reflection cluster corresponds to the part of the object with the greatest amount of body reflection, which will be those parts of the object with surface normals pointing towards the light source. A point halfway along the body reflection cluster will correspond to surface normals of $\cos^{-1}(1/2)$ equaling 60 degrees off the light source direction. The point where the highlight and body clusters meet determines the amount of body reflection at the point where the highlight occurs, telling us the angular difference between the light source and camera directions.

We have shown that an image of a single dichromatic object will exhibit combinations of two distinct colors (surface reflection and body reflection) and its color histogram will form a plane in RGB space, and that the clusters in the histogram have a specific length, direction and orientation, which directly relate to many scene properties.

Previously we developed a theory to relate color histograms to scene parameters in a concrete, quantitative way. We are now working to develop an algorithm that can effectively apply the relationships between clusters in a color histogram to the analysis of real images. Our goal is to develop a method that will accept a color histogram as input and determine the surface roughness and illumination geometry. This type of color appearance analysis is a result of applying physics-based techniques to image understanding. The analysis of color histograms allows us to explain color appearance and to exploit it to recover useful information about the world from color images.

1.2.2 Measuring Shape and Roughness by Photometric Sampling

Many industrial inspection tasks require measuring surface shape and roughness. Researchers have developed light-based approaches for objects having specular reflectance characteristics, objects such as polished metals. Other techniques measure the shape of objects that have diffuse reflectance characteristics, things such as matte, painted objects. Many industrially significant surfaces, such as those of solder joints and brushed metals, however, exhibit specular diffuse lobe reflectance characteristics. Research on techniques appropriate to measuring the shape and roughness of such objects has, unfortunately, been scarce. Our work has developed such a technique and began by developing a model that describes specular diffuse lobe reflectance.

We chose to use the photosampler device to illuminate the object under inspection. The photosampler guarantees that we will observe both the specular and diffuse lobe reflectance components. Since the photosampler uses extended light sources, we had to develop an approximate specular lobe reflectance model which incorporated the effects of the extended light sources. This model was developed after extensive simulation. Once this model was obtained, we developed an algorithm which could determine surface shape and surface roughness.

The specular diffuse lobe component is a function of surface roughness, which can be described by a statistical function. Our reflectance model for the specular diffuse lobe component is based on the Beckmann-Spizzichino physical optics model and the Torrance-Sparrow geometrical optics model. Since these models depend on surface roughness as well as on surface orientation, we are able to recover both parameters with our method. We took a set of image brightness values measured at each surface point by using our 3-D photosampler device. The photometric sampling method uses many extended light sources. The entire array of extended sources is scanned by sequentially activating each light source one at a time and by taking an image from a fixed viewing direction. Therefore, the scanning process results in a set of image brightness values. Each brightness value provides one nonlinear image irradiant equation, which contains unknown parameters for surface orientation, reflectance, and surface roughness. The algorithm iteratively solves the set of image irradiant nonlinear equations with respect to these parameters.

Experiments conducted on several rough surfaces show a high accuracy in estimated surface orientations, and a good estimation of surface roughness. In the first experiment, we determined the shape and reflectance of a sand blasted brass cylindrical object. We were able to determine surface orientation with an accuracy of 1.6 degrees in the range of -60 to +60 degrees. For the second experiment we determined the roughness of two microfinish comparators. One had RMS roughness of 20 microinches, the other, 60 microinches. We obtained consistent roughness measurements of our samples over the entire surface of the two microfinish comparator samples; the roughness of the 20 microinch comparator was consistently less than the roughness of the 60 microinch comparator. The third experiment we conducted was to determine the shape of a solder joint on a 10W resistor. The solder joint was textured in appearance and was partly covered with solder flux. We were able to build an elevation map for the solder joint. The elevation map clearly shows the curved solder surface. The fourth experiment we conducted was to detect a surface defect in a thick film of gold deposited on a LSI chip package. The gold film had a rough textured surface, but was not uniformly deposited. Our method detected the nonuniformity of the gold film by measuring the surface roughness. In areas where the film was nonuniform, the roughness value varied.

1.2.3 Measuring Shape and Roughness by Color

In a related project we also addressed the challenge of measuring shape and surface properties —such as albedo and roughness—for objects showing *both* diffuse and specular reflectance characteristics. Such objects, too, are relatively common in industrial settings, and research on relevant techniques is regrettably thin. Our work has led to a method for measuring both surface shape and surface properties simultaneously and to a design for experimental apparatus that can demonstrate the validity of our proposed algorithm.

Color spaces, especially the RGB color space, have been widely used by the computer vision community to analyze color images. Previously, the method has been proposed to separate the body and specular reflection components in images by geometrically clustering a scatter plot of an image in RGB color space. The separated body reflection component is used for segmentation of a color image without segmentation errors that can be caused by highlights. This method assumes that the directions of surface normals in an image are widely distributed in all directions. This assumption guarantees that both the body reflection vector and the specular reflection vector will be visible. Therefore, this algorithm cannot handle cases where only a few planar surface patches exist in an image.

We propose a novel method to analyze a sequence of color images. A series of color images are examined in a four dimensional space, which we call the temporal-color space, whose axes are the three color axes (RGB) and one temporal axis. The term *temporal-color space* implies an augmentation of the RGB color space with an additional dimension that varies with time. This dimension represents the geometric relationship between the viewing direction, illumination direction, and surface normal. We keep the viewing direction and surface normal orientation fixed. We vary the illumination direction with time, taking a new image at each new illumination direction.

The significance of the temporal-color space lies in its ability to represent the change of image color with time, whereas a conventional color space analysis yields the histogram of the colors in an image, only at an instance of time. Conceptually, the two reflection components from the dichromatic reflection model—the specular reflection component and the body reflection component—form two subspaces in the temporal-color space. We developed an algorithm to extract those two fundamental components by principal component analysis using the singular value decomposition technique.

The proposed technique does not require any knowledge about surface reflectance and shape. Therefore, both surface shape and surface properties such as albedo and roughness can be obtained simultaneously without making any assumptions. In contrast to conventional color space analyses, only local information is needed to perform the proposed analysis. In other words, we can recover the surface orientation and reflectance based on color change at each pixel individually. Our method does not depend on the assumption of a global distribution of surface normals in an image. This fact implies that our proposed method has a wide range of applications for industrial inspection problems.

1.3 Computing Shape

We are investigating a new way to represent image data that is serving as a basis for new algorithms for image understanding and is allowing us to combine algorithms in order to understand more realistic images. Computer vision researchers have traditionally used the spatial domain (e.g., *(x,y)* pixel coordinates) in solving their problems, largely because geometric relations play an important role in understanding and describing a scene. By contrast, image processing researchers rely heavily on the spatial frequency domain (e.g., Fourier transforms) for their work, because of the importance of linear filtering and because images often exhibit repetitive structures. Some computer vision problems, however, are best characterized and solved only with a combination of these two techniques. We have been investigating the combined space/frequency image representation for solving these problems. The space/frequency representation shows the local spatial frequency characteristics of each pixel in the image. It is like having a small, 2-D Fourier transform around each pixel.

Our most mature space/frequency research is the analysis of texture in images. Historically, vision algorithms analyzed images by first finding regions of uniformly varying intensity. However, most contiguous regions in an image do not have this property. Instead, some (if not most) regions in images are characterized by similar texture rather than similar intensity. In fact, every visible object appears textured at some level of magnification. Image texture exists in almost every natural image, being caused by fields of grass, gravel roads, brick walls, and everything else with an extended, coherent fluctuation in shape or brightness. Therefore, an absolute prerequisite to competent image understanding is an understanding of texture.

We are studying a way to represent texture by the *spectrogram*, a kind of Fourier transform centered around each pixel in the image. The spectrogram allows the geometric reasoning methods of machine vision to be combined in a single framework with image processing techniques based on Fourier transforms. The spectrogram is an image representation that shows image spatial and local spatial frequency characteristics simultaneously. Our preliminary research showed how various phenomena, including texture, affected the local Fourier transforms of an image. We demonstrated and analyzed the effects of texture boundaries, shape, aliasing, zoom, and focus [Krumm and Shafer 90a, Krumm and Shafer 90b].

In [Krumm and Shafer 92] we developed our ideas for shape-from-texture into a working algorithm. It is based on our mathematical model of the spatial frequency shifts one can expect in an image of a textured surface due to shape. For example, a cobblestone road will appear to have higher spatial frequencies at greater distances, because the cobblestones will appear closer together. These frequency shifts will vary with the slope of the road. We examine these shifts by computing local Fourier transforms at preselected points on an image of a textured surface. Using our model, we can find the surface orientation of textured planes to within about four degrees. Our algorithm requires no texture features, which is significant because finding texture features is a notoriously difficult problem in all texture algorithms that require it. Because we work with a low-level representation of the texture (in the form of the local power spectra), we can account for other, normally confounding effects of imaging such as aliasing and depth-of-field, conditions that arise naturally in any imaging system.

Our most recent effort has been texture segmentation, that is, grouping together regions of an image based on their similar texture. Past segmentation research was based on the assumption that the textured objects are flat and viewed frontally — an unlikely occurrence in a general scene. Past shape-from-texture research was based on the assumption that the similarly textured regions have already been segmented. These two assumptions are mutually exclusive, so there was no hope of applying any current texture understanding system to a realistic image.

We have shown how to solve the combined problem of segmentation *and* shape from texture simultaneously [Krumm and Shafer 94]. This is an important result, because it breaks through the mutually exclusive assumptions that prevented the combination of every other texture segmentation and shape algorithm. Our program can correctly find multiple regions of uniform texture even if the textures are on slanted surfaces. The key is that we explicitly account for 3D shape effects. The local Fourier transforms of pixels on the same 3D surface are similar, within a linear (affine) transform of each other; but, across surfaces, the Fourier transforms differ significantly. Our analysis proceeds by creating several hypotheses about surface orientation based on small regions in the image. Each hypothesis consists of an estimate of the local surface normal and an estimate of what the local frequency distribution of the texture would look like if viewed frontally. This ‘‘frontalized’’ frequency distribution is computed by undoing the effects of the estimated surface normal on the local frequency distribution. Thus, two adjoining regions of similar texture and different depth will have the same hypothesized local frequency distribution. We then merge similar hypotheses to form regions. Our merge criterion is based on the minimum description length principle, which allows us to reduce the number of adjustable parameters (‘‘magic numbers’’) that typically plague most segmentation algorithms. This method has been successfully tested in the Calibrated Imaging Laboratory.

We are also applying spatial frequency analysis to the stereo matching problem. One of the open problems in stereo research is how to match successfully in an image pair areas containing repetitive structure (e.g., a checkerboard pattern). Existing methods often generate many *false matches* in such areas, and then yield gross errors in the resulting depth map. They fail to exploit large-scale effects, such as using the border of the repetitive area to constrain the matching. We have developed a solution for this problem using the image scalogram, a dense sampling of complex Gabor wavelet coefficients. This is a variation of the space/frequency representation that we used for our texture analysis above. The representation has many desirable properties for stereo matching. The dense sampling allows us to focus attention at the most appropriate scales, the phase term gives us disparity estimates with subpixel precision, and the magnitude term gives us a measure of confidence in that estimate. This wavelet representation lets us combine information from many scales simultaneously, so that the final disparity estimate effectively blends low and high frequency information. We have tested our method on several images so far, both real and synthetic, and it compares favorably with existing methods (the three methods studied in the ARPA Stereo Evaluation project and the Kanade-Okutomi variable window refinement technique). In the test images our method successfully matched areas with repetitive structure more often than other methods, and was at least as precise as other methods in those areas where the latter also succeeded.

We predict that several different computer vision algorithms will eventually be combined using the space/frequency representation. We have already shown how it can perform tasks of texture segmentation, shape-from-texture, and stereo. This common representation has allowed us to solve the two texture problems simultaneously, and we have developed a theory to combine this solution with stereo, depth-from-focus, and aliasing. Other research groups have shown how to apply the same representation to problems in optical flow and shape-from-shading.

1.3.1 Recovering Shape and Determining Camera Motion

Sensing the shapes of objects and their motion relative to a camera is of great importance in a wide range of applications, such as autonomous navigation, robotic manipulation, and cartography. This structure-from-motion problem has been a topic of great interest in today’s computer vision research. Unfortunately, as soon as the objects in front of the camera are more than a few focal distances away, the traditional approach of determining shape by taking differences of depth values is too sensitive to image noise to yield satisfactory results. To overcome this problem, we developed a factorization method that combines displacement measurements from many image frames and then directly computes shape and motion, without first having to compute depth as an intermediate step.

The factorization method [Tomasi and Kanade 92a] consists of two steps:

- Identify features and track them from each image to the next, to produce a sequence of *(x,y)* positions representing each feature's apparent trajectory. These values are stored as entries in the measurement matrix.
- From the measurement matrix, recover the camera motion and the three-dimensional locations of the feature points.

The first step is reliable because we use image frames sampled in fine increments so that apparent motion is small. In the second step we use a large number of images and track a large number of features, providing highly redundant measurement data. Also, we have formulated the structure-from-motion problem using orthographic projection in object-based coordinates, rather than perspective projection in camera-centered coordinates. In this formulation the redundancy of information is concisely expressed by the fact that the noise-free measurement matrix has rank 3. Input data noise causes the measurement matrix rank to be not precisely 3, so we use the well-established singular value decomposition (SVD) technique to compute the best rank 3 approximation to the measurement matrix. This technique enables us to reject much of the noise and provides for robust shape and motion recovery.

We completed both the software for feature detection and tracking and the shape and motion computation program. We have been testing the method with images from indoor, outdoor, and controlled laboratory scenes with excellent results. On controlled laboratory images for which ground-truth data are available, typical errors have been between 0.02 and 0.20 degrees for object rotation (the hardest part to recover), and between 0.5 and 2.0% of the object size for shape information. These figures represent significant improvements over numbers reported in recent research literature. In particular the rotation result is an improvement by a factor of about 10%.

The chief limitation of the original method was its assumption of orthographic projection. This assumption is approximately valid for long-range scenarios such as aerial imagery or use of long telephoto lenses, but it becomes rather inaccurate for shorter-range situations such as robot vehicle navigation and terrestrial terrain mapping. Unfortunately, the accurate perspective model leads to complex mathematics, which makes the problem intractable. Therefore, we developed a new model of image projection, called *paraperspective*, that captures the most important aspects of perspective projection, yet leads to relatively simple mathematics. Most importantly, paraperspective projection retains the important property of our factorization method, which is that the measurement matrix has rank 3. Therefore, we can continue to apply the singular-value decomposition that allows our method to obtain such accurate results. Using paraperspective, we have been able to analyze a wider range of image sequences than we could previously [Poelman and Kanade 92].

We have tested the method extensively on synthetic data in order to assess its robustness quantitatively in the presence of noise and to determine its performance under a wide variety of object shapes and motions. Results demonstrate that the method recovers shape and motion very accurately. For an object whose feature points were distributed within a unit cube positioned 30 units away from the camera, the three-dimensional positions the points were recovered with an average of 0.002 units error. We have also tested the method on several real image sequences, including an aerial mapping sequence, and found the method to accurately recover the object shape and camera motion.

We have extended the method to accommodate new feature points entering the image sequence or old feature points disappearing from the image. This allows us to recover shapes of object do not appear in their entirety throughout the sequence - for example, an aerial sequence recording the terrain over a long distance, or a sequence of an object rotating through 360 degrees in which the back of the object is not initially visible. Our method replaces the singular value decomposition with a similar computation that can be applied to a matrix in which some elements are unknown.

We also addressed the other critical part of the factorization method—faster and more reliable feature tracking. The previous tracker took 100 ms per image feature on Sparc 10, and the tracking range (the maximum motion that the tracker can successfully track) was at most 2 pixels per image. We have devised and implemented a hierarchical coarse-to-fine search strategy, as well as efficient coding techniques, for the second version. The new tracker now can track 20 features in 100 ms per image (on Sparc 10). The range of motion that is reliably trackable has been also extended to 50 pixels per image. This performance is fairly close to some real-time applications, such as video-based helicopter control. We are currently reimplementing the code on a Texas Instrument C40 DSP for further improvement.

1.4 Vision Algorithm Compiler

One of the biggest problems in implementing machine vision systems is the need for extensive customized programming. This is very error-prone work and requires an extensive knowledge of geometry, physics, measurement, pattern recognition, and other specialized topics. For this reason programming is a great bottleneck in implementing machine vision systems.

We are developing a Vision Algorithm Compiler (VAC) that can automatically generate a machine vision program to recognize and locate an object based on a solid model. The VAC acts as a skilled and knowledgeable vision programmer to produce the machine vision program.

We decompose the task of object localization into two distinct subtasks:

- Aspect classification (AC): An object image is classified into one of a small number of topologically distinct appearance groups called *aspects*. Each aspect represents a collection of viewpoints from which the object looks roughly the same.
- Linear shape change (LC): Aspect classification yields only a rough estimate of the position and orientation of an object. However, this estimate can be used to initialize a more accurate procedure that then matches model features to observed image features.

Previously, we developed a VAC module for generating the minimum-cost aspect-classification tree.

We began work under this contract by modifying and refining our existing LC module to accommodate a new AC module. To strengthen the capability of a LC module in handling curved objects, we also developed a new representation, the Complex Extended Gaussian Image (Complex EGI). While the traditional EGI only determines the attitude of an object, our complex EGI determines both attitude and position of curved objects. A complex EGI assigns a complex weight associated with each convex surface normal. The normal distance of the surface from the predefined origin is encoded as the phase. The magnitude of the weight is the total visible area of surfaces having the surface normal direction. This approach decouples the orientation and translation determination into two distinct, least-square fitting problems. We have implemented our algorithm to recover object attitude and position based on this complex EGI. We have also applied this technique to synthesized and real data, obtaining accurate results, and completed an error analysis of this method.

1.4.1 Congruent Aspects

One of the phenomena noted while developing the minimum-cost aspect-classification tree module was that some aspects that could not be distinguished from one another based on the values of the available features. We termed these aspects *congruent aspects*. Congruent aspects may result from actual symmetries in objects, but more often derive from apparent symmetries due to the finite number of features used in classification.

The presence of congruent aspects reduces the effectiveness of the AC stage of object localization. Rather than introducing a unique aspect for each observation, the presence of congruent aspects means that a classification may be ambiguous, and the observation may be due to any of several possible aspects. Each candidate aspect must then be used to initialize the LC stage, and a decision procedure employed to select the best pose from among the results. Consequently, the LC stage becomes much more expensive and the results less certain.

Within the paradigm of a vision algorithm compiler, the phenomenon of congruent aspects is unavoidable. A VAC must work with a fixed, finite set of features, and for any such feature set, some collection of objects will exhibit congruent aspects. One possible method for dealing with congruent aspects is to utilize multiple observations from different sensor locations, and combine the information to disambiguate, or resolve, congruent aspects.

We also started investigating how to plan for multiple observations. Our basic idea is to use the known spatial distribution of aspects to resolve congruent aspects. Given an aspect and a sequence of sensor locations, the associated observations can be predicted. Congruent aspects can then be resolved by finding some set of sensor locations for which the observations will differ for each aspect and then matching the actual sequence of observations against the set of possibilities.

We have developed a representation, the observation graph, that relates possible observations to the spatial distribution of aspects. Based on this representation, we began developing a planner for selecting the best sequences of possible observations for resolving congruent aspects. The planner's output is a collection of possible observation sequences represented as a tree, the resolution tree. Our initial implementation of the planner is restricted to planning for 2-D sensor motions.

We then implemented a simulation system to test the resolution planner. The domain chosen for simulation is that of specular objects. From our simulation experiments, we have found that multiple observations were very effective at resolving congruent aspects. In the hundreds of cases tested, no errors in resolution were traceable to the planned sequence of observations. Instead, the errors were due to incorrect aspect classification resulting either from observations near the borders of aspects, or from unstable specular features.

Our research leads us to conclude that multiple observations provide a viable paradigm for object recognition.

1.4.2 VAC for SAR and Specular Image Analysis

One important military application of object recognition is analysis, identification, and localization of manmade objects such as airplanes or tanks using images given by exotic sensors such as IR sensors or SAR sensors. Since the appearances of objects in such images are so different from those in images produced by familiar optical sensors, it is quite difficult for even a human programmer to make object recognition programs. In theory, however, using VAC technologies, recognition programs for such sensors can be automatically generated in the same way as those for optical images (intensity images given by TV cameras). The key is to simply exchange the appearance prediction modules in the VAC.

To validate this idea, we developed a new VAC to recognize airplanes in optical specular images and SAR images [Sato et al. 91]. Analyses of specular and SAR images are essentially equivalent because, at the frequencies used in synthetic aperture radar, the reflections are predominantly specular. During the development of this VAC, we tried many of the original VAC methods, encountered several problems, and solved them in the following ways:

- *Difficulty in prediction of specular features*— Typical CAD modelers, such as Vantage [Ikeuchi and Robert 91], can predict appearances given by a TV camera or a rangefinder successfully. However, it is impossible for such modelers to predict the appearances of specular features. For optical specular images, we developed a sensor simulator [Fujiwara et al. 91] using a physics-based reflectance model [Nayar et al. 91a]. For SAR images, we used the SARSIM simulator developed by The Analytics Sciences Corporation under a different ARPA contract.
- *Instability of specular (SAR) features*— Specular features appear, disappear, and change shape suddenly depending on viewing directions.

Specular features may also appear due to interreflections among near-by surfaces. Features that are highly unstable and dependent upon precise alignments of lights, surfaces, and sensors are called *accidental* features, since they are not generally expected in an image but occur due to accidental alignments. For purposes of recognition, a feature should be easy to detect and extract in an image, and should also be present over a wide range of viewpoints. Specular features are typically easy to detect, but some specular features (the accidental features, in particular) are detectable in only very limited range of viewpoints. In order to classify features into those which are useful for recognition, and those which are not, we introduced two measures: stability and detectability [Sato et al. 91]. The detectability measure reflects how easily the feature can be extracted from images. Using these two measures, we defined evaluation functions to identify features useful for aspect classification.

- *Unreliability of AC*— Specular features are difficult to classify. Misclassifications make the binary decisions for aspect classification, employed by the previous VAC, quite unreliable. For our specular (SAR) VAC, we abandoned the binary decision scheme for classification and introduced a continuous decision making process based on the Dempster-Shafer belief theory. Each specular feature provides a likelihood distribution for viewing directions taken over all possible directions. We developed an aspect classifier which select possible aspects by combining the likelihood distributions of selected features following the Dempster-Shafer formula [Sato et al. 91].
- *Unreliability of LC*— Positions of specular features vary dramatically within an aspect. Thus, any LC technique based on a fixed template cannot be applied. We developed a deformable template that can match observed features during the LC phase [Sato et al. 91]. These deformations also result in costs that are dependent upon the amount of energy required to make the deformation. For each applicable template, the system tries to find an optimum position that minimizes the total energy of the match. By comparing the minimum energy levels achieved for several possible aspects, the system can recognize and localize an object.

We constructed these modules for the specular VAC and generated recognition programs. We also applied to the generated programs to real optical specular images and synthesized SAR images, and demonstrated the applicability of our VAC technology to a wider range of applications.

Real optical specular images

In order to obtain real specular images, we constructed a testbed which consists of a rotation table and a TV camera on which a point light source is mounted. We built several objects including a specular airplane and recorded the images on camera. The original image captured by the camera was 480x512; however, for similarity to SAR images, we have reduced the image size to 64x64. We took 10 real images using this testbed.

The object model for the airplane was represented using our geometric modeler Vantage. The model consists of 10 planar, 12 spherical, 15 cylindrical, and 2 conical primitive specular surfaces. We described the sensor model using camera parameters given by the calibration. The optics of the camera was modeled using the perspective projection.

There were 72 images under the optical specular sensor generated by the sensor simulator module of the VAC. From these images, a collection of specular features were extracted. Among these specular features, the VAC selects using two measures: detectability and stability. The aspect generator module of the VAC chose six specular features as effective for aspect classification. These features were: real fuselage, front fuselage, left wing, right wing, tail fin, and right tail plane. From these 72 images, the template generator module of the VAC generated 72 deformable templates for linear change determination (LC). These compilation process took a few days on a Sun 3/260.

We applied the recognition program to real images generated using the testbed. Total recognition time was around 10-15 minutes on our Sun 3/260. 25% of the total recognition time was consumed by the AC classifier, and the remainder was consumed by the LC matcher. We have applied the recognition program to the 10 real images generated using the testbed. Nine out of ten images were recognized correctly, yielding an accuracy rate of 90%.

Synthesized SAR images

For our work on SAR images, we decided to use synthesized images. The object model for the airplane was similar to the one for optical specular images. It consists of 10 planar, 20 spherical, 30 cylindrical and 40 conical primitive specular surfaces. The SAR simulator (SARSIM), developed by TASC generated synthesized SAR images. In order to obtain one SARSIM image, we had to execute the QKSAR simulator, a simulator to generate instantaneous images for SAR reflections, 100 times.

We generated 36 sample SARSIM images by executing the QKSAR simulator 3600 times. The aspect generator module of the VAC chose 6 effective features: rear fuselage, front right wing, real left wing, front right wing, rear right wing, and front fuselage from these sample images. The template generator module of the VAC also generated deformable templates for LC. Total compilation time was a few weeks of CPU time on a Sun 3/260.

We applied the recognition program to the synthesized images generated by SARSIM. All the images were recognized correctly. The total recognition time was 10-15 minutes on a Sun 3/260. The ratio between AC and LC was the same as for the optical specular images. We have applied the recognition program to 10 synthesized images.

1.4.3 VAC for Recognizing Partially Occluded Objects

We began work on developing an efficient algorithm for object recognition for large object databases and scenes with multiple objects that may be partially occluded. Previously, our work on the VAC has concentrated on localization of unoccluded objects in an image. Partial occlusion of objects and large model databases greatly affect the execution time required for recognition process. Typical approaches to reduce the recognition time include feature grouping and the use of feature adjacency information. Unfortunately, these properties can't be reliably computed in general, and the accuracy of a recognition program using it will be low.

We have designed and implemented a novel object-recognition algorithm. The algorithm does not rely on feature grouping or adjacency information and uses realistic constraints to minimize the number of hypotheses requiring verification. Prior constraints are compiled offline through simulation of the imaging and low-level feature extraction process (the sensor model). Through the simulation process, images of the objects are generated where the correspondence of the model features and image features are known. Using the correspondences, statistics are accumulated to produce conditional probability distributions of the extracted features given the visibility of each model feature in the scene. The statistics provide accurate constraints on the features extracted from the object images. The constraints account for effects of sensing, feature extraction, self occlusion of the object, and feature detectability.

The selection of hypotheses by our algorithm is based on Bayesian posterior estimation of the likelihood of primitive matches with respect to our prior constraints. Our algorithm selects only the most likely hypotheses based on our prior constraints. In effect, we are deciding to only verify the most likely hypotheses — expending computational resources on the possibilities which are, according to our a priori models, most likely. We use the Markov Random Field formalism to represent the posterior probability of the possible sets of matchings and we use the Highest Confidence First estimation technique to compute the most likely set of matchings. The matchings are then ordered in terms of likelihood and verified with both a test and a project operation.

Our most recent effort has been developing an efficient object-recognition algorithm for large object databases and scenes with multiple objects that may be partially occluded. The algorithm does not rely on feature grouping or adjacency information and uses realistic constraints (compiled using a sensor model) to minimize the number of hypotheses requiring verification.

In another effort we have focused on the problem of robust pose refinement (fine localization). Our recognition algorithm produces object hypotheses using a minimal number of feature correspondences. Because of image noise, partial occlusion, and object tolerances, the pose of an object computed using a minimal number of feature correspondences may be inaccurate. An inaccurate pose estimate will often lead to mistakes in the verification decision resulting in inaccurate recognition results.

The basic approach to pose refinement is to perform a local search for the pose which minimizes the error between visible object features and image features. Typical approaches minimize the least squared error between object and image features. Unfortunately, least-squares approaches are very sensitive to outliers. In the pose refinement problem, outliers may be missing object features in the image. These features may be absent because of occlusion or sensing effects. In addition, an object feature may not really be visible in the current pose even though the algorithm thinks it should be; it may be an artifact of approximations for the object feature's visibility computation (common in multiview visibility approximations). With only a few missing image features, the pose estimates acquired using least-squares techniques can often be worse than the initial pose estimate.

We have designed and implemented a technique for robust pose refinement. Our technique provides us with reliable pose estimates in spite of significant outliers from the object model. We employ a nonlinear error function which reduces the effect of outliers on the solution. Minimizing a nonlinear error function is equivalent to finding the *maximum a posteriori* estimate of the pose, assuming some distribution of the errors between object and image features. We have had success using an error distribution similar in shape to a Gaussian distribution but having a relatively higher probabilities for large errors. The increased likelihood of large errors in the assumed distribution tends to reduce the effect of outliers from the model, giving a robust estimate of the true pose of the object.

1.5 Bibliography

[Cerrada et al. 91] Cerrada, C., K. Ikeuchi, L. Weiss, and R. Reddy.
A 3D-Object Reconstruction System Integrating Range-Image Processing and Rapid Prototyping.
Technical Report CMU-RI-TR-90-32, Carnegie Mellon Robotics Institute, December, 1991.

[Delingette et al. 91a]
Delingette, H., Hebert, M., and Ikeuchi, K.
Shape Representation and Image Segmentation Using Deformable Surfaces.
In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June, 1991.

[Delingette et al. 91b]
Delingette, H., Hebert, M., and Ikeuchi, K.
Deformable Surfaces: a Free-Form Shape Representation.
In *SPIE Symposium on Geometric Methods in Computer Vision*. SPIE, July, 1991.

[Delingette et al. 91c]
Delingette, H., Hebert, M., and Ikeuchi, K.
Energy Functions for Regularization Algorithms.
In *SPIE Symposium on Geometric Methods in Computer Vision*. SPIE, July, 1991.

[Delingette et al. 91d]
Delingette, H., Hebert, M., and Ikeuchi, K.
Trajectory Generation with Curvature Constraint Based on Energy Minimization.
In *Proceedings of IEEE/RSI International Workshop on Intelligent Robots and Systems*. IEEE, November, 1991.

[Delingette et al. 92]
Delingette, H., Hebert, M., and Ikeuchi, K.
A Spherical Representation for the Recognition of Curved Objects.
Technical Report CMU-CS-92-214, School of Computer Science, Carnegie Mellon University, November, 1992.
Also appears in Proceedings of the International Conference on Computer Vision, Berlin, May 1993.

[Firschein et al. 93]
Firschein, O., Fischler, M., and Kanade, T.
Creating Benchmarking Problems in Machine Vision: Scientific Challenge Problems.
In *Proceedings of the DARPA Image Understanding Workshop*. DARPA, April, 1993.

[Fujiwara et al. 91] Fujiwara, Y., S. Nayar, and K. Ikeuchi.
Appearance simulator for computer vision research.
Technical Report CMU-RI-TR-91-16, Carnegie Mellon Robotics Institute, August, 1991.

[Gremban and Ikeuchi 92a]
Gremban, K.D., and Ikeuchi, K.
Appearance-Based Vision and the Automatic Generation of Object Recognition Programs.
Technical Report CMU-CS-92-159, School of Computer Science, Carnegie Mellon University, July, 1992.
Also appears in the book, ‘‘Three-Dimensional Object Recognition Systems,’’ A.K. Jain and P.J. Flynn, editors. Elsevier Science Publishers 1993.

[Gremban and Ikeuchi 92b]
Gremban, K.D., and Ikeuchi, K.
Planning Multiple Observations for Object Recognition.
Technical Report CMU-CS-92-146, School of Computer Science, Carnegie Mellon University, December, 1992.
Also appears in Proceedings of the IEEE Conference on Robotics and Automation, Atlanta, May 1993.

[Gruss et al. 91] Gruss, A., Carley, R., and Kanade, T.
Integrated Sensor and Range-Finding Analog Signal Processor.
IEEE Journal of Solid-State Circuits 26(3):184-191, 1991.

[Gruss et al. 92] Gruss, A., Tada, S., and Kanade, T.
A VLSI Smart Sensor for Fast Range Imaging.
In *IEEE/RSJ Int'l Conference on Intelligent Robots and Systems*. IEEE, July, 1992.

[Hebert et al. 91] Hebert, M., Delingette, H., and Ikeuchi, K.
Free-Form Deformable Surfaces for Object Modeling and Recognition.
In *29th Annual Allerton Conference on Communication, Control, and Computing*. Allerton, October, 1991.

[Ikeuchi and Hebert 92]
Ikeuchi, K., and Hebert, M.
Task-Oriented Vision.
In *IEEE/RSJ Int'l Conference on Intelligent Robots and Systems*. IEEE, July, 1992.
Available as Technical Report CMU-CS-91-163. Also appears in *Proceedings of 1990 DARPA Image Understanding Workshop*, September 1990.

[Ikeuchi and Hong 91]
Ikeuchi, K., and Hong, K.S.
Determining Linear Shape Change: Toward Automatic Generation of Object Recognition Programs.
CVGIP: Image Understanding 53(2):154-170, 1991.

[Ikeuchi and Robert 91]
Ikeuchi, K., and Robert, J.C.
Modeling Sensor Detectability with VANTAGE Geometric/Sensor Modeler.
IEEE Transanctions on Robotics and Automation 7(6):771-784, December, 1991.

[Ikeuchi and Sato 91]
Ikeuchi, K., and Sato, K.
Determining Reflectance Properties of an Object using Range and Brightness Images.
IEEE Transactions on Pattern Analysis and Machine Intelligence 13(11):1139-1153, 1991.

[Ikeuchi and Suehiro 92a]
Ikeuchi, K., and Suehiro, T.
Recognizing Human Assembly Tasks.
In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June, 1992.

[Ikeuchi and Suehiro 92b]
Ikeuchi, K., and Suehiro, T.
Towards an Assembly Plan from Observation: Part I: Assembly Task Recognition Using Face-Contact Relations (Polyhedral Objects).
In *Proceedings of IEEE Conference on Robotics and Automation*. IEEE, May, 1992.
Also available as Technical Report CMU-CS-91-167, August 1991.

[Ikeuchi and Suehiro 92c]
Ikeuchi, K., and Suehiro, T.
Towards an Assembly Plan from Observation: Task Recognition with Polyhedral Objects.
In *Proceedings of DARPA Image Understanding Workshop*. DARPA, January, 1992.

[Ikeuchi and Suehiro 93]
Ikeuchi, K., and Suehiro, T.
Task Model for Assembly Plan from Observation System.
Journal of the Robotics Society of Japan 11(2):281-290, 1993.

[Ikeuchi et al. 91] Ikeuchi, K., T. Suehiro, P.Tanguy, and M. Wheeler.
Assembly Plan from Observation.
Annual Research Review 1990 ():37-53, 1991.

[Ikeuchi et al. 93a] Ikeuchi, K., Kawade, M., and Suehiro, T.
Assembly Task Recognition with Planar, Curved, and Mechanical Contacts.
In *Proceedings of IEEE Conference on Robotics and Automation*. IEEE, May, 1993.

[Ikeuchi et al. 93b] Ikeuchi, K., Kawade, M., and Suehiro, T.
Towards an Assembly Plan from Observation, Part III: Assembly Task Recognition (General Case).
In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, May, 1993.

[Kanade 91] Kanade, T.
Computer Vision as a Physical Science.
CMU Computer Science: A 25th Anniversary Commemorative.
In Rashid,
ACM Press, 1991.

[Kanade 92] Kanade, T.
CMU Image Understanding Program.
In *Proceedings of DARPA Image Understanding Workshop*. DARPA, January, 1992.
Also available as Technical Report CMU-CS-91-168.

[Kanade and Bajcsy 93]
Kanade, T., and Bajcsy, R. (eds.).
Computational Sensors.
In *Report from the DARPA Workshop*. DARPA, May, 1993.

[Kanade and Shafer 90]
Kanade, T., and S.A. Shafer.
CMU Image Understanding Program.
In *Proceedings of 1990 DARPA Image Understanding Workshop*. DARPA, September, 1990.

[Kanade et al. 91] Kanade, T., Gruss, A., and Carley, R.
A Very Fast VLSI Rangefinder.
In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, April, 1991.

[Kanade et al. 92] Kanade, T., Okutomi, M., and Nakahara, T.
A Multiple-baseline Stereo Method.
In *Proceedings of DARPA Image Understanding Workshop*. DARPA, January, 1992.

[Kanade et al. 93] Kanade, T., Poelman, C.J., and Morita, T.
A Factorization Method for Shape and Motion Recovery.
Transactions of the Institute of Electronics, Information and Communication Engineers D-II , August, 1993.
In Japanese.

[Kang and Ikeuchi 90]
Kang, S.B., and K. Ikeuchi.
3-D Object Pose Determination Using Complex EGI.
Technical Report CMU-RI-TR-90-18, CarnegieMellonRoboticsInstitute, October, 1990.

[Kang and Ikeuchi 91]
Kang, S.B., and K. Ikeuchi.
Determining 3-D Object Pose Using the Complex Extended Gaussian Image.
In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, June, 1991.

[Kang and Ikeuchi 92]
Kang, S.B., and Ikeuchi, K.
Grasp Recognition Using the Contact Web.
In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, July, 1992.

[Kang and Ikeuchi 93a]
Kang, S.B., and Ikeuchi, K.
Temporal Segmentation of Tasks from Human Hand Motion.
Technical Report CMU-CS-93-150, School of Computer Science, Carnegie Mellon University, April, 1993.

[Kang and Ikeuchi 93b]
Kang, S.B., and Ikeuchi, K.
A Grasp Abstraction Hierarchy for Recognition of Grasping Tasks from Observation.
In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, July, 1993.

[Kang and Ikeuchi 93c]
Kang, S.B., and Ikeuchi, K.
The Complex EGI: A New Representation for 3-D Pose Determination.
IEEE Transactions on Pattern Analysis and Machine Intelligence , July, 1993.

[Kang and Ikeuchi 93d]
Kang, S.B., and Ikeuchi, K.
Toward Automatic Robot Instruction from Perception: Recognizing a Grasp from Observation.
IEEE International Journal of Robotics and Automation , 1993.
To appear.

[Kemmotsu and Kanade 93]
Kemmotsu, K., and Kanade, T.
Uncertainty in Object Pose Determination with Three Light-Stripe Range Measurements.
Technical Report CMU-CS-93-100, School of Computer Science, Carnegie Mellon University, January, 1993.
Also appears in Proceedings of the 1993 IEEE International Conference on Robotics and Automation, May 1993.

[Kiuchi and Ikeuchi 92]
Kiuchi, T., and Ikeuchi, K.
Determining Surface Roughness and Shape of Specular Diffuse Lobe Objects Using Photometric Sampling Device.
In *Proceedings of IAPR-92 Workshop on Machine Vision Applications*. IAPR, December, 1992.
Also appears in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, May 1993.

[Krumm and Shafer 90a]
Krumm, J., and S.A. Shafer.
Local Spatial Frequency Analysis for Computer Vision.
In *Proceedings of 1990 DARPA Image Understanding Workshop*. DARPA, September, 1990.

[Krumm and Shafer 90b]
Krumm, J., and S.A. Shafer.
Local Spatial Frequency and Analysis of Image Texture.
In *Proceedings of Third International Conference on Computer Vision (ICCV '90)*. , December, 1990.

[Krumm and Shafer 91]
Krumm, J., and S.A. Shafer.
Sampled-grating and crossed-grating models of moire patterns from digital imaging.
Optical Engineering 30(2):195-206, 1991.

[Krumm and Shafer 92]
Krumm, J., and Shafer, S.
Shape from Periodic Texture Using the Spectrogram.
In *Proceedings of the 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, June, 1992.
Also available as Technical Report CMU-RI-TR-91-29.

[Krumm and Shafer 93]
Krumm, J., and Shafer, S.
Segmenting Textured 3D Surfaces Using the Space/Frequency Representation.
Technical Report CMU-RI-93-14, Carnegie Mellon Robotics Institute, April, 1993.

[Krumm and Shafer 94]
Krumm, J. and S.A. Shafer.
Segmenting 3D Textured Surfaces Using the Space/Frequency Representation.
Spatial Vision , 1994.
To appear.

[Nakahara and Kanade 93]
Nakahara, T., and Kanade, T.
Experiments in Multiple-Baseline Stereo.
Technical Report CMU-CS-93-102, School of Computer Science, Carnegie Mellon University, August, 1993.

[Nayar et al. 91a] Nayar, S.K., K. Ikeuchi, and T. Kanade.
Surface reflection: physical and geometrical perspectives.
IEEE Transactions on Pattern Analysis and Machine Intelligence 13(7):661-634, July, 1991.
Also appeared in *Proceedings of 1990 DARPA Image Understanding Workshop*, September 1990.

[Nayar et al. 91b] Nayar, S., K. Ikeuchi, and T. Kanade.
Recovering Shape in the Presence of Interreflections.
In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*. IEEE, April, 1991.

[Nayar et al. 91c] Nayar, S.K., Ikeuichi, K., and Kanade, T.
Shape from Interreflections.
International Journal of Computer Vision 6(3):173-195, 1991.
Also appeared in *Proceedings of Third International Conference on Computer Vision (ICCV'90)*, December 1990.

[Novak and Shafer 91]
Novak, C.L., and S.A. Shafer.
Anatomy of a color histogram.
Technical Report CMU-CS-91-203, School of Computer Science, Carnegie Mellon University, 1991.
Also in Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, June 1992.

[Novak and Shafer 92]
Novak, C.L., and Shafer, S.
Estimating Scene Properties from Color Histograms.
Technical Report CMU-CS-92-212, School of Computer Science, Carnegie Mellon University, November, 1992.

[Novak and Shafer 93]
Novak, C.L., and Shafer, S.
A Method for Estimating Scene Parameters from Color Histograms.
Technical Report CMU-CS-93-177, School of Computer Science, Carnegie Mellon University, July, 1993.

[Okutomi and Kanade 91]
Okutomi, M., and T. Kanade.
A Multiple-Baseline Stereo.
In *Proceedings of IEEE Robitics and Automation Conference*. IEEE, April, 1991.
Also available as Technical Report CMU-RI-TR-90-189.

[Okutomi and Kanade 92]
Okutomi, M., and Kanade, T.
A Locally Adaptive Window for Signal Matching.
International Journal of Computer Vision 7(2):143-162, 1992.
Also appeared in *Proceedings of Third International Conference on Computer Vision (ICCV '90)*, December 1990.

[Poelman and Kanade 92]
Poelman, C., and Kanade, T.
A Paraperspective Factorization Method for Shape and Motion Recovery.
Technical Report CMU-CS-92-208, School of Computer Science, Carnegie Mellon University, 1992.

[Reece and Shafer 91a]
Reece, D., and Shafer, S.
A Computational Model of Driving for Autonomous Vehicles.
Technical Report CMU-CS-91-122, School of Computer Science, Carnegie Mellon University, April, 1991.
Also in ''Transportation Research-A'', Vol. 27A, No.1, pp. 23-50. Pergamon Press Ltd. 1993.

[Reece and Shafer 91b]
Reece, D., and Shafer, S.
Active Vision at the System Level for Robot Driving.
In *AAAI-91 Fall Symposium on Sensory Aspects of Robotic Intelligence*. AAAI, 1991.
Also in Proceedings of the DARPA Image Understaning Workshop, January 1992.

[Reece and Shafer 91c]
Reece, D., and Shafer, S.
Using Active Vision to Simplify Perception for Robot Driving.
Technical Report CMU-CS-91-199, School of Computer Science, Carnegie Mellon University, November, 1991.

[Reece and Shafer 92]
Reece, D., and Shafer, S.
Planning for Perception in Robot Driving.
In *Proceedings of the DARPA Image Understanding Workshop*. DARPA, January, 1992.

[Sato and Ikeuchi 92]
Sato, Y., and Ikeuchi, k.
Temporal-Color Space Analysis of Reflection.
Technical Report CMU-CS-92-207, School of Computer Science, Carnegie Mellon University, November, 1992.
Also in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, June 1993.

[Sato et al. 90] Sato, H., S. Nayar, and K. Ikeuchi.
Extracting Shape and Reflectance of Glossy Surfaces by Using 3D Photometric Sampling Method.
In *Proceedings of MVA '90: IARP Workshop on Machine Vision Applications*. IARP, November, 1990.

[Sato et al. 91] Sato, K., K. Ikeuchi, and T. Kanade.
Model based recognition of specular objects using sensor models.
In *Proceedings of the IEEE workshop on Directions in Automated CAD-based Vision*, pages 2-10. IEEE, 1991.
Also in ''CVIP: Image Understanding,'' 1992, 55(2):155-169.

[Shafer et al. 93] Shafer, S., Kanade, T., and Ikeuchi, K.
Image Understanding Research at CMU.
In *Proceedings of the DARPA Image Understanding Workshop*. DARPA, April, 1993.

[Solomon and Ikeuchi 91]
Solomon, F., and Ikeuchi, K.
Extracting the Shape and Roughness of Specular Lobe Objects using Four Light Photometric Stereo.
Technical Report CMU-RI-TR-91-28, Carnegie Mellon Robotics Institute, 1991.
Also in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR-92), June 1992.

[Solomon and Ikeuchi 92]
Solomon, F., and Ikeuchi, K.
Inspecting Specular Lobe Objects Using Four Light Sources.
In *Proceedings of IEEE Conference on Robotics and Automation*. IEEE, May, 1992.

[Suehiro and Ikeuchi 91]
Suehiro, T., and Ikeuchi, K.
Towards an Assembly Plan from Observation: Fine Localization Based on Face Contact Constraints.
Technical Report CMU-CS-91-168, School of Computer Science, Carnegie Mellon University, 1991.

[Suehiro and Ikeuchi 92]
Suehiro, T., and Ikeuchi, K.
Towards an Assembly Plan from Observation, Part II: Fine Localization Based on Face Contact Relations.
In *IEEE/RSJ Int'l Conference on Intelligent Robots and Systems*. IEEE, July, 1992.

[Tomasi and Kanade 90]
Tomasi, C., and T. Kanade.
Shape and Motion from Images Streams: a Factorization Method 1. Planar Motion.
Technical Report CMU-RI-TR-90-166, Carnegie Mellon Robotics Institute, September, 1990.

[Tomasi and Kanade 91a]
Tomasi, C., and T. Kanade.
Shape and Motion from Image Streams: a Factorization Method-Part 3 Detection and Tracking of Point Features.
Technical Report CMU-CS-91-132, School of Computer Science, Carnegie Mellon University, April, 1991.

[Tomasi and Kanade 91b]
Tomasi, C., and Kanade, T.
Factoring Image Sequences into Shape and Motion.
In *Proceedings of IEEE Workshop on Visual Motion*. IEEE, October, 1991.

[Tomasi and Kanade 92a]
Tomasi, C., and Kanade, T.
The Factorization Method for the Recovery of Shape and Motion from Image Streams.
In *Proceedings of the DARPA Image Understanding Workshop*. DARPA, January, 1992.

[Tomasi and Kanade 92b]
Tomasi, C., and Kanade, T.
Shape and Motion from Image Streams under Orthography: a Factorization Method.
International Journal of Computer Vision 9(2):137-154, 1992.

[Tomasi and Kanade, 92]
Tomasi, C., and Kanade, T.
Shape and Motion from Image Streams: A Factorization Method - Parts 2, 8, 10: Full Report on the Orthographic Case.
Technical Report CMU-CS-92-104, School of Computer Science, Carnegie Mellon University, January, 1992.

[Wheeler and Ikeuchi 92]
Wheeler, M., and Ikeuchi, K.
Towards A Vision Algorithm Compiler for Recognition of Partially Occluded 3-D Objects.
Technical Report CMU-CS-92-185, School of Computer Science, Carnegie Mellon University, November, 1992.

[Wheeler and Ikeuchi 93]
Wheeler, M., and Ikeuchi, K.
Sensor Modeling, Markov Random Fields, and Robust Localization for Recognizing Partially Occluded Objects.
In *Proceedings of the DARPA Image Understanding Workshop*. DARPA, April, 1993.

[Willson and Shafer 91a]
Willson, R.G., and S.A. Shafer.
Dynamic Lens Compensation for Active Color Imaging and Constant Magnification Focusing.
Technical Report CMU-RI-TR-91-26, Carnegie Mellon Robotics Institute, November, 1991.

[Willson and Shafer 91b]
Willson,R., and S.A. Shafer.
Active Lens Control for High Precision Computer Imaging.
In *Proceedings of 1991 IEEE International Conference on Robotics and Automation*. IEEE, April, 1991.

[Willson and Shafer 92]
Willson, R., and Shafer, S.
Precision Imaging and Control for Machine Vision Research at Carnegie Mellon University.
Technical Report CMU-CS-92-118, School of Computer Science, Carnegie Mellon University, March, 1992.
Also presented at SPIE Conference on High Resolution Sensors and Hybrid Systems, San Jose CA, February 1992.

[Willson and Shafer 93a]
Willson, R., and Shafer, S.
What is the Center of the Image?.
Technical Report CMU-CS-93-122, School of Computer Science, Carnegie Mellon University, April, 1993.
Also in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, June 1993.

[Willson and Shafer 93b]
Willson, R.G. and S.A. Shafer.
A Perspective Projection Camera Model for Zoom Lenses.
In A. Gruen and H. Kahmen (editors), *Proceedings of Second Conference on Optical 3-D Measurement Techniques*, pages 149-158. Institute of Geodesy and Photogrammetry, Federal Institute of Technology (ETH) Zurich, Zurich, Switzerland, October, 1993.

[Xiong and Shafer 93]
Xiong, Y., and Shafer, S.
Depth From Focusing and Defocusing.
Technical Report CMU-RI-93-07, Carnegie Mellon Robotics Institute, 1993.
Also in Proceedings of DARPA Image Understanding Workshop, April 1993, and Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, June 1993.

2. VLSI RESEARCH

2.1 Project Overview

The research supported under this contract had two major thrusts: programming and compiling for parallel machines, and formal methods for verifying digital hardware. In both areas our research ranged from developing underlying theoretical foundations to implementing and evaluating our ideas experimentally.

Goals for the research included:

- Developing effective primitive operations, languages, and compilation tools to map data parallel programs onto a range of high performance machines.
- Implementing and evaluating specific parallel algorithms on a number of different machines.
- Developing methodologies and tools that will allow digital designers to specify formally desired system behaviors and test designs for adherence to specification.
- Developing efficient techniques for analyzing and verifying finite state automata representations of digital systems.

2.2 Summary of Results

2.2.1 Portable Parallel Programming

Our goal in parallel computing is to develop programming models, languages, and compilers that will allow programs to be written independent of the machine they will be run on, and then be automatically and efficiently mapped onto a variety of different machine types. In support of this goal, we have pursued the following efforts:

- *Developing data-parallel primitives*— Our work builds on the notion that data-parallel models are the most effective way of achieving parallelism that scales to machines with a large number of processors. However, selecting a good set of data-parallel primitives is critical to the expressive power of a data-parallel model. In particular, we have found that a set of scan primitives are very important, as is some way of nesting data parallel constructs.
- *Developing a language for nested parallelism*— Rather than trying to extract parallelism out of programs expressed in conventional languages, we developed a new language based on a set of parallel primitives.
- *Implementation and experiments on parallel algorithms*— We looked at particular problems, such as sorting, and compared the implementation of different algorithms for these problems on a variety of parallel architectures. From these implementations and experiments on them we determined how well algorithms can port across different machine classes, and the performance characteristics of different machines.

- *Development of runtime support for parallel computing*— To reach our goal of porting programs across multiple machines, we must provide a common base of capabilities on each of the target machines. We implemented this base as a set of runtime libraries supporting our data-parallel primitives.
- *Theoretical foundations of parallel programming*— We have been investigating the theoretical foundations of parallel programming, with particular emphasis on deterministic, functional parallel programming. One aim of this research is the development of a novel parallel programming language. This language would allow the programmer to express, at a high level of abstraction, the degree of parallelism intended in his program. The language would thus be well suited to programming parallel algorithms in a wide variety of applications.

The following list highlights the concrete results of this research:

- A fully functional implementations of the NESL language— As well as being used here at Carnegie Mellon, NESL has been used by researchers at Berkeley, Duke, MIT, UNC, and Thinking Machines. It has also been used for teaching courses on parallel algorithms here and at Dartmouth.
- A **C** library of Vector operations and its implementation on the CM2, C90 and CM5— As well as serving as the back end to NESL, it has been used as the back end for Proteus (a joint project between Duke University, UNC and the Kestrel Institute).
- An understanding of many issues of implementing a portable programming language for massively parallel machines— Of special importance are the techniques used for efficiently implementing nested-parallelism.
- A collection of techniques for efficiently implementing parallel algorithms— Based on these techniques we have developed the fastest implementations of sorting on the Connection Machine CM2, the Cray Y-MP C90 and the Connection Machine CM5. We also have developed the fastest general-purpose implementation of sparse-matrix vector multiply, and of linear recurrences for the Cray C90.

NESL Language and Implementation

We designed and implemented a machine-independent programming language, NESL, for massively parallel computers. The project involved language design, implementation, and analysis. Our goal was to design a portable parallel language that is well suited for problems on irregular and dynamic data structures as well as regular and static ones. We implemented a prototype of the language on three parallel machines: the CM2 Connection Machine, the Cray Y-MP C90, and the CM5 Connection Machine. We also studied the performance of our implementation and gained useful insights from the results.

The language is based on the idea of nested data-parallelism. Nested data-parallel languages provide hierarchical data structures, in which elements of an aggregate data structure may themselves be aggregates, and support the parallel application of parallel functions to multiple sets of data. Nested data-parallel languages maintain the advantages of data-parallel languages (fine-grained parallelism, a simple programming model, and portability) while being better suited than flat data-parallel languages for describing algorithms on irregular data structures. On the other hand, compiling nested data-parallel languages to run efficiently is more complicated. NESL is the first language whose implementation supports nested data-parallelism.

Our implementation of NESL is based on compiling to a common intermediate language called VCODE, also designed as part of the project. VCODE, in turn, can either be compiled into **C** threads or interpreted using a library of vector operations called CVL. Our implementations on the CM2 Connection Machine, the Cray Y-MP C90, and the CM5 Connection Machine are based on the interpreter. Because the VCODE operations are on aggregates, the interpretive overhead is not serious if the aggregates are large such that the computation cost dominates the overhead. Our **C**-threads implementation is designed to run on shared memory MIMD multiprocessors (to date we have only run it on the Encore Multimax).

We have reported benchmark results for NESL on the CM2, the Y-MP, and on serial workstations. On these benchmarks we were able to get within a factor of two of machine specific-code in most cases (as mentioned, the problems need to be large to overcome the interpreter overhead). We are currently working on benchmarks results for the CM5. We have also implemented some relatively large programs in NESL, including an algorithm for finding spectral and geometric separators of finite-element meshes, an iterative finite-element solver, and an algorithm for simulating Ising-spin models using the Swendsen-Wang algorithm.

Implementation Experiments on Parallel Machines

We have made an extensive study of sorting on parallel machines, trying many combinations of algorithm and computer. We picked sorting since it is one of the more basic problems, but is by no means trivial—there have been hundreds of parallel sorting algorithms suggested in the literature. Sorting also has many applications, and understanding the practical issues of sorting might lead to a better understanding of related problems such as searching.

We have implemented three parallel sorting algorithms on the Connection Machine CM2: Batcher's bitonic sort, a parallel radix sort, and a sample sort similar to Reif and Valiant's flashsort. We have also evaluated the implementation of many other sorting algorithms proposed in the literature. Our experiments show that the sample sort algorithm, which is a theoretically efficient, ''randomized'' algorithm, is the fastest of the three algorithms on large data sets. On a 64K-processor CM2, our sample sort implementation can sort 32×10^6 64-bit keys in 5.1 seconds, over 10 times faster than the CM2 library sort. Our implementation of radix sort, although not as fast on large data sets, is deterministic, much simpler to code, stable, faster with small keys, and faster on small data sets (few elements per processor). Our implementation of bitonic sort, which is pipelined to use all the hypercube wires simultaneously, is the least efficient of the three on large data sets, but is the most efficient on small data sets, and is considerably more space efficient. This research was joint work with Charles Leiserson at MIT, Bruce Maggs at NEC Research, Greg Plaxton at University of Texas, and Steven Smith at Thinking Machines.

We also implemented an efficient fully vectorized and parallelized radix sort on the Cray Y-MP. This involved modifying the parallel algorithm we used on the CM2 and showing how it could be vectorized. On one processor of the Cray Y-MP, our sort is over five times faster on large sorting problems than the optimized library sort provided from Fortran. On eight processors we achieve an additional speedup of almost five, yielding a routine over 25 times faster than the library sort. On one processor, on large sorting problems, our sort is also five times faster than a fully vectorized implementation of quicksort, and over 10 times faster than a vectorized version of bitonic sort.

We have implemented Batcher's bitonic sort on the Intel iWarp, making use of logical channels to support an embedded hypercube network (required by bitonic sort) on the physical torus of the iWarp. The algorithm sorts one million keys in 0.55 seconds (1.8 million keys per second). To map the binary hypercube onto the torus, a maximal subset of connections is mapped directly to links using Gray code mappings. The remaining hypercube connections are set up using iWarp pathways as indirect links. A programmer can use a mapped topology in his or her algorithms as if it were a physical topology. For a binary six-cube emulated in an eight-by-eight torus, one physical link is shared by at most two connections. The implemented bitonic sort algorithm relies on a radix sort to sort the elements locally within a processor and then continues to merge the sequences between the 64 processors using bitonic merges on the embedded hypercube. The profiling data show a very good balance between local sorting (computation) and merging (communication) for a wide range of problem sizes. The balanced communication architecture and the use of pathways resulted in an extremely simple and efficient implementation of parallel sorter on our iWarp machines. The execution times scale linearly over wide range of problem sizes, including smaller ones.

In addition to sorting algorithms we have developed new approaches to sparse-matrix computations on vector multiprocessors. Our technique is based on the efficient implementation of segmented operations. A segmented operation views a vector as broken into contiguous segments, each potentially a different size. The segmented operations can then function on each segment independently, such as summing the values in each segment. These segmented operations can be implemented on vector-multiprocessors such that the running time is not affected by the size of the segments—the time only depends on the total number of segments and the total number of elements in a vector. This feature makes the operations well suited for the use in irregular sparse computations. We have developed algorithms for performing sparse matrix-vector multiplication and solving sparse triangular systems, and we have implemented these algorithms on the Cray Y-MP C90. Our approach provides excellent performance while requiring only a modest amount of extra storage and preprocessing. We are in the process of comparing the performance of our approach with other formats (such as Compressed Sparse Row, ITPACK, and Jagged Diagonal) on matrices from the Harwell-Boeing collection. On 16 processors we get over four gigaflops on very sparse and irregular matrices with almost no preprocessing (the preprocessing requires no more time than a single matrix-vector multiply) and only 1/60 additional memory.

Additional Data-Parallel Primitives

In earlier work we demonstrated the expressive power of *segmented scan* operations for diverse computational tasks, including sorting, searching, and image processing. We continued this line of research by investigating additional primitives. One class we found particularly powerful were the `match` and `move` operations, first proposed by Gary Sabot. For each of the elements of two vectors of keys, `match` constructs a communication path wherever two keys are equal. The `move` operator sends data through these paths. We have found the `match`/`move` paradigm ideal for expressing a number of algorithms operating on irregular data structures such as graphs and sparse matrices.

`Match` and `move` may be implemented on parallel and vector computers using either sorting or hashing techniques to match equal keys. We implemented and investigated both approaches for the Connection Machine CM2, and for the Cray Y-MP.

Implementing hashing-based match on the Cray Y-MP required the development of a vectorized ‘‘multiprefix’’ operator for that machine. The multiprefix operation generalizes conventional histogram calculation by recording the value of each bucket before it is incremented. Because histogram calculation implies that multiple updates are made to each bucket, this operation is not easily vectorized. Our technique breaks the update loop into four phases that each guarantee that write conflicts do not occur. Thus, each phase may be completely vectorized. We showed how the multiprefix operator by itself can be used to create a very fast integer sorting algorithm, and that it can be used for the multiplication of a sparse matrix by a vector. We submitted our sorting results to the NAS parallel benchmarks committee, where our novel approach achieved speeds exceeding those achieved by other sophisticated vectorizing compiler technologies on the Cray Y-MP. Our sparse-matrix, vector multiplication subroutine achieves performance comparable to those of the standard Fortran libraries.

Our results on the Cray Y-MP show that, for problems involving sparse matrices, algorithms using `match` and `move` based on hashing and multiprefix can achieve performance comparable to that of more traditional algorithms. High performance is ensured by fully vectorizing all operations involving hashing and multiprefix calculation. This unconventional approach will also lend itself to using multiple processors on the Cray in a manner that is transparent to the user of the match and move primitives.

Theoretical Foundations of Parallel Programming

We have also investigated the semantic foundations of parallel programming, with particular emphasis on the development of mathematical models that can be used to support reasoning about various important classes of program behavior. We have focussed mainly on three paradigms for parallel programming: dataflow networks, shared variable concurrency, and communicating processes.

Dataflow networks, when given a suitable semantics, provide an abstract model for VLSI systems. A VLSI design may be represented as a network whose nodes model circuit components like transistors or registers and whose arcs represent communication paths. We showed that one can use techniques and results from denotational semantics (concerning fixed point theory and its use in describing the behavior of recursive processes) to justify the use of certain retiming operations (due to Leiserson and Saxe) in the development of systolic arrays. Similarly, rather elementary ideas and results from the theory of programming languages can be used to show the correctness of pipelining and to establish whether or not one may safely replace part of a circuit design by another without affecting overall behavior. This kind of analysis is important because it supports a ‘‘compositional’’ approach to circuit design and verification.

A shared-variable, concurrent language permits parallel execution of programs that interact by updating and reading a common memory. It is well known that because of the complex ways in which parallel ‘‘threads’’ may interact, it can be extremely difficult to write parallel programs whose behavior meets a desired specification. There are several important kinds of property commonly used in specifications for such programs, including partial and total correctness, deadlock-freedom, and more general safety and liveness properties. A parallel program deadlocks if it gets stuck during execution, unable to make progress towards termination, typically because it is waiting for some other parallel process to do something first. We have developed a proof methodology for establishing correctness assertions of the form $\{P\}C\{Q\}[R]$, interpreted as saying that whenever C is executed from an initial state satisfying P, if it terminates successfully, Q will be true, and if it deadlocks R will be true. For instance, when R is false this is a combination of the traditional notion of partial correctness (introduced by Hoare for sequential programs) with deadlock-freedom. We have designed a syntax-directed (compositional) proof system suitable for proving such assertions and have used it to prove correctness of several examples from the literature on program proofs, including Peterson’s mutual exclusion algorithm and a concurrent program for partitioning a set.

We have developed new, mathematically simple models of parallel programs, designed at exactly the right level of abstraction to support compositional reasoning about various notions of program behavior. Earlier semantic models have tended to be complex and difficult to use and failed to validate a number of natural and intuitive programming laws. We have worked on two paradigmatic cases: a shared variable parallel language, in which parallel programs interact by reading and writing a common memory, and a language of communicating processes, in which parallel programs have disjoint local memories and interact by synchronized message passing. Our models incorporate an elegant account of fairness, corresponding to the natural assumption that when running parallel processes no individual process is delayed forever. Technically, we have developed ‘‘fully abstract’’ semantic models in each case, so that the semantics distinguishes between two program fragments precisely when there is a program context in which the fragments can induce different behavior. These ideas and results can be adapted to different parallel programming paradigms, such as the form of synchronized message-passing exemplified by CSP or the Ada rendezvous mechanism.

In our work on intensional semantics we have developed a mathematical model of functional programs in which the meaning of a program conveys not just what function the program computes but also the computation strategy (such as order of evaluation, efficiency, degree of parallelism). This approach permits reasoning both about traditional forms of program correctness and about intensional properties related to computation strategy. The model makes appropriate distinctions between different programs for the same function, based on the differences in computational strategy. One may also use this model to give an intensional semantics to dataflow networks, gaining insight into how the network computes its results as well as what results it produces. We hope that this may form the basis for the development of network design transformations that preserve the functionality of a network while improving its efficiency.

2.2.2 Formal Hardware Verification

Our long-term goal in this research is to develop highly automated tools that can mathematically verify that a system design meets its specification at varying levels of abstraction from transistor circuits at the lowest to communicating processes at the highest. We believe that such tools have the potential to vastly improve the quality and reduce the design time for complex digital systems.

Approaches to Formal Verification

In formulating approaches to verification we have identified two distinct classes of systems, differing in their high level view of system operation. We believe these two classes call for different verification strategies in terms of how the desired behavior is specified and how system correctness should be established.

A *sequential processor* conceptually executes a single sequence of operations. The processor has a user-visible state, and the effect of each operation is to modify this state. For example, a microprocessor executes a single sequence of instructions, each modifying the state of the CPU registers and the processor memory. Note that the underlying implementation may have complex temporal behavior due to pipelining, multiple functional units, and superscalar operation. The goal in verifying such a system is to show that, despite these complexities, the overall behavior corresponds to the single sequence view of the specification.

A *reactive system* consists of a number of independent agents communicating and synchronizing according to some protocol. Rather than describing the exact behavior of such a system, the specification consists of a set of general properties that the agents and the protocol must fulfill. For example, a distributed, shared-memory system contains a number of bus controllers, memories, and caches interacting according to a cache coherency protocol. The specification states that the effects of read and write operations must satisfy some general consistency properties without fixing the exact system behavior.

In both cases we have exploited the power of Binary Decision Diagrams (BDDs) to represent and analyze symbolically the behavior of circuits and finite state machines that far exceed the capabilities of other approaches.

Sequential Processor Verification

A large class of digital systems, including microprocessors, digital signal processors, memories, and data paths are generally described in terms of how each operation modifies a conceptual system state. Their desired behavior is generally deterministic and (almost) completely specified. On the other hand their implementations are often both large and conceptually complex. They are implemented using techniques such as pipelining and caching so that the low-level system operation and state representation is quite different from that implied by the high-level specification. These complexities often lead to design errors, due to unexpected interactions between logically independent operations. The sophistication and complexity of pipelined systems is increasing greatly with the advent of high-performance, superscalar processors. Thus we perceive a significant need for formal verification tools that can handle large scale, temporally-complex processors.

To verify such systems, we believe that a simulation-based methodology is most appropriate. Unlike conventional simulation, where only a minute fraction of the possible combinations of operation and data are evaluated, our approach applies a form of symbolic simulation to cover a large set of operating conditions in a single simulation run. In this project we developed an approach, called *symbolic trajectory evaluation*, that combines the capabilities of symbolic, switch-level simulation with limited forms of temporal analysis. Such a verifier can prove the correspondence between systems operating at different levels of temporal abstraction, such as between the instruction execution level of a microprocessor and the clock-phase level of the actual implementation. We have also demonstrated that this approach can deal with limited forms of pipelining as well. We believe this approach scales to more complex forms of pipelining, including superscalar program execution.

Even a sophisticated tool such as a symbolic trajectory evaluator is not in itself capable of formally verifying hardware systems. We also require a comprehensive methodology for specifying the intended behavior, for describing the relation between the specification and implementation, and for executing a set of tests that determine whether the specification is satisfied. We made significant progress in formulating such a methodology for the case of sequential processors. In our methodology, the desired behavior is specified as a series of *assertions* describing how each system operation modifies some component of the state. We also require the user to provide us with an *implementation mapping*, describing how the state components of the specification are realized in the implementation, both in terms of the physical storage elements and the detailed timing. The assertions, combined with the implementation mapping, are then used to generate a set of symbolic patterns to be verified by the trajectory evaluator. Given a successful evaluation, we can then guarantee that the specification is satisfied.

Some highlights of this work include:

- Development of a procedural library for symbolic Boolean manipulation based on BDDs— This code has been received widespread distribution and has become a standard against which all other packages are benchmarked.

- Development of a prototype symbolic trajectory evaluator based on extensions to the COSMOS switch-level simulator— This evaluator has been used at both Carnegie Mellon and at Intel to verify actual circuit designs.
- Development of a methodology for sequential processor verification— We have rigorously formulated techniques for specifying the desired system behavior and for checking, via symbolic trajectory evaluation, that a circuit adheres to its specification.
- Demonstration of verification capability on actual microprocessor— We verified a representative sample of instructions and addressing modes for an actual 16-bit microprocessor. This is one of the first successful verifications of an existing microprocessor. Most other efforts have used designs artificially created only for the verification.

Reactive System Verification

Our approach to verification of reactive systems is based on a technique called *temporal logic model checking*. In this approach specifications are expressed in a propositional temporal logic, and circuits are modeled as state-transition systems. An efficient search procedure is used to determine automatically if the specifications are satisfied by the transition system. The technique has several important advantages over mechanical theorem provers or proof checkers for sequential circuit verification. The most important is that the procedure is completely automatic. Typically, the user provides a high level representation of the model and the specification to be checked. The model checking algorithm will either terminate with the answer *true*, indicating that the model satisfies the specification, or give a counterexample execution that shows why the formula is not satisfied.

Prior to this project the main disadvantage of model checking was the *state explosion problem* that can occur if the device being verified has many components that can make transitions in parallel or if the circuit contains very wide data paths. In earlier implementations of the model checking algorithm, transition relations were represented explicitly by adjacency lists, and it was only possible to handle transition systems with a few thousand states.

During this project we developed model-checking algorithms that use a symbolic representation for state transition systems based on *binary decision diagrams* (BDDs). BDDs provide a canonical form for Boolean formulas that is often extremely compact, and very efficient algorithms have been developed for manipulating them. The implicit representation used by the new method is quite natural for modeling sequential circuits. Each state is encoded by an assignment of Boolean values to the set of state variables associated with the circuit. The transition relation can, therefore, be expressed as a Boolean formula in terms of two sets of variables: one set encoding the old state and the other encoding the new. The resulting formula is represented by a binary decision diagram. Using this technique, we have been able to handle examples that are many orders of magnitude larger than could be handled by the original algorithm. In fact, various refinements of this idea (like partitioning the transition relation) have made it possible to verify circuits with more than 10^{100} states. By combining model checking with various abstraction techniques, we have been able to handle even larger systems. In one example we were able to verify a pipelined ALU with more than 10^{1300} states (including 64 registers of 64 bits each).

To demonstrate the feasibility of symbolic model checking, we developed a tool, Symbolic Model Verifier (SMV), that checks specifications given in the temporal logic CTL and includes a built-in hardware description language for describing complex circuit designs. One of the most important features of SMV is its counterexample facility. If the given model does not satisfy one of its specifications, then SMV produces an execution trace illustrating why the specification is false. These counterexamples are extremely useful for debugging. Moreover, for models of moderate complexity, they are generally produced within a few minutes. This makes SMV extremely effective during the design process.

Although representing state machines symbolically vastly improves the capability of our verifier, many real-life examples are still intractable without additional refinements to the approach. In practice circuits often exhibit considerable symmetry. We have investigated techniques for reducing the complexity of temporal logic model checking in the presence of symmetry. In particular we have demonstrated that symmetry can frequently be used to reduce the size of the state space that must be explored during model checking. In the past symmetry has been used in computing the set of reachable states of a system when the transition relation is represented explicitly. However, earlier research did not consider arbitrary temporal properties or the complications that arise when BDDs are used in such procedures. We have formalized what it means for a finite state system to be symmetric and developed techniques for reducing such systems when the transition relation is given explicitly in terms of states or symbolically as a BDD. We have also identified an important class of temporal logic formulas that are preserved under this reduction. Finally, we have tested our ideas on a simple cache-coherence protocol based on the IEEE Futurebus+ standard.

To demonstrate that model checking techniques can help in the design of real industrial standards, we constructed a precise model of the the cache coherence protocol described in the draft IEEE Futurebus+ standard (IEEE Standard 896.1-1991) and used this procedure to show that the model satisfied a formal specification of cache coherence. Our model for the cache coherence protocol consists of 2300 lines of code (not counting comments) and is highly nondeterministic, both to reduce the complexity of verification (by hiding details) and to cover allowed design choices (indicated in the standard using the word *may*). The largest configuration that we verified had three bus segments, eight processors, and over 10^{30} states. In the process of formalizing and verifying the protocol, we discovered a number of errors and ambiguities. As far as we know, this is the first time that formal methods have been used to find nontrivial errors in a proposed IEEE standard. The final result of the project was a concise, comprehensible and unambiguous model of the cache coherence protocol that should be useful both to the Futurebus+ Working Group members, who are responsible for the protocol, and to actual designers of Futurebus+ boards. We believe that work on other standards would result in significantly faster development of correct designs if formal specification and verification techniques were used. Such a design strategy would no doubt result in faster and lower cost implementations by vendors as well.

In order to check realistic designs, it is necessary to also use abstraction and compositional verification techniques. Abstraction involves hiding details to make simple, high-level models of components. Compositional verification techniques involve using the decomposition of a design into a number of components running in parallel to simplify the verification. In previous research we developed the theoretical basis for doing compositional verification in the context of model checking. We have also shown how abstraction can be used to verify circuits with data paths. We expanded this work in two main ways. First, we have tried to apply compositional reasoning to check properties of the cache coherence protocol in the IEEE Futurebus+ standard. This effort grew out of our initial attempts to verify the protocol. In the initial work, we found that we could check various safety properties, but we were unable to directly verify liveness. Safety properties specify that ‘‘something bad never happens.’’ For example, we were able to check that two different caches that both hold a copy of some memory location must agree on the data in that location. Liveness properties state that ‘‘something good must happen.’’ For example, we may want to know that if a cache requests a copy of some memory location, then it eventually receives it. By using compositional reasoning techniques, we were able to show that the protocol did in fact satisfy a number of liveness conditions.

We have also developed methods for timing abstraction. Our earlier abstraction work focused on being able to collapse data paths. In timing abstraction, we compare an implementation against a specification that may have radically different timing. For example, we might want to compare a pipelined digital filter with a simple I/O description of the function that it should compute. Our approach is based on a very flexible form of mapping between different levels of abstraction. This mapping uses observer processes that watch the changes in state of the implementation and produce an abstract-level view of its behavior. An important property of our approach is that it works in conjunction with the compositional reasoning techniques. That is, we can abstract two parts of an implementation separately, verify the combined system at the abstract level, and then deduce the correctness of the combined implementation. This is a nontrivial property: We verify that the abstract parts communicate correctly, but each part of the implementation may have very different timing than its abstract representation. Our methodology guarantees that the parts of the implementation will communicate in an appropriate manner.

Some highlights of this research include:

- Development of the SMV symbolic model checker— This checker allows users to describe system models in a hardware description language and specified desired properties in temporal logic. The program then constructs a symbolic representation of the state machine and tests its adherence to the specification. If the specification is violated, SMV generates a "counter-example" trace of some failing case. This trace can be used as a guide in debugging the circuit.

- Verification of the cache coherency protocol in the IEEE Futurebus+ Standard— We found a number of errors in the protocol that had not been detected previously. This is the first time that automatic verification tools have been used to find errors in an IEEE standard.
- Collaboration with The Intel Design Technology Group to produce versions of our temporal logic model checking tools that are suitable for use in industry.
- Collaboration with the Intel Supercomputer Systems Division to use temporal logic model checking to verify cache coherency protocols in high performance processor designs.

2.2.3 Extensions and Applications of BDDs

Over the past few years, our Boolean manipulation methods based on Binary Decision Diagrams (BDDs) have seen widespread application in hardware verification, synthesis, and testing. We have continued to extend the basic representation and applications of BDDs.

Representation of Inductively-Defined Boolean Functions

We have developed a methodology for verification of inductively-defined hardware, which combines reasoning by induction and tautology-checking in a way that potentially incorporates the advantages of both.

The underlying basis of the proposed methodology is to identify classes of inductively-defined Boolean functions (IBFs) that are useful for representation of circuits and specifications, to associate a schema with each class (consisting of a canonical representation and symbolic manipulation algorithms), and to manipulate these representations appropriately for performing formal verification. So far we have focused on two classes of IBFs—linearly inductive functions (LIFs) and exponentially inductive functions (EIFs). We have provided representation schemata for each, based on extensions of BDDs. The important feature is that complexity of these representations and manipulations is independent of the parameter of inductive description. The class of LIFs is particularly useful since it naturally captures the temporal induction inherent in sequential system descriptions, and our LIF schema directly provides a canonical representation for sequential functions. We have also explored additional representation machinery required to represent structurally-inductive hardware in terms of LIFs and EIFs, including support for handling compositional descriptions.

The application of our schemata to formal verification lies in exploiting the power of induction that our representations are based upon. For example, we have shown that a proof by induction on a size parameter, for a specification property expressed as a Boolean combination of LIFs, corresponds to performing a tautology-check on the specification LIF formula. We can also exploit the canonicity property of our LIF representations to check input/output equivalence of two finite state machines, with different number of states and different state encodings. Other interesting properties we can verify include language equivalence/containment, the set of reachable states, etc. We are also exploring other applications, e.g. model checking. We have developed a prototype implementation of the IBF methodology. Initial results obtained with this implementation are very encouraging.

Computation and Application of Spectral Transforms

Spectral transforms such as the Walsh transform have numerous applications in CAD, particularly in the synthesis and testing of combinational circuits. Unfortunately, the usefulness of these techniques in practice has been limited by the size of the Boolean functions that can be handled by the transform. Since this transform is given as a vector with length of 2^n where n is the number of variables in the function, currently available techniques limit the functions to less than 20 variables. Recently, we were able to compute concise representations of the transform for functions with several hundred variables. Our technique is based on the use of BDDs to represent large, recursively-defined integer matrices such as the Walsh matrix, and to perform standard operations on them. The basis for the implementation of the matrix operations is an efficient algorithm for performing arithmetic on functions that map large Boolean vectors into the integers.

We applied our technique to Boolean technology mapping, an important problem in CAD. In Boolean technology mapping, it is necessary to check frequently whether two functions are equivalent under permutation and complementation of inputs and complementation of outputs. This equivalence checking can be quite time consuming. To make the process more efficient, it is important to have a good filter that can immediately reject unmatchable functions. The Walsh transformation can be used as the basis of such a filter. By computing the Walsh transformation using BDDs, we were able to perform technology mapping using cell libraries in which the individual cells have a large number of inputs. We modified the Ceres technology mapping system, developed at Stanford, to use the Walsh transformation as a filter. Our experimental results show that this filter is quite good. In fact it rejected all unmatchable functions that we encountered. Consequently, every function which passed the Walsh filter really matched a cell in the technology library.

2.3 Bibliography

[Blelloch 90] Blelloch, G.E.
Prefix Sums and Their Applications.
Technical Report CMU-CS-90-190, School of Computer Science, Carnegie Mellon University, November, 1990.

[Blelloch 92] Blelloch, G.E.
Portable parallel algorithms.
In *Proceedings 1992 DAGS Symposium*. June, 1992.

[Blelloch 93] Blelloch, G.E.
NESL: A Nested Data-Parallel Language (Version 2.6).
Technical Report CMU-CS-93-129, School of Computer Science, Carnegie Mellon University, April, 1993.

[Blelloch and Chatterjee 90]
Blelloch, G.E., and Chatterjee, S.
VCODE: A Data-Parallel Intermediate Language.
In *Proceedings Frontiers of Massively Parallel Computation*, pages 471-480. October, 1990.

[Blelloch and Hardwick 93]
Blelloch, G.E., Hardwick, J.C.
Programming parallel algorithms.
Technical Report CMU-CS-93-115, Computer Science Department, Carnegie Mellon University, February, 1993.

[Blelloch et al. 91] Blelloch, G.E., Leiserson, C.E., Maggs, B.M., Plaxton, C.G., Smith, S.J., Zagha, M.
A Comparison of Sorting Algorithms for the Connection Machine CM-2.
In *Proceedings Symposium on Parallel Algorithms and Architectures*, pages 3-16. Hilton Head, SC, July, 1991.

[Blelloch et al. 92] Blelloch, G.E., S. Chatterjee, and M. Zagha.
Solving linear recurrences with loop raking.
In *Proceedings Sixth International Parallel Processing Symposium*. March, 1992.

[Blelloch et al. 93a]
Blelloch, G.E., Chatterjee, S., Hardwick, J.C., Sipelstein, J., and Zagha, M.
Implementation of a Portable Nested Data-Parallel Language.
In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, May, 1993.

[Blelloch et al. 93b]
Blelloch, G.E., Chatterjee, S., Hardwick, J.C., Sipelstein, J., and Zagha, M.
CVL: A C Vector Library.
Technical Report CMU-CS-93-114, School of Computer Science, Carnegie Mellon University, February, 1993.

[Blelloch et al. 93c]
Blelloch, G.E., Heroux, M.A., and Zagha, M.
Segmented Operations for Sparse Matrix Computation on Vector Multiprocessors.
Technical Report CMU-CS-93-173, School of Computer Science, Carnegie Mellon University, August, 1993.

[Brookes 92a] Brookes, S.
Using fixed point theorems to prove retiming lemmas.
Journal on Formal Methods in System Design , 1992.

[Brookes 92b] Brookes, S.
An axiomatic treatment of partial correctness and deadlock in a shared variable parallel language.
Technical Report CMU-CS-92-154, School of Computer Science, Carnegie Mellon University, June, 1992.

[Brookes 93a] Brookes, S.
Using Fixed Point Semantics to Prove Retiming Lemmas.
Formal Methods in System Design 2:73-91, 1993.

[Brookes 93b] Brookes, S.
Full Abstraction for a Shared Variable Parallel Language.
In *Logic in Computer Science, Proceedings of the Eighth Annual IEEE Symposium*. IEEE, June, 1993.
Also appears as Technical Report CMU-CS-93-141.

[Brookes 93c] Brookes, S.
Fair Communicating Processes.
A Classical Mind: Essays in Honour of C. A. R. Hoare.
In Roscoe, A.W.,
Prentice-Hall International, 1993.

[Brookes and Geva 92]
Brookes, S., and S. Geva.
A Cartesian closed category of parallel algorithms Between Scott Domains.
Semantics of Programming Languages and Model Theory.
Gordon and Breach Science Publishers, 1992.
Also available as Technical Report CMU-CS-91-160.

[Bryant 92a] Bryant, R.E.
Symbolic Boolean manipulation with ordered binary decision diagrams.
In *ACM Computing Surveys*. ACM, June, 1992.
Also available as CMU-CS-92-160.

[Bryant 92b] Bryant, R.E.
Symbolic boolean manipulation with ordered binary decision diagrams.
ACM Computing Surveys 24(3):293-318, 1992.

[Bryant 93] Bryant, R.E.
Symbolic Analysis Methods for Masks, Circuits, and Systems.
In *International Conference on Computer Design*. October, 1993.

[Bryant et al. 93] Bryant, R.E., Tygar, J.D., and Huang, L.P.
Geometric characterization of series-parallel variable resistor networks.
In *Proceedings - IEEE International Symposium on Circuits and Systems*. IEEE Circuits and Systems Society, May, 1993.

[Burch and Long 92]
Burch, J.R., and D.E. Long.
Efficient boolean function matching.
In *IEEE International Conference on Computer-Aided Design*. IEEE, November, 1992.

[Burch et al. 91] Burch, J.R., E.M. Clarke, and D.E. Long.
Representing Circuits More Efficiently in Symbolic Model Checking.
In *Proceedings of the 28th ACM/IEEE Design Automation Conference*. ACM, June, 1991.

[Chatterjee et al. 90]
Chatterjee, S., Blelloch, G.E., and Zagha, M.
Scan Primitives for Vector Computers.
In *Proceedings Supercomputing~'90*, pages 666-675. November, 1990.

[Chatterjee et al. 91]
Chatterjee, S., Blelloch, G.E., and Fisher, A.L.
Size and Access Inference for Data-Parallel Programs.
In *ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 130-144. June, 1991.

[Clarke et al. 93a] Clarke, E., Grumberg, O., Hiraishi, H., Jha, S., Long, D.E., McMillan, K., and Ness, L.
Verification of the Futurebus+ cache coherence protocol.
In *Proceedings of 1993 IFIP Conference on Hardware Description Languages and Their Applications*. IFIP, April, 1993.

[Clarke et al. 93b] Clarke, E., McMillan, K., Zhao, X., Fujita, M., and Yang, J.
Spectral transforms for large Boolean functions with applications to technology mapping.
In *Proceedings of the ACM/IEEE Design Automation Conference*. ACM/IEEE, 1993.

[Clarke et al. 93c] Clarke, E.M., Grumberg, O., Hiraishi, H., Jha, S., Long, D.E., McMillan, K.L., and Ness, L.A.
Verification of the Futurebus+ Cache Coherence Protocol.
In *CHDL '93: The IFIP Conference on Hardware Description Languages and their Applications*. IFIP, April, 1993.

[Gupta and Fisher 93a]
Gupta, A., and Fisher, A.L.
Parametric circuit representation using inductive Boolean functions.
In *Computer-Aided Verification: Proceedings*. Springer-Verlag, June, 1993.

[Gupta and Fisher 93b]
Gupta, A., and Fisher, A.L.
Representation and Symbolic Manipulation of Linearly Inductive Boolean Function.
In *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*. November, 1993.

[Kung et al. 91] Kung, H.T., P. Steenkiste, M. Gubisto, and M. Khaira.
Parallelizing a New Class of Large Applications over High-Speed Networks.
In *Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, April, 1991.

[Long 93] Long, D.E.
Model Checking, Abstraction and Compositional Verification.
PhD thesis, Computer Science Department, Carnegie Mellon University, July, 1993.
Appears as Technical Report CMU-CS-93-178.

[McMillan and Dill 92]
McMillan, K.L., and D.L. Dill.
Algorithms for interface timing verification.
In *Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*. Princeton University, March, 1992.

[Schwabe et al. 92]
Schwabe, E., G.E. Blelloch, A. Feldmann, O. Ghattas, J. Gilbert, G. Miller, D. O'Hallaron, J. Shewchuk, and S.-H Teng.
A separator-based framework for automated partitioning and mapping of parallel algorithms in scientific computing.
In *Proceedings 1992 DAGS Symposium*. June, 1992.

[Sheffler 92] Sheffler, T.J.
Match and move, an approach to data parallel computing.
PhD thesis, CarnegieMellon, October, 1992.
Available as technical report CMU-CS-92-203.

[Sheffler 93a] Sheffler, T.J.
Implementing the multiprefix operation on parallel and vector computers.
In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*. ACM, 1993.
Also appears as technical report CMU-CS-92-173.

[Sheffler 93b] Sheffler, T.J.
Match and move: parallel constructs for sparse matrix and graph algorithms.
In *DIMACS Workshop on Parallel Algorithms for Unstructured and Dynamic Problems.*. DIMACS, 1993.

[Sheffler 93c] Sheffler, T.J.
Implementing the Multiprefix Operation on Parallel and Vector Computers.
In *1993 Symposium on Parallel Algorithms and Architectures*. July, 1993.

[Sheffler 93d] Sheffler, T.J.
Writing Portable Parallel Programs with Match and Move.
In *Leeds Second Workshop on Abstract Machine Models for Highly Parallel Computers*. April, 1993.

[Sheffler and Bryant 92a]
Sheffler, T.J., and R.E. Bryant.
Match and move: An approach to data parallel computing.
In *MIT/Brown Conference on Advanced Research in VLSI*. April, 1992.

[Sheffler and Bryant 92b]
Sheffler, T.J., and Bryant, R.E.
Work efficient hashing on parallel and vector computers..
Technical Report CMU-CS-92-172, Computer Science Department, Carnegie Mellon University, August, 1992.

[Sheffler and Bryant 93a]
Sheffler, T.J., and Bryant, R.E.
An analysis of hashing on parallel and vector computers.
In *International Conference on Parallel Processing*. 1993.

[Sheffler and Bryant 93b]
Sheffler, T.J., and Bryant, R.E.
An Analysis of Hashing on Parallel and Vector Computers.
In *Proceedings 1993 International Conference on Parallel Processing*. August, 1993.

[Sipelstein and Blelloch 91]
Sipelstein, J., and Blelloch, G.E.
Collection-Oriented Languages.
Proceedings of the IEEE 79(4):504-523, April, 1991.

[Zagha and Blelloch 91]
Zagha, M., and G.E. Blelloch.
Radix sort for vector multiprocessors.
In *Supercomputing '91: Proceedings*. International Supercomputing Institute, November, 1991.

3. RESEARCH IN OBJECT MANAGEMENT

Our research in Object Management in the last three years has been conducted along four lines:

- The Venari project addressed the problem of managing *persistent* object repositories. It focused on designing and implementing, in Standard ML, a search environment for program module libraries; hence the objects of interest were software components. It also focused on runtime extensions for Standard ML that could support this search application.
- The Coda project addressed the problem of how to manage efficiently very large *distributed* object bases. It focused primarily on disconnected operation and automatic server resolution of distributed and replicated file systems.
- The Dyad project addressed the problem of managing the *security* of object bases. It focused primarily on the applications of secure co-processors and formal models of secure systems.
- Other work addressed the problem of making semantic analyzers, such as model checkers and theorem provers, more *efficient* and practical. These analyzers can in principle be used to aid in the match process of an object repository search.

Systems like Tabs and Camelot demonstrate the viability of layering a general-purpose transactional facility upon an operating system. Languages such as Argus and Avalon/**C**$^{++}$ go one step further by providing linguistic support for transactions in the context of a general-purpose programming language. In principle programmers can now use transactions as a unit of encapsulation to structure an application program without regard for how they are implemented at the operating system level.

In practice, however, transactions have not yet been shown useful in general-purpose applications programming. One problem is that state-of-the-art transactional facilities are so tightly integrated that application builders must buy into a facility in toto, even if they need only one of its services. For example, the Coda file system 3.2 was originally built on top of Camelot, which supports distributed, concurrent, nested transactions. Coda needs transactions for storing ‘‘metadata’’ (e.g., inodes) about files and directories, and is structured such that updates to metadata are guaranteed to occur with only one thread executing at a single-site, within a single, top-level transaction. Hence Coda needs only single-site, single-threaded, nonnested transactions, but by using Camelot, was forced to pay the performance overhead for Camelot’s other features.

A second problem with existing transactional facilities is that they have been designed primarily with applications such as electronic banking, airline reservations, and relational databases in mind. Non-traditional applications such as proof support environments, software development environments, and CAD/CAM systems want transactional features, most notably data persistence, but have different performance characteristics. For example, these applications do not manipulate simple database records but rather complex data structures such as proof trees, abstract syntax trees, symbol tables, car engine designs, or VLSI designs. Also, users interact with these data during long-lived ‘‘sessions’’ rather than short-lived transactions; indeed we can view a ‘‘session’’ itself as a sequence of transactions. For example, during a proof session a user might explore one path in a proof tree transactionally; if the path begins looking like a dead-end the user may choose to abort, backing all the way up to the first node in the path or perhaps to some intermediate node along the way. Also, though multiple users may need to share these data, simultaneous access might be less frequent. For example, proof developers might work on independent parts of a proof tree, perhaps each proving auxiliary lemmas of the main theorem; software developers might modify different modules of a large program. Finally, these nontraditional applications typically support different update patterns. Whereas travel agents make frequent updates to airline reservations databases, we do not expect to make updates as frequently to proofs of theorems saved in proof libraries.

3.1 Venari

The Venari Project application domain is software development environments. One specific problem we are addressing is searching large libraries, e.g., specification and program libraries, used in developing software systems. We imagine a scenario in which a user searches a large library for a program module that ‘‘satisfies’’ a particular specification. We might wish to perform each query as a transaction, for example, to guarantee isolation from any concurrent update transaction or to abort the query after the first *n* modules are returned.

Another problem in software development is version management. Many people working on a large software project need to coordinate updates to different components of the software under development. Systems like RCS provide some configuration management help. The Venari Project has been investigating support for separate compilation for Standard ML as a first step toward solving the version management problem for Standard ML.

The Venari Project is revisiting support for transactions by adopting a ‘‘pick-and-choose’’ approach rather than a ‘‘kit-and-kaboodle’’ approach. Ideally, we want to provide separable components to support transactional semantics for different settings, e.g., in the absence or presence of concurrency and/or distribution. Programmers are then free to compose those components supporting only those features of transactions they need for their application. Our approach also enables programmers to code some applications that cannot be written without an explicit separation of concerns.

We want to support this approach at the programming language level. The current status of the Venari Project is that we can support concurrent, multithreaded, nested transactions in the context of Standard ML. We have not yet addressed distribution and see that as our next big step. Our primary concern was to break apart separable transactional concepts before tackling distribution.

The first of the following three subsections address our results with respect to the runtime support for SML for persistence and concurrency. The subsequent four subsections address our results with respect to the software development application domain.

3.1.1 Extensions to SML for Persistence and Transactions

We designed and implemented extensions to Standard ML to support concurrent, multithreaded, nested transactions. This design separates the orthogonal concerns of persistence and undoability, and the extensions are implemented using Recoverable Virtual Memory (RVM) [Nettles and Wing 92a].

The implementation for the single-threaded version includes approximately 200 lines of new SML code and modifications to about 80 lines of existing SML code, and 850 lines of new **C** code and modifications to 250 lines of existing **C** code. It runs on top of Mach 2.5. Preliminary benchmarks indicate that we can perform one or two user-level transactions per second, which is acceptable for our application domain (search through program module libraries). Most of the cost in persistence is time spent on scanning the persistent heap; most of the cost in undoability is garbage collection.

Our progress on Venari's extensions to the runtime of SML/NJ include:

- We provide the following key interfaces: persistence, undoability, threads, skeins, and locks in terms of SML modules. These can be mixed and matched to support different models of computation. For example, multithreaded persistence, multithreaded undo, concurrent transactions, and concurrent multithreaded transactions are all possible. All nested forms of each model is possible as well [Wing et al. 92a].
- We devised and provided a new control abstraction, *skein*, which is a self-contained bundle of threads. Skeins enable us to implement easily models of computation that previous transactional systems do not support.
- We implemented a nontrivial application using the Venari extensions for persistence for SML: a simple distributed file system. We reported on this work at the San Francisco ML workshop [Dean 92].
- We implemented a nontrivial application using all the Venari/ML interfaces, showing the utility of concurrent multithreaded transactions: a library of `.bib` files over which database-like queries can be performed.
- We merged the support for persistence and undoability in the runtime using a general notion of *undo lists*.
- We extended SML with a standard threads package. This work is reported in more detail in 3.1.2.
- We worked on improving garbage-collection response times by adding a concurrent, incremental garbage collector with excellent performance [Nettles et al. 92a]. This work is reported in more detail in 3.1.3.

We incorporated work on improved virtual memory performance [Cooper et al. 92a] into the runtime.

We also helped the Fox Project work on porting SML/NJ to Mach. We designed and wrote a code generator for Motorola 88100, and ported SML/NJ Multiprocessor Interface [Morrisett and Tolmach 92] to Omron Luna 88k. We designed and implemented a Mach 3.0 kernel/IPC interface for SML/NJ, and an SML/Mig 3.0 stub generator for SML/NJ and Mach 3.0.

3.1.2 A Portable Multiprocessor Interface for SML/NJ

We have built a portable platform for running Standard ML of New Jersey programs on multiprocessors. It can be used together with first-class continuations to implement user-level thread packages for multiprocessors within the ML language. The platform supports experimentation with different thread scheduling policies and synchronization constructs. It has been used to construct a Modula-3 style thread package and a version of Concurrent ML, and has been ported to three different multiprocessors, including SGI 4D/380S, Omron Luna88K, and Sequent Symmetry. [Morrisett and Tolmach 93a] describes the platform's design, implementation, and performance.

We continued investigating the impact of including *first-class stores* in a language. A first-class store is an object that represents the state of all mutable values at a particular point in a program's execution. [Morrisett 93] shows that by extending first-class stores with the ability to partition the store, a variety of new applications present themselves. The paper details three example applications: ‘‘functional’’ arrays, a nested transaction system, and a replay-debugger. The paper then gives a denotational semantics for first-class stores when added to an ML-like language. We show how first-class stores can be implemented portably within a language such as SML. We then show how integration with the language implementation (i.e. garbage collector) can provide a more efficient realization of the primitives.

3.1.3 Replication-Based Copying Garbage Collection

[Nettles et al. 92b] introduces a new *replication-based* copying garbage collection technique. We have implemented one simple variation of this method to provide incremental garbage collection on stock hardware with no special operating system or virtual memory support. The performance of the prototype implementation is excellent: Major garbage collection pauses are completely eliminated with only a slight increase in minor collection pause times.

Unlike the standard copying algorithm, the replication-based method does not destroy the original replica when a copy is created. Instead, multiple copies may exist, and various standard strategies for maintaining consistency may be applied. In our implementation for Standard ML of New Jersey, the mutator continues to use the from-space replicas until the collector has achieved a consistent replica of all live data in to-space.

We have designed a concurrent garbage collector using the replication-based technique. We expect replication-based GC methods to be useful in providing services for persistence and distribution.

In [Nettles and O'Toole 93] we describe how the replication-based copying garbage collector permits continuous unimpeded mutator access to the original objects during copying. The garbage collector incrementally replicates all accessible objects and uses a mutation log to bring the replicas up-to-date with changes made by the mutator. An experimental implementation demonstrates that the costs of using our algorithm are small and that bounded pause times of less than 50 milliseconds can be readily achieved.

[Nettles 92] provides an informal but abstract description of copying collection, a formal specification of copying collection written in the Larch Shared Language and the Larch/**C** Interface Language, a simple implementation of a copying collector written in **C**, an informal proof that the implementation satisfies the specification, and a discussion of how the specification applies to other types of copying GC such as generational copying collectors. Limited familiarity with copying GC or Larch is needed to read the specification.

Other work in garbage collection yielded [Cooper et al. 92b]. We improved the performance of garbage collection in the Standard ML of New Jersey system by using the Mach kernel's virtual memory facilities. We took advantage of Mach's support for *large sparse address spaces* and *user-defined paging servers* and decreased elapsed time for realistic applications by as much as a factor of four.

We have implemented the first copying garbage collector that permits continuous, unimpeded, mutator access to the original objects during copying [O'Toole and Nettles 93]. No low-level synchronization between the client and the garbage collector is required on individual object operations. The garbage collector replicates live heap objects and periodically synchronizes with the client to obtain the client's current root set and mutation log. An experimental implementation using the Standard ML of New Jersey system on a shared-memory multiprocessor demonstrates excellent pause time performance and moderate execution time speedups.

We developed the first concurrent compacting garbage collector for a persistent heap [Nettles et al. 93]. Client threads read and write the heap in primary memory and can independently commit or abort their write operations. When write operations are committed, they are preserved in stable storage and can thus survive system failures. Clients can freely access the heap during a garbage collection because a replica of the heap is created by the *stable replica collector*. A log is maintained to capture client write operations. This log is used to support both the transaction system and the replication-based garbage collection algorithm.

Experimental data from our implementation were obtained from a transactional version of the SML/NJ compiler and modified versions of the TPC-B and OO1 database benchmarks. The pause time latency results show that the prototype implementation provides significantly better latencies than stop-and-copy collection. For small transactions, throughput is limited by the logging bandwidth of the underlying log manager. The results provide strong evidence that the replication copying algorithm imposes less overhead on transaction commit operations than other algorithms.

3.1.4 Types as Search Keys for SML Program Libraries

The Venari Project is developing support for browsing and searching SML program libraries in an environment of X Windows and Gnu Emacs. Searching and browsing are implemented through the use of index files (generated by the SML/NJ compiler) that map identifiers to their point of definition and their type.

Currently browsing is done through the Gnu Emacs Tags package. While editing an SML program module, one can click the mouse on an externally defined identifier, and the defining source module is displayed in another window. If the user gives a text pattern, a menu of matching identifiers is generated. The user can click on an entry of this menu to visit the defining source module.

Our system supports searching libraries using types as search keys. The user gives a type query as a string (e.g., `int->int`), and the system parses that string into an Abstract Syntax Tree (AST). For each library object, the query AST is unified with the type AST for the library object. If the attempt succeeds, information about the library object is presented to the user.

Our first implementation of the type searching system was slow. For each query all index files in the library were scanned, and the type strings for each library object were parsed into an AST before unification with the query AST. The second implementation exploited a persistent object base to store the type ASTs of the library objects.

We compared the performance of these two approaches on a test library of 2460 objects. The test query was `string->string` and 100 library objects matched it. The first version took 57 seconds of real time to process the test query; the second took less than one second. This result shows the effectiveness of using a persistent object base to store indexes for libraries of semantically rich objects.

3.1.5 Signature Matching

Software reuse is only effective if it is easier to locate (and appropriately modify) a reusable component than to write it from scratch. We present ‘‘signature matching’’ as a method for achieving this goal by using signature information easily derived from the component [Zaremski and Wing 93]. We consider two kinds of software components, functions and modules, and hence two kinds of matching, function matching and module matching. The signature of a function is simply its type; the signature of a module is a multiset of user-defined types and a multiset of function signatures. For both functions and modules, we consider not just exact match, but also various flavors of relaxed match. We briefly describe an experimental facility written in Standard ML for performing signature matching over a library of ML functions.

We finished implementing the components necessary to do function-signature matching for SML. These provide the ability to do basic function-signature matching (with variable renaming) and five different kinds of relaxed function-signature matching: generalized matching (function is more general than query), specialized matching (function is less general than query), unified matching (function and query types can be unified), reordered tuple matching (tuples in function are permutations of tuples in query), and curried/uncurried matching (function and query are curried and uncurried forms of the same function).

3.1.6 A Specification Language for SML

Signature matching is not as accurate as specification matching. In order to do specification matching, we needed to design a specification language for a subset of ML. We chose to build upon the Larch approach.

The Venari Project worked out a preliminary design of Larch/ML, a Larch interface language for SML. The Larch family of specification languages features the Larch Shared Language (LSL) and interface languages tailored to specific programming languages. We have shown how the Larch specification approach can be applied to a nontrivial example, that of specifying a visual language editor [Wing and Zaremski 91a] and how it can be integrated with more common software design methods like Structured Analysis and Structure Charts [Wing and Zaremski 91b].

Larch/ML enables one to attach specification clauses to objects in SML signatures. A specification clause for a function consists of a precondition, a postcondition, and a declaration of side effects. In Larch/ML one can write specifications for polymorphic, side effecting, and higher-order functions. Function parameters can be polymorphic and side effecting also. A specification can refer to the specification of a function parameter.

We are starting to design and implement tools for Larch/ML, and have written a lexical specification, a parsing grammar, and abstract syntax. From these descriptions, a syntax checker is generated by sml-yacc and sml-lex.

Larch/ML clauses are embedded within SML signatures inside special comments. An SML compiler can compile Larch/ML sources as SML signatures, ignoring the specifications.

We use Larch/ML specifications as search keys. In our design, the Larch/ML translator generates fully-checked ASTs from Larch/ML specifications. The specification ASTs for an SML program library can be maintained in a library catalog implemented as a persistent object base. Specification queries can be translated into checked ASTs and then matched against specification ASTs in the library catalog.

We implemented name scope checking for the Larch/ML checker. In our design of Larch/ML we chose to disallow declaring names in nested scopes so as to shadow existing names. During implementation we realized that, if a specification imports a Larch trait, many names exported by that trait would be introduced into the importing specification. This situation could lead to many name conflicts between imported traits and, because of our nonshadowing principle, many names that could not be used in the specification. In a slight departure from other Larch interface languages, we require that any imported name be qualified by the name of the trait that defines the name.

We began work on Larch/ML type checking. Each ML type is mapped to a Larch sort, either through a ‘‘based on’’ clause in the specification or by convention for built-in ML types. Type checking a Larch/ML specification consists of ensuring that the sort of each requires-and-ensures clause is Boolean and that each subexpression of those clauses is well-sorted [Wing et al. 92b].

3.1.7 Separate Compilation for SML

The Venari Project re-engineered the separate compilation facility in the Standard ML of New Jersey (SML/NJ) compiler. We introduced a new abstract interface for separate compilation primitives, implemented these primitives in the compiler, and developed a separate compilation facility, SourceGroup, as a library that uses the compilation primitives. SML/NJ includes this new separate compilation facility in its standard release from Princeton University and AT&T Bell Labs.

Our new compilation primitives generate target binaries that are over 60% smaller than the those generated by the old system. The new separate compilation facility is also faster than the old one. In an experiment involving a system of 114 source files, a change to one particular file triggers the recompilation of 30 source files. The old system took 29 minutes of real time to recompile the system, whereas the new system took just 4 minutes. Other source changes to the system showed similar results.

SourceGroup is an open-architecture, selective recompilation system designed for interactive compilers and interpreters. Separation of concerns was a significant principle guiding the design of SourceGroup. Although some other solutions to this problem have large user communities and have industrial-strength implementations, they are large monolithic systems that may include a module system, compilation methods, source dependency descriptions, version control, and configuration management. Some of these systems are embedded in the operating system or entangled in a programming environment and wedded to a particular editor. Various pieces of knowledge are hardwired into them. SourceGroup, by contrast, is organized as a library with publicly available abstract interfaces. It can be integrated with separate version control and configuration management systems and is easily adapted to the compilation methods and module systems of many languages and tools.

Although the popular tool ‘‘make’’ focuses only on separate compilation, it has three fundamental problems: (1) It forces one to describe file dependencies directly. This global system information can change if a new system configuration is desired. (2) It mixes dependency and source processing descriptions. Makefiles must be rewritten if one wants to do source processing in a different way. (3) It assumes a batch style of source processing. It does not support interactive compilers and interpreters.

SourceGroup allows one to describe what objects or modules each source file imports and exports. From this local information, it determines the dependencies among system files. The description of this information can be separated from compilation methods. This enables developing several applications (e.g., selective recompilation, program analysis, target file deletion, source file renaming) using the same module descriptions. While other systems assume a batch compiling style, SourceGroup also supports interactive compilers and interpreters that maintain an environment (symbol-table) in which a software system is incrementally built from its parts.

The SourceGroup separate compilation system, which was done by the Venari Project in 1991, prompted the implementors of the SML/NJ compiler to make substantial changes in the underlying compiling primitives. The Venari Project upgraded SourceGroup to exploit these changes to the compiler and to make the system much easier for new users to learn. Researchers at Cornell rely heavily on SourceGroup in their implementation of Distributed ML (DML).

Based on experience with SourceGroup, we have completed a paper design that reformulates a separate compilation facility for SML. The new formulation fixes some problems with SourceGroup and exploits the new implementation in SML/NJ of first-class environments. Environments, which are semantic entities in the definition of Standard ML, are sets of bindings from module names to actual modules (signatures, functors, and structures). The SML/NJ compiler now features a type environment and associated operations.

We also began work on applying new parsing techniques to ML, with a view to providing a more sophisticated development environment to Venari users [Haines 92].

3.1.8 Other Developments

Over the span of this contract, we achieved significant results in other areas tangentially related to the Venari Project.

Developments in Larch

[Martin and Wing 93] contains papers on the Larch languages and uses of the Larch Prover. It presents state-of-the-art research results on the Larch language design and the application of LP to examples ranging from hardware verification and civil engineering. Authors attended the First International Workshop on Larch in Dedham, MA in July 1993, which the editors organized.

[Guttag and Horning 93] is the Larch Book and contains a description of the Larch specification languages, including the Larch Shared Language, a Larch interface language for **C**, and a Larch interface language for Modula-3. It contains a description of Larch's most powerful tool, the Larch Prover (LP), which can be used to do semantic analysis of Larch specifications. Finally, it contains a handbook of LSL traits.

[Wing et al. 92c] presents in more detail our preliminary design for a Larch interface language for ML, discussed earlier in section .[1] ML's support for higher-order functions

[1] Since this design covers only a subset of Standard ML, we will refer to our interface language as Larch/ML rather than Larch/SML.

suggests a need to go beyond the first-order logical foundations of Larch languages. We also propose a new application, *specification matching*, for the Larch Prover — a benefit of extending LP to handle full first-order logic. This paper describes on-going work and suggests a number of open problems related to Larch/ML and to LP as used for specification matching. We assume rudimentary knowledge of Larch, its languages, and the ‘‘two-tiered’’ approach.

Formal specifications can be used in software development to make programming easier and programs better. Larch supports a two-tiered approach to the formal specification of program interfaces. One tier is written in the Larch Shared Language (LSL). An LSL specification describes mathematical abstractions such as sets, relations, and algebras; its semantics are defined in terms of first-order theories. The second tier is written in a Larch interface language, one designed for a specific programming language. An interface specification describes the effects of individual modules, e.g., state changes, resource allocation, and exceptions; its semantics is defined in terms of first-order predicates over two states, where state is defined in terms of the programming language's notion of state. Thus, LSL is programming-language-independent while a Larch interface language is programming-language-dependent.

The Larch toolkit includes syntax checkers for LSL and various Larch interface languages, including Larch/**C** and Larch/Modula-3. Most significantly, the Larch toolkit includes a semantic analyzer in the form of the Larch Prover. LP is based on a fragment of first-order logic, and its design derives from a long series of theorem provers based on rewrite-rule theory.

We discuss our Larch work in [Guttag et al. 93], a self-contained presentation of the Larch methodology and of specific Larch languages for specifying **C** and Modula-3 interfaces. Written for practicing programmers and students, it covers both tutorial and advanced material, and includes numerous examples, among them an extensive handbook of Larch Shared Language specifications. It also provides information on publicly available tools (including LP) that support the methodology and the languages. [Wing 93a] provides further information on our research in Larch.

Subtyping

Hierarchy is an important component of object-oriented design. Hierarchy allows the use of type families, in which higher level supertypes capture the behavior that all of their subtypes have in common. For this methodology to be effective, it is necessary to have a clear understanding of how subtypes and supertypes are related. [Liskov and Wing 92] takes the position that the relationship between subtypes and supertypes should ensure that any property proved about supertype objects also holds for its subtype objects. The paper presents two ways of defining the subtype relation, each of which meets this criterion, and each of which is easy for programmers to use. It also discusses the ramifications of this notion for the design of type families and for the contents of type specifications and presents a notation for specifying types formally.

3.2 Coda

The Coda Project achieved a substantial improvement in robustness of Coda clients and servers. We can now deploy Coda on portable computers to a small user community. We have implemented, debugged, and stress-tested an incremental backup system for Coda, and weaned Coda off Camelot by using RVM on production as well as test servers.

The Coda group spent substantial effort stress testing, bulletproofing, and performance optimizing RVM. We are exploring design alternatives for weakly connected operation of Coda (i.e., operation over low-bandwidth networks such as cellular modem links). We are also implementing SynRGen, a synthetic file reference generator for stressing UNIX file systems.

Coda provides resilience to server and network failures through the use of two distinct but complementary mechanisms. One mechanism, *server replication*, stores copies of a file at multiple servers. The other mechanism, *disconnected operation*, is a mode of execution in which a caching site temporarily assumes the role of a replication site. Disconnected operation is particularly useful for supporting portable clients.

The Coda Project achieved the following milestones:

- We ported Coda from **C++** 1.2 to **C++** 2.0.
- We substantially improved the robustness of RVM.
- We restructured and cleaned up both Coda and RVM for external distribution.
    - We distributed Coda and RVM to to IBM Yorktown Heights and Digital Equipment Corp., Littleton.
    - We enlarged Coda's user community for Coda, including an expanded Coda base at Carnegie Mellon.
- We re-implemented Coda's resolution subsystem using persistent logs in RVM.
- We performed detailed experiments to evaluate performance of directory resolution with logs in RVM.
- Our incremental backup subsystem is in production use on Coda.
- We began collecting Coda usage data.
- We implemented a network emulator to allow experimentation with weakly connected operation.
- We enhanced the repair tool to handle cross-directory renames and to simplify use.
- The Coda backup system was substantially bullet-proofed and enhanced to simplify operator procedures.
- We demonstrated MS-DOS access to Coda functionality on Mach 3.0.
- RVM has been adopted as a base for the Thor project at MIT.

3.2.1 Alleviating Limitations of Disconnected Operation

Disconnected operation in Coda has proven to be effective for using distributed file systems from portable computers. However, disconnected operation has several limitations:

- Cache misses are not transparent. A user may be able to work in spite of some cache misses, but certain critical misses may frustrate his efforts.
- Long disconnects increase the likelihood of resource exhaustion on the client due to the growing change log and new data in the cache.
- Long disconnects also increase the probability of conflicts requiring manual intervention upon reconnection.

Wireless technologies, such as cellular phone, and even traditional dialup lines, present an opportunity to alleviate some of the shortcomings of disconnected operation. The characteristics of these networks differ substantially from those of local area networks (LANs), on which many distributed file systems are based. These ''weak connections'' are slower than LANs, and some of the wireless technologies have the additional properties of intermittence and nontrivial cost.

We have enhanced the RPC2 and SFTP protocols to support weakly connected operation in Coda. The protocol parameters now dynamically adapt to changing network conditions. Reliable communication is now possible from 10Mb/s down to 1200b/s.

3.2.2 Application Specific File Resolution

Our work on the Coda File System has shown that optimistic replication is a key enabling technology for high availability. Although this strategy provides higher data availability, it cannot guarantee data consistency across partitions. It is imperative to detect and resolve concurrent updates in multiple partitions.

Our previous accomplishments in Coda resulted in the ability to perform transparent resolution of directory updates. We have now been working on an extension of the Coda resolution subsystem to support transparent resolution on arbitrary files. Since the operating system does not possess any semantic knowledge of file contents, it is necessary to obtain assistance from applications. The key to this assistance comes from an application-independent invocation mechanism allowing pre-installed, application-specific resolvers to be transparently executed when a conflict arises.

We have completed the preliminary design of an interface for application-specific file resolution. This interface will allow user-installed code to be transparently invoked by Venus to perform file resolution in the course of normal file accesses.

3.2.3 Concurrency Control for Optimistic Replication

With the increasing frequency and scale of data-sharing activities made possible by distributed UNIX file systems such as AFS, Coda, and NFS, there is a growing need for effective consistency support for concurrent file accesses. The problem is especially acute in the case of mobile computing, because periods of disconnected or weakly connected operation may lead to read/write inconsistencies in shared data.

Previous efforts to address this problem in UNIX have proved impractical because they have often been direct transplants of traditional database transactions into the UNIX file system world. The result has been poor performance because of the significant differences in user environment, transaction duration, and object size between UNIX file systems and database systems.

We have completed the preliminary design of an UNIX-compatible transactional facility, LWT, that will allow file accesses to be grouped by transactional boundaries for purposes of read/write and write/write conflict detection and resolution.

3.2.4 Evaluation and Improvement of Hoarding

Hoarding, fetching data in anticipation of a distant cache miss, is essential to the success of disconnected and weakly connected operation. Although Coda supports this technique, it only has a limited set of tools to help users hoard effectively at present.

Cache analysis work to date has relied on the assumption that all cache misses are basically equal. In other words, such analysis assumes that all cache misses occur at roughly the same cost to the user, and that the cache miss will be fulfilled in a reasonable time frame. This assumption is not valid during disconnected operation, and it may also not be valid during weakly connected operation. The cache miss ratio further fails to account for differences in the timing of cache misses, or in differences in file length.

We have completed a preliminary identification of metrics to evaluate quality of hoard information and designed a series of experiments that will help quantify users' ability to assist in long-term cache management.

3.2.5 RVM Optimizations

We completed implementations of intra- and intertransaction optimizations in RVM. Intratransaction optimizations are valuable because they detect and coalesce overlapping ranges of RVM modifications. Such overlaps turn out to be frequent occurrences in Venus and on the servers. Our measurements indicate that intratransaction optimizations result in nearly 25% reduction in log traffic on the Coda servers.

The intertransaction optimizations are valuable when no-flush transactions are used. Because of reference locality, the same set of memory locations are often involved in no-flush transactions. Intertransaction optimizations substantially reduce the size of data logged and thereby contribute to performance. Measurements of intertransaction optimizations are in progress.

3.2.6 Coda Usage Analysis

We are involved in a preliminary collection and analysis of Coda usage data. The data on a variety of usage-related parameters are being transparently collected and stored in an SQL database. Postprocessing scripts to query the database have been written. A preliminary analysis of these data suggests a number of improvements that we are currently implementing.

''Recoverable virtual memory'' refers to regions of a virtual address space on which transactional guarantees are offered. [Satyanarayanan et al. 93a] describes RVM, an efficient, portable, and easily-used implementation of recoverable virtual memory for UNIX environments. A unique characteristic of RVM is that it allows independent control over the transactional properties of atomicity, permanence, and serializability. This characteristic leads to considerable flexibility in the use of RVM, potentially enlarging the range of applications than can benefit from transactions. It also simplifies the layering of functionality such as nesting and distribution atop RVM. The paper shows that RVM performs well throughout its intended range of usage, even though it does not benefit from specialized operating system support. We also demonstrate the importance of intra- and intertransaction optimizations in supporting recovering virtual memory.

In [Satyanarayanan et al. 93b] we present qualitative and quantitative data on file access in a mobile computing environment. This information is based on actual usage experience with the Coda File System over a period of about two years. Our experience confirms the viability and effectiveness of disconnected operation. It also exposes certain deficiencies of the current implementation of Coda and identifies new functionality that would enhance its usefulness for mobile computing. The paper concludes with a description of what we are doing to address these issues.

In [Kumar and Satyanarayanan 93a] we describe an interface that facilitates incorporating application-specific knowledge in an optimistically replicated file system. Conflicts arise in such systems because object replicas can be modified simultaneously in different network partitions. Application-specific knowledge is made available by the application writer in specialized tools called Application-Specific Resolvers (or ASRs). The interface we describe here is used to bind ASRs to objects and to specify the conditions under which a specific ASR is invoked. This implementation allows the system to make conflict resolution transparent to the user, thus improving the usability of optimistic replication.

[Satyanarayanan 93] asserts that mobility will influence the design of distributed systems in fundamental ways. Extrapolating from the results of the Coda File System, the paper identifies resource asymmetry, security, and network adaptivity as the key long-term problems facing mobile computing systems. The content of this paper is thus a bridge between results and insights from work done under the current Object Management ARPA contract and the research proposed in the ''Secure Mobile Computing'' ARPA contract.

3.3 Verification

3.3.1 Model and Proof Checking

Verifying speed-dependent asynchronous circuits requires accurately modeling the propagation delay of gates. This is most often done using the inertial delay model. In [Burch 92a] we show that the inertial delay model can lead to false-positive verification results. We also describe an alternative model, called chaos delay, that avoids this problem.

In [Burch et al. 92] we describe an efficient algorithm for doing symbolic model checking on the propositional Mu-Calculus. This algorithm is used in verification methods for a variety of styles of modeling and specifying concurrent systems.

Recently, the size of the circuits that can be verified using this technique has increased by many orders of magnitude. In [Clarke et al. 92] we describe some of the algorithms that have made this increase possible. These algorithms are based on the use of *binary decision diagrams* to represent transition systems and state sets.

This powerful approach to verification uses state exploration methods to check the state space of a finite state system for hazards, deadlocks, etc. Much of our previous work has focused on developing implicit representations for systems and sets of states. These representations make it possible for us to verify systems with up to about 10^{100} states. In past reports we described search techniques that use these representations to explore efficiently the state space of systems composed of asynchronous processes. Recently, we have begun to apply these techniques to verify some of the protocols used in telephone services. Similar work based on explicit state space representations has been conducted in the past, but by using our implicit representations, we hope to be able to handle larger, more realistic protocols.

Sequential chemical process control systems are verified using a model based approach that automatically determines whether the system, including both the process equipment and control system hardware and software, satisfies safety and operability specifications. The specification is expressed in temporal logic. A model checking algorithm is used to verify the system automatically against the specification. In [Moon et al. 92] we applied the method to chemical process control systems including a simple combustion system and an ‘‘alarm acknowledge’’ system to reveal discrete event errors.

3.3.2 Verification of Real-Time Concurrent Systems

Verification methodologies for real-time systems can be classified according to whether they are based on a continuous time model or a discrete time model. Continuous time often provides a more accurate model of physical reality, while discrete time can be more efficient to implement in an automatic verifier based on state exploration techniques. Choosing a model appears to require a compromise between efficiency and accuracy. In the thesis [Burch 92b] we avoid this compromise by constructing discrete time models that are conservative approximations of appropriate continuous time models. Thus, if a system is verified to be correct in discrete time, then it is guaranteed to also be correct in continuous time. We also show that models with explicit simultaneity can be conservatively approximated by models with interleaving semantics.

Proving these results requires constructing several different domains of agent models. We have devised a new method for simplifying this task, based on abstract algebras we call *trace algebra* and *trace structure algebra*. A trace algebra has a set of traces as its carrier, along with operations of projection, and renaming on traces. A trace can be any mathematical object that satisfies certain simple axioms, so the theory is quite general. A trace structure consists, in part, of a subset of the set of traces from some trace algebra. In a trace structure algebra, operations of parallel composition, projection, and renaming are defined on trace structures in terms of the operations on traces. General methods for constructing conservative approximations are described and are applied to several specific real-time models. We believe that trace algebra is a powerful tool for unifying many models of concurrency and abstraction beyond the particular ones described in this thesis.

3.3.3 A Theorem Prover for Mathematica

Analytica (described in [Clarke and Zhao 93]) is an automatic theorem prover for theorems in elementary analysis. The prover is written in the Mathematica language and runs in the Mathematica environment. The goal of the project is to use a powerful symbolic computation system to prove theorems that are beyond the scope of previous automatic theorem provers. The theorem prover is also able to guarantee the correctness of certain steps that are made by the symbolic computation system and therefore prevent common errors like division by a symbolic expression that could be zero. We describe the structure of Analytica and explain the main techniques that it uses to construct proofs. We have tried to make the paper as self-contained as possible so that it will be accessible to a wide audience of potential users. We illustrate the power of our theorem prover by several nontrivial examples including the basic properties of the stereographic projection and a series of three lemmas that lead to a proof of Weierstrass's example of a continuous, nowhere-differentiable function. Each of the lemmas in the latter example is proved completely automatically.

In [Clarke et al. 93a] we consider the language inclusion and equivalence problems for six different types of ω-automata: Buchli, Muller, Rabin, Streett, the L-automata of Kurshan, and the $\forall$-automata of Manna and Pnueli. We give a 6#×#()6 matrix in which each row and column is associated with one of these types of automata. The entry in the i^{th} row and j^{th} column is the complexity of showing inclusion between the i^{th} type of automaton and the j^{th}. Thus,for example, we give the complexity of showing language inclusion and equivalence between a Buchi automaton and a Muller or Streett automaton. Our results are obtained by a uniform method that associates a formula of the computation tree logic CTL with each type of automaton. Our algorithms use a *model checking* procedure for the logic with the formulas obtained from the automata.

The results of this paper are important for verification of finite state concurrent systems with fairness constraints. A natural way of reasoning about such systems is to model the finite state program by one ω-automaton and its specification by another.

Other work, done in conjunction with the VLSI group, is reported in Chapter 2.

3.4 Security

In the Dyad project we use physically secure coprocessors to realize new protocols and systems addressing a number of perplexing security problems. The coprocessors can be produced as boards or IC chips and can be directly inserted in standard workstation or PC-style computers. Such physically secure coprocessors permit easily implementable solutions to a variety of security problems:

- Protecting the integrity of publicly accessible workstations.
- Tamper-proof accounting/audit trails.
- Copy protection.
- Electronic currency without centralized servers.

We outline the architectural requirements for the use of secure coprocessors in [Tygar and Yee 93a].

The notion of time is a prerequisite ingredient for describing and verifying the security properties of key management protocols. Without a notion of time properties relating to the expiration of keys and the freshness of messages are meaningless and cannot be verified. In [Heintze and Tygar 92] we approach this problem by developing a model theory for protocol security. Our model differs substantially from previous ones appearing in the protocol verification literature in that it includes a notion of time. Also, unlike other models that were closely tied to verification tools and included many implicitly assumptions about types of protocols and methods for breaking them, our approach is more general and focuses on a justification of the notion of model itself. Our definition allows us to evaluate the ‘‘logic of authentication’’ formalism and find places where it is unsound and incomplete.

The reached the following milestones as part of an exchange with IBM, which is considering potential commercial applications of some of our research results:

- The DMA port for our secure coprocessor board in the Dyad project has been completed. This is the single largest portion of the Mach 3.0 port onto this board. We anticipate that applications will be running on the board by early fall.
- We completed a design for the ‘‘negotiations’’ language in Dyad, allowing a secure coprocessor board to enter into commitments with other boards to perform operations.
- We developed a procedure for secure boot of the Dyad secure coprocessor board.
- We received rights from IBM to redistribute source code developed by us and IBM on this project without restriction.

- We made substantial progress in porting the Mach operating system onto our secure coprocessor hardware.
- We defined a language and set of cryptographic protocols for exchanging services and permitting secure remote execution between machines that do not trust each other.
- We provided mechanisms for electronic billing, electronically exchanging tokens (which we call ‘‘electronic currency’’), and providing one-way messages (which we call ‘‘electronic stamps’’) that support token exchange.

In addition, we re-examined issues of authentication and thoroughly studied both the Kerberos authentication system and zero-knowledge authentication. The Kerberos study highlighted several security flaws in that widely used protocol package [Kay and Tygar 93].

Also, we continued our work on secure clock mechanisms for demonstrating causality in systems. We invented a secure clock mechanism called secure vector timestamps [Smith and Tygar 92], allowing us to simplify a number of complicated distributed protocol questions and to provide Orange Book (TCSEC) style confinement for our system.

We examined the structure of our system in light of the new proposed Federal Criteria for Information Technology Security (FCITS). We expect this to lead both to some revisions in our system design and to comments on that FCITS proposal. In [Tygar 92] we review new developments in computer security architectures, including zero-knowledge proofs, secure coprocessors, and high-availability systems.

We have also begun investigating new models of access-control protection that use notions of anonymous, correlated, and relative security. By these terms we mean that access to certain types of devices should be temporal in nature and related to one’s current state of access rights to other objects. For example, in a mobile computing environment, one may have temporary access to a certain range of display objects outside of the mobile computation device, and while one has access to those display objects, one prevents others hijacking those display objects or having access to them—even though they may have legitimate access to the same objects at other times.

Our continued pursuit of security through the use of secure coprocessors in the context of the Dyad project has focused on providing mechanisms for electronic currency, billing, and electronic contracts over networks. We have developed protocols for full autonomous use of electronic currency—without any required central ‘‘bank’’ servers [Tygar and Yee 93a].

As a dramatic example of the electronic currency work, we are currently investigating with the US Postal Service the use of encrypted packets of information in 2-D bar codes (such as PDF 417) to replace standard postage-meter franking indicia. These packets encode electronic currency token values that completely protect the tokens from forgery, replication, or fraud, while allowing support for financial anonymity (if desired).

3.5 Bibliography

[Bose et al. 92] Bose, S., E.M. Clarke, D.E. Long, and S. Michaylov.
PARTHENON: A parallel theorem prover for non-Horn clauses.
Journal of Automated Reasoning 8(2):153-182, 1992.

[Burch 91a] Burch, J.R.
Using BDDs to verify multipliers.
In *Proceedings of the 28th ACM/IEEE Design Automation Conference*. IEEE, June, 1991.

[Burch 91b] Burch, J.R.
Verifying liveness properties by verifying safety properties.
In *Computer-Aided Verification, Proceedings of the 1990 Workshop, Volume 3 of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1991.

[Burch 92a] Burch, J.R.
Delay models for verifying speed-dependent asynchronous circuits.
In *Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*. Princeton University, March, 1992.
Also in IEEE International Conference on Computer Design, October 1992.

[Burch 92b] Burch, J.R.
Trace algebra for automatic verification of real-time concurrent systems.
PhD thesis, School of Computer Science, Carnegie Mellon University, August, 1992.

[Burch et al. 91] Burch, J.R., E.M. Clarke, and D.E. Long.
Representing circuits more efficiently in symbolic model checking.
In *Proceedings of the 28th of the ACM/IEEE Design Automation Conference*. ACM/IEEE, 1991.

[Burch et al. 92] Burch, J.R., E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang.
Symbolic model checking: 10^{20} states and beyond.
Information and Computation 98(2):142-170, 1992.

[Chen et al. 92] Chen, B., A. Borg, and Jouppi.
A simulation-based study of TLB performance.
In *Proceedings of 19th International Symposium on Computer Architecture*. 1992.

[Clarke 93a] Clarke, E.M.
Analytica: A Theorem Prover for Mathematica.
The Mathematica Journal 3(1):56-71, 1993.

[Clarke 93b] Clarke, E.M.
Verification Tools for Finite-State Concurrent Systems.
In *REX '93 School Workshop: A Decade of Concurrency*. Springer Lecture Notes in Computer Science, June, 1993.

[Clarke and Fujita 93]
Clarke, E.M., and Fujita, M.
Application of BDDs to CAD for Digital Systems.
Journal of the Information Processing Society of Japan 34(5):609-616, 1993.

[Clarke and Zhao 93]
Clarke, E., and X. Zhao.
Analytica—a theorem prover for mathematica.
The Mathematica Journal 3(1), 1993.

[Clarke et al. 91a] Clarke, E.M., D.E. Long, S. Kimura, S. Michaylov, S.A. Schwab, and J.P. Vidal.
Parallel symbolic computation on shared memory multiprocessors.
In *In Proceedings of the International Symposium on Shared Memory Multiprocessors*. ISSMM, April, 1991.
Also in Shared Memory Multiprocessing, MIT Press 1992 .

[Clarke et al. 91b] Clarke, E.M., D.E. Long and K.L. McMillan.
A language for compositional specification and verification of finite state hardware controllers.
In *In Proceedings IEEE*. IEEE, September, 1991.

[Clarke et al. 92] Clarke, E.M., J.R. Burch, O. Grumberg, D.E. Long, and K.L. McMillan.
Automatic verification of sequential circuit designs.
In *Proceedings of the Royal Society of London*. 1992.

[Clarke et al. 93a] Clarke, E.M., Draghicescu, I.A., and Kurshan, R.P.
A Unified Approach for Showing Language Containment And Equivalence between Various Types of ω-Automata.
Information Processing Letters , 1993.

[Clarke et al. 93b] Clarke, E., Filkorn, T. and Jha, S. .
Exploiting symmetry in symbolic model checking.
1993.

[Clarke et al. 93c] Clarke, E.M., Grumberg, O., and Long, D.E.
Model checking and abstraction.
1993.

[Clarke et al. 93d] Clarke, E.M., Burch, J., Long, D., McMillan, K., and Dill, D.
Symbolic Model Checking for Sequential Circuit Verification.
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems , 1993.

[Clarke et al. 93e] Clarke, E.M., Yoneda, T., Shibayama, A., and Schlingloff, H.
Efficient Verification of Parallel Real-Time Systems.
In *Proceedings the of Conference on Computer-Aided Verification*. June, 1993.

[Clarke et al. 93f] Clarke, E.M., Filkorn, T., and Jha, S.
Exploiting Symmetry in Temporal Logic Model Checking.
In *Proceedings of the Conference on Computer-Aided Verification*. June, 1993.

[Cooper et al. 92a] Cooper, E., S. Nettles, and I. Subramanian.
Improving the performance of SML garbage collection using application-specific virtual memory management.
In *ACM Lisp and Functional Programming*. ACM, June, 1992.

[Cooper et al. 92b] Cooper, E., Nettles, S.M., and Subramanian, I.
Improving the performance of SML garbage collection using application-specific virtual memory management.
In *Proceedings of the ACM Conference on Lisp and Functional Programming*, pages 43-52. June, 1992.

[Dean 92] Dean, D.
A file system in standard ML.
In *ACM SIGPLAN Workshop on ML and its Applications*. ACM, June, 1992.

[Detlefs et al. 91] Detlefs, Herlihy and Wing.
Avalon C^{++}.
Advanced Language Implementation: Recent Research at Carnegie Mellon University , 1991.

[Eppinger et al. 91]
Eppinger,Mummert and Spector.
Camelot and Avalon: A Distributed Transaction Facility.
The Morgan Kaufmann Series in Data Management Systems.
Morgan Kaufmann Publishers, Inc., 1991.

[Grumberg and Long 91]
Grumberg, O., and D.E. Long.
Model checking and modular verification.
In *Proceedings of CONCUR '91: 2nd International Conference on Concurrency Theory*. Volume 527 in Lecture Notes in Computer Science, Springer-Verlag, August, 1991.

[Guattery and Miller 92]
Guattery, S.M., and G.L. Miller.
A contraction procedure for planar directed graphs.
In *Proceedings of the Symposium on Parallel Algorithms and Architectures*. 1992.

[Gupta 91] Gupta, Aarti.
Formal Hardware Verification Methods: A Survey.
Technical Report CMU-CS-91-193, School of Computer Science, Carnegie Mellon University, October, 1991.

[Guttag and Horning 93]
Guttag, J.V., and Horning, J.J. (editors).
Larch: Languages and Tools for Formal Specification.
Springer-Verlag, 1993.

[Guttag et al. 93] Guttag, J.V. and Horning, J.T., with Garland, S.J., Jones, K.D., Modet, A., and Wing, J.M.
Larch: Languages and Tools for Formal Specification.
Springer-Verlag Texts and Monographs in Computer Science.
Springer-Verlag, 1993.

[Haines 92] Haines, N.
Parsing and ML: A position paper.
In *ACM SIGPLAN Workshop on ML and its Application*. ACM, June, 1992.

[Heintze and Tygar 92]
Heintze, N.C. and J.D. Tygar.
Timed models for protocol security.
Technical Report CMU-CS-92-100, School of Computer Science, Carnegie Mellon University, 1992.

[Herlihy and Wing 91]
Herlihy, M.P., and J.M. Wing.
Specifying Graceful Degradation.
IEEE Transactions on Parallel and Distributed Computing ():, January, 1991.

[Herlihy et al. 91] Herlihy, Ling and Wing.
Implementation of Commit Timestamps in Avalon.
Technical Report CMU-CS-91-107, School of Computer Science, Carnegie Mellon University, January, 1991.

[Heydon and Tygar 92]
Heydon, C.A., and J.D. Tygar.
Specifying and checking Unix security constraints.
In *USENIX Third Security Symposium*. 1992.

[Heydon et al. 90a]
Heydon, A., M. Maimone, D. Tygar, J.M. Wing, and A. Moorman Zaremski.
Miro: Visual Specifications of Security.
IEEE Transactions on Software Engineering ():1185-1197, October, 1990.

[Heydon et al. 90b]
Heydon, A., A. Maimone, D. Tygar, J.M. Wing, and A.M. Zaremski.
Miro: Visual Specifications of Security.
IEEE Transactions on Software Engineering 16(10):1185-1197, 1990.

[Kay and Tygar 93]
Kay, J., and D. Tygar.
Are Kerberos tickets good for a free ride?.
Technical Report, School of Computer Science, Carnegie Mellon University, 1993.

[Kimura and Clarke 90]
Kimura, S., and E.M. Clarke.
A parallel algorithm for constructing binary decision diagrams.
In *1990 IEEE International Conference on Computer Design*. IEEE, September, 1990.

[Kistler and Satyanarayanan 91]
Kistler, J.J., and M. Satyanarayanan.
Disconnected operation in the Coda file system.
ACM Transactions on Computer Systems 10(1), 1991.
Presented at the 13th ACM Symposium on Operating Systems Principles in October, 1991. Also appeared as Technical Report CMU-CS-91-166.

[Krueger 90] Krueger, C.W.
Persistent Long-term Transactions for Software Development.
Technical Report CMU-CS-90-188, School of Computer Science, Carnegie Mellon University, December, 1990.

[Krueger 91] Krueger.
Application-Specific Architectures for Object Databases.
In *Proceeding of the DARPA/TI Workshop on an Open Architecture for Object Database Systems*. DARPA, March, 1991.

[Kumar and Satyanarayanan 91]
Kumar, P., and M. Satyanarayanan.
Log-Based Directory Resolution in the Coda File System.
Technical Report CMU-CS-91-164, School of Computer Science, Carnegie Mellon University, December, 1991.

[Kumar and Satyanarayanan 93a]
Kumar, P., and Satyanarayanan, M.
Application-Specific Resolution in an Optimistically Replicated File System.
In *Proceedings of the 4th IEEE Workshop on Workstation Operating Systems*. IEEE, October, 1993.

[Kumar and Satyanarayanan 93b]
Kumar, P., and Satyanarayanan, M.
Log-based directory resolution in the Coda file system.
In *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*. January, 1993.

[Liskov and Wing 92]
Liskov, B.H., and Wing, J.M.
Family values: a semantic notion of subtyping.
Technical Report CMU-CS-92-220, School of Computer Science, Carnegie Mellon University, December, 1992.
Also published as MIT-LCS-TR-562.

[Liskov and Wing 93a]
Liskov, B.H., and Wing, J.M.
A new definition of the subtype relation.
In *Proceedings of ECOOP '93*. 1993.

[Liskov and Wing 93b]
Liskov, B.H., and Wing, J.W.
Family Values: A Behavioral Notion of Subtyping.
Technical Report CMU-CS-93-187, School of Computer Science, Carnegie Mellon University, July, 1993.
Supercedes CMU-CS-93-149 and CMU-CS-92-220. Also submitted to ACM Transactions on Programming Languages and Systems.

[Liskov and Wing 93c]
Liskov, B.H., and Wing, J.W.
Specifications and Their Use in Defining Subtypes.
In *Proceedings of OOPSLA '93*. September, 1993.

[Liskov and Wing 93d]
Liskov, B.H., and Wing, J.W.
A New Definition of the Subtype Relation.
In *Proceedings of ECOOP '93*. July, 1993.
Also available as Technical Report CMU-CS-93-149 and MIT LCS Programming Methodology Group Memo 76, May 1993.

[Martin and Wing 93]
Martin, U., and Wing, J.M. (editors).
Proceedings of the First International Workshop on Larch.
Springer-Verlag, 1993.
Workshops in Computing Series.

[McGeoch and Tygar 90]
McGeoch, and D. Tygar.
Optimal Sampling Strategies for Quicksort.
In *Proceedings of the 28th Annual Allerton Conference on Communication, Control, and Computing*. Allerton, October, 1990.

[McMillan and Schwalbe 91]
McMillan and Schwalbe.
Formal verification of the encore gigamax cache consistency protocal.
In *Proceedings of the International Symposium on Shared Memory Multiprocessors*. ISSMM, April, 1991.

[Moon et al. 92] Moon, I., G.J. Powers, J.R. Burch, E.M. Clarke.
Automatic Verification of Sequential Control Systems using Temporal Logic.
American Institute of Chemical Engineers Journal , 1992.

[Morrisett 93] Morrisett, J.G.
Generalizing First-Class Stores.
In *Proceedings of the ACM SIGPLAN Workshop on State in Programming Languages*. ACM, June, 1993.
Also available as technical report YALEU/DCS/RR-968, Department of Computer Science, Yale University.

[Morrisett and Tolmach 92]
Morrisett, J.G., and A. Tolmach.
A portable multiprocessor interface for Standard ML of New Jersey.
Technical Report CMU-CS-92-155, School of Computer Science, Carnegie Mellon University, June, 1992.

[Morrisett and Tolmach 93a]
Morrisett, J.G., and Tolmach, A.
Procs and locks: a portable multiprocessor interface for Standard ML of New Jersey.
In *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. 1993.

[Morrisett and Tolmach 93b]
Morrisett, J.G., and Tolmach, A.
Procs and Locks: A Portable Multiprocessing Platform for Standard ML of New Jersey.
In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. ACM, May, 1993.

[Nettles 92] Nettles, S.M.
A Larch specification of copying garbage collection.
Technical Report CMU-CS-92-219, School of Computer Science, Carnegie Mellon University, December, 1992.

[Nettles and O'Toole 93]
Nettles, S.M., and O'Toole, J.W.
Replication-based real-time garbage collection.
1993.
In PLDI93.

[Nettles and Wing 92a]
Nettles, S.M., and J.M. Wing.
Persistence + Undoability = Transactions.
In *Proceedings of Hawaii International Conference on Systems Science 25*. January, 1992.
Also appears as Technical Report CMU-CS-91-173.

[Nettles and Wing 92b]
Nettles and Wing.
Persistence + Undoability = Transactions.
Technical Report HICSS-25, School of Computer Science, Carnegie Mellon University, January, 1992.

[Nettles et al. 92a] Nettles, S.M., J. O'Toole, D. Pierce, and N. Haines.
Replication-based incremental copying collection.
In *Proceedings of the Workshop on Memory Management*. September, 1992.

[Nettles et al. 92b] Nettles, S.M., O'Toole, J.W., Pierce, D., and Haines, N.
Replication-based incremental copying collection.
In *Proceedings of the SIGPLAN International Workshop on Memory Management*, pages 357-364. ACM, Springer-Verlag, September, 1992.

[Nettles et al. 93] Nettles, S., O'Toole, J., and Gifford, D.
Concurrent Garbage Collection of Persistent Heaps.
Technical Report CMU-CS-93-137, School of Computer Science, Carnegie Mellon University, 1993.
Also Accepted for the Proceedings of the ACM Symposium on Operating Systems, to appear December 1993.

[O'Toole and Nettles 93]
O'Toole, J., and Nettles, S.
Concurrent Replication Garbage Collection.
In *Proceedings of the ACM Symposium on Programming Language Design and Implementation*. ACM, July, 1993.
Also available as Technical Report CMU-CS-93-136.

[Program Specification 91]
Wing.
Encyclopedia of Computer Science and Technology.
1991.

[Reif et al. 90] Reif, D. Tygar, and T. Yoshida.
The Computability and Complexity of Optical Beam Tracing.
In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*. SFCS, October, 1990.

[Rollins and Wing 90]
Rollins, E.R., and J.M. Wing.
Specifications as Search Keys for Software Libraries: A Case Study Using Lambda Prolog.
Technical Report CMU-CS-90-159, School of Computer Science, Carnegie Mellon University, September, 1990.
Also in Proceedings of the International Conference on Logic Programming, June 1991.

[Satyanarayanan 92]
Satyanarayanan, M.
The Influence of Scale on Distributed File System Design.
IEEE Transactions on Software Engineering :1-8, 1992.

[Satyanarayanan 93]
Satyanarayanan, M.
Mobile Computing.
IEEE Computer, Hot Topics , September, 1993.

[Satyanarayanan et al. 92a]
Satyanarayanan, M., J. Kistler, P. Kumar, and H. Mashburn.
On the ubiquity of logging in distributed file systems.
In *Proceedings of the 3rd IEEE Workshop on Workstation Operating Systems*. IEEE, May, 1992.

[Satyanarayanan et al. 92b]
Satyanarayanan, M., Steere, D.C., Kudo, M., and Mashburn, H.
Transparent logging as a technique for debugging complex distributed systems.
In *Fifth ACM SIGOPS Workshop*. ACM, September, 1992.

[Satyanarayanan et al. 93a]
Satyanarayanan, M., Mashburn, H.H., Kumar, P., Steere, D.C., and Kistler, J.J.
Lightweight Recoverable Virtual Memory.
In *14th ACM Symposium on Operating Systems Principles*. ACM, December, 1993.
Nominated to ACM Transactions on Computer Systems.

[Satyanarayanan et al. 93b]
Satyanarayanan, M., Kistler, J.J., Mummert, L.B., Ebling, M.R., Kumar, P., and Lu, Q.
Experience with Disconnected Operation in a Mobile Computing Environment.
In *Proceedings of the 1993 USENIX Conference on Mobile and Location-Independent Computing*. USENIX, August, 1993.

[Schwab 91]
Schwab, S.A.
Extended parallelism in the Grobner basis algorithm.
Technical Report CMU-CS-91-137, School of Computer Science, Carnegie Mellon University, 1991.

[Smith and Tygar 92]
Smith, S., and D. Tygar.
Secure vector timestamps: a framework for partial order time.
Technical Report CMU-CS-92-116, School of Computer Science, Carnegie Mellon University, 1992.

[Tygar 92]
Tygar, J.D.
New Directions in Computer Security.
Journal of Electronics, Information, and Communication Engineers , September, 1992.
Special 75th anniversary issue. In Japanese.

[Tygar and Camp 93]
Tygar, J., and Camp, J.
Protecting Privacy while Allowing Auditing.
Journal of the Information Society , 1993.

[Tygar and Yee 93a]
Tygar, D., and Yee, B.
Dyad: a system for using physically secure coprocessors.
Technological Strategies for the Protection of Intellectual Property.
Harvard University Press, 1993.
To appear.

[Tygar and Yee 93b]
Tygar, J., and Yee, B.
Cryptography: It's Not Just for Electronic Mail Anymore.
Technical Report CMU-CS-93-107, School of Computer Science, Carnegie Mellon University, March, 1993.

[Tygar et al. 91]
Tygar and Yee.
Strongbox: A System for Self-Securing Programs.
In Richard F. Rashid (editor), *Carnegie Mellon Computer Science: A 25-Year Commemorative*. Carnegie Mellon University School of Computer Science, 1991.

[Wing 90a]
Wing, J.M.
A Specifier's Introduction to Formal Methods.
IEEE Computer :8-24, September, 1990.

[Wing 90b]
Wing, J.M.
Using Larch to Specify Avalon/C^{++} Objects.
IEEE Transactions on Software Engineering 16(9):1076-1088, 1990.

[Wing 90c]
Wing, J.M.
Verifying Atomic Data Types.
In *Proceedings of the REX Workshop on Stepwise Refinement of Distributed Systems: Models, Formalism, Correctness*. REX, May, 1990.

[Wing 91] Wing, J.M.
Textual Specifications of Visual Specifications.
Formal Methods in Computer Graphics.
In Duce and Faconti,
Springer-Verlag, 1991.

[Wing 93a] Wing, J.M.
A Specifier's Introduction to Formal Methods.
Encyclopedia of Software Engineering.
John Wiley & Sons, 1993.

[Wing 93b] Wing, J.M.
Composing Transactional Concepts.
In *Proceedings of the ECOOP '93 Workshop on Object-based Distributed Programming*. July, 1993.

[Wing and Zaremski 91a]
Wing, J.M., and A.M. Zaremski.
A Formal Specification of a Visual Language Editor.
In *Proceedings of the Sixth International Workshop on Software Specification and Design*. October, 1991.
Also appears as Technical Report CMU-CS-91-112.

[Wing and Zaremski 91b]
Wing, J.M., and A.M. Zaremski.
Unintrusive Ways to Integrate Formal Specifications in Practice.
In *Proceedings VDM '91, Lecture Notes in Computer Science 551*. Springer-Verlag, October, 1991.
Also appears as Technical Report CMU-CS-91-113.

[Wing and Zaremski 91c]
Wing, J.M., and A.M. Zaremski.
A Formal Specification of a Visual Language Editor.
In *Proceedings of the Sixth International Workshop on Software Specification and Design*. October, 1991.
Also as Technical Report CMU-CS-91-112.
.

[Wing et al. 91] Wing, J.M., et al.
The Avalon Language.
Camelot and Avalon: A Distributed Transaction Facility.
In Eppinger, Mummert and Spector,
Morgan Kaufmann, 1991.
Part IV, Chapters 19-22.
.

[Wing et al. 92a] Wing, J.M., M. Faehndrich, J.G. Morrisett, S. Nettles.
Extensions to standard ML to support transactions.
In *ACM SIGPLAN Workshop on ML and its Applications*. ACM, June, 1992.
Also available as Technical Report CMU-CS-92-132.

[Wing et al. 92b] Wing, J.M., E.R. Rollins, A.M. Zaremski.
Thoughts on Larch/ML and a new application for LP.
In *Proceedings of the First International Workshop on Larch*. July, 1992.
Also available as Technical Report CMU-CS-92-135.

[Wing et al. 92c] Wing, J.M., Rollins, E.R., and Zaremski, A.M.
Thoughts on a Larch/ML and a new application for LP.
In *Proceedings of the First International Workshop on Larch*. Dedham, July, 1992.
Also Technical Report CMU-CS-92-135.

[Wing et al. 93] Wing, J.M., Faehndrich, M., Haines, N., Kietzke, K., Kindred, D., Morrisett, J.G., and Nettles, S.
Venari/ML Interfaces and Examples.
Technical Report CMU-CS-93-123, School of Computer Science, Carnegie Mellon University, March, 1993.

[Zaremski and Wing 93]
Zaremski, A.M., and Wing, J.M.
Signature Matching: A Key to Reuse.
Technical Report CMU-CS-93-151, School of Computer Science, Carnegie Mellon University, 1993.
Also to appear in ACM SIGSOFT ’93 Symposium on Foundations of Software Engineering, December 1993.

4. RESEARCH IN INTEGRATED ARCHITECTURES FOR INTELLIGENT SYSTEMS

Architectures capable of supporting integrated intelligent systems (*integrated architectures* for short) have now emerged as serious contenders to become a central line in the development of advanced, artificially intelligent systems. An integrated architecture combines into a seamless whole the various reasoning, sensing, and learning capabilities developed individually in AI, including:

- Problem solving, planning, knowledge-intensive operation, acquisition of new tasks, etc.
- Direct interaction with the external world through natural language and robotic sensors and effectors.
- Learning by experience throughout all aspects of system behavior.

Three functional integrated architectures (Prodigy, Soar and Theo) have been under active development at Carnegie Mellon during the contract period.

4.1 Modeling General Cognition with Soar

Soar is a system that formulates all tasks as search in problem spaces. It does this both for externally given tasks, such as factory scheduling or designing an algorithm, and for internal subtasks, such as selecting which operator to use, implementing an operator, or attempting to satisfy an operator precondition. Soar's ubiquitous use of problem-space allows us to view it as a problem-space architecture.

The system has a single uniform learning mechanism, chunking, that records as a new production the knowledge that resolved each impasse. Since all behavior is driven by impasses, chunking occurs continuously and produces learning on all aspects of behavior (search control, operator implementations, etc). In addition to being an AI system, Soar is also a theory of human cognition, so that many of the tasks that have been studied with it are those involved in psychological experiments.

4.1.1 Automatic Compilation of Expert Systems

[Yost 92] gathers together our experience in devising and testing an experimental language called TAQL for rapid task acquisition in Soar5. We conducted three rounds of experiments to evaluate TAQL. In the first (100-500 word descriptions), total task acquisition time, from domain-oriented description to running system, took about five minutes per rule. The rate of coding time per production remained constant in the second round (1000-5000 word descriptions) despite tenfold increase in task size. In the third round tests on midsize to large task descriptions (5000-15,000 words), the coding rate again remained constant despite the increase in task size.

In a significant demonstration of the scalability of task acquisition in Soar, one TAQL system produced during the third round of experiments was for a 5000+ word problem description taken from the literature. The task domain involved assessing the effects of electromagnetic pulses on a hardware system and advising the user on ways to harden the system against EMP. As reported in the literature, the OPS5 version of the expert system contained between 900 and 1200 productions and took 21 days to code. The TAQL system contained 247 productions, took 2 days to code, and produced identical input/output for the sample problem provided.

As part of our continuing effort to increase Soar's usability, [Altmann and Yost 92] presents a detailed tutorial for building an expert system using TAQL. This is a fully-tested, self-contained reference for the end-to-end development of a Soar system. It presents a natural-language task description, a system design, and a sample implementation, including a documented code listing. It also discusses chunking (Soar's learning mechanism) in the context of the sample implementation.

4.1.2 Direct Interaction with the Outside World

In recent years a rift has occurred in the AI and cognitive science communities between standard symbolic approaches to intelligence and highly reactive, ''situated'' approaches. The claim on the part of situated cognitivists has been that symbolic approaches are inadequate to explain or model the sorts of behaviors that consist largely of cued interaction with the environment. Extending symbolic theories to such tasks is an important step toward bridging these theoretical frameworks.

[Vera et al. 93]describes two research projects that study typical Situated Action tasks using traditional cognitive science methodologies. The two tasks are decision making in a complex production environment and interaction with an Automated Teller Machine (ATM). Together, these projects suggest that tasks studied by Situated Action research pose interesting but not unachievable challenges for traditional symbolic theories. Both tasks require that the decision maker and the user search for knowledge in the environment in order to execute their tasks. The goal of these projects is to investigate the interaction between internal knowledge and dependence on external cues in such tasks. We have used the classical expert/novice paradigm to study information search in the decision making task, and cognitive modeling to predict the behavior of ATM users.

The results of the first project strongly indicate that decision makers must rely on environmental cues (knowledge in the environment) to make decisions, independent of their expertise level. We also found that performance and information search are radically different between experts and novices. Our explanation is that prior experience in dynamic decision tasks improves performance by changing information search behavior instead of inducing superior decision heuristics. In the second study, we describe a computer model, based on the Soar cognitive architecture, that learns part of the task of using an ATM machine. The task is performed using only the external cues available from the interface itself, and knowledge assumed of typical human users (e.g., how to read, how to push buttons).

Natural Language

[Lehman et al. 91a] introduced NL-Soar, Soar’s natural language system. It incorporates a single-path parsing algorithm, top-down and bottom-up knowledge, and a limited repair capability for correcting wrong interpretations. These characteristics closely match what is required for real-time comprehension. In addition the system relies on Soar’s learning mechanism to transform its behavior from deliberate search to ‘‘recognitional’’ processing. Integration of knowledge sources is an acknowledged method for controlling the size of the search for an interpretation. By achieving integration automatically through Soar’s learning mechanism, however, NL-Soar’s method has the additional property that it permits both new knowledge sources and new knowledge within a source to be added to the system in an incremental and modular way. This is key to direct interaction with the external world for two reasons. First, from a pure engineering point of view, NL-Soar’s two-faceted knowledge integration makes it easier to create and maintain a system with increasingly broad coverage without losing the essential real-time characteristic of comprehension. Second, from a theoretical point of view, it makes it possible to extend the system language through direct interaction and have the acquired linguistic knowledge integrated into future understanding exactly as if it had been programmed. [Steier et al. 92] examines this integration issue further in Soar, including NL-Soar’s solution to the problem.

Throughout the contract, we have tested NL-Soar on an expanding corpus, with grammar and lexicon augmented accordingly. As the size of the corpus grows, we are increasingly confident in the robustness and generality of our approach. The system has demonstrated it can handle the larger data set without significant modification. This included both more complex syntactic constructions and multisentence inputs.

In addition to the basic implementation of a comprehension system in Soar, we have examined the role of language in cognition, uncovering two phenomena inherent in Soar’s language capability: *linguistic task operators* and *taskification*. Linguistic task operators (LTOs) provide a method for problem solving by deploying the skills of comprehension and generation in service of a task. ‘‘Taskification’’ is a process by which task-specific knowledge can be smoothly integrated into comprehension, modifying and specializing the meaning of words and facilitating communication with experts. Each of these mechanisms, whose possibility was previously unsuspected, will play a major role in the effective use of natural language by an intelligent agent [Lehman et al. 92a].

In [Lehman et al. 92b] we analyze the ways in which the Soar architecture influenced the design of NL-Soar and conclude that there are hard requirements (such as comprehension real-time constraint) that restrict certain design choices at the architectural level. The issue of how strong a role particular architectures play in the creation of large, complex systems is a central concern of the field; this paper is essentially a case study addressing that concern.

In [Lewis 92] we describe a theory of garden path phenomena that has emerged from the work on NL-Soar, a computational model of language comprehension. The idea behind building a cognitively plausible comprehension system is simple: Humans are our best exemplar of fast, flexible language understanders. In understanding how they perform this feat, it is crucial to examine both what people can comprehend and what they cannot. Garden path phenomena are a class of sentences that cause problems for human comprehension; as such, they provide a constraint on the power of the model. If our models do not have problems comprehending these sentences, then the models are too powerful and may suffer from more global inefficiencies as a result. In addition there is a class of syntactic constructions that appear similar to garden path constructions but which do not cause the same comprehension problems. Ideally, then, a system that has the same flexibility and efficiency as the human comprehender should fail to comprehend garden path sentences but not their nongarden path counterparts. The current NL-Soar performs with about 95% predictive accuracy on the garden and nongarden path corpora.

The work in [Lewis 92] was extended in [Lewis 93]. In this paper we lay out the basic arguments and mechanisms for making NL-Soar a detailed computational model that provides an account of a large range of sentence-level phenomena: immediacy of interpretation, garden path effects, unproblematic ambiguities, parsing breakdown on difficult embeddings, acceptable embedding structures, and both modular and interactive ambiguity resolution effects. The theory goes beyond explaining just a few examples and addresses over 80 different kinds of constructions. Further, we explain why Soar is not merely an implementation language for the model, but plays a central theoretical role. The predictive power of NL-Soar derives largely from architectural mechanisms and principles that shape the comprehension capability so that it meets the real-time constraint.

Using External Software Systems

[Newell and Steier 91] makes that case that, in general, the ability to use external software systems is important to Soar's success as an intelligent, integrated agent because it allows Soar to use existing, special-purpose programs to augment its task knowledge and problem-solving skills. As a specific example of external software use, [Doorenbos et al. 92] discusses the addition of capabilities to Soar for reading and writing to external Structured Query Language (SQL) databases. Soar invokes the database program as a child process and communicates with it through input/output when a database-query operator is selected. The form of this operator is an annotated model, similar to an entity/relationship specification of the database, with unknowns for the desired properties.

Agent-Tracking

A final area of progress in this field was the completion of a Soar system that performs agent-tracking within its domain of intelligent tutoring for electrostatics [Ward 91]. The system is able to model the student's dynamic behavior during the tutorial as a path through its representation of legitimate action sequences. This is a difficult problem both because most actions can be part of many sequences and because students may leave actions out altogether. By detecting the probable action sequence intended by the student, the tutor can provide guidance and remediation in the face of student errors. The success of the system was demonstrated empirically in experiments comparing the Soar tutor's performance with the performance of human tutors: Students working with the Soar system did as well or better on a standard set of electrostatics problems as did those students working with human tutors.

4.1.3 Machine Learning

In [Altmann 93a] we motivate a metric for comparing multistrategy learning systems, presenting a high-level approach to two problems facing such systems: managing the complexity of multiple interacting learning components and managing the cost of implementing them. We demonstrate that analysis of the task domain can generate constraints on how performance and learning components interact. This analysis can be verified with respect to human performance, providing a standard against which to measure the scope of learning in the system. The marginal cost of adding new varieties of learning to a system can be reduced by using sharable learning components, like those embedded in general problem-solving architectures. These approaches require a meaningful definition of what constitutes learning variety, in order that systems can be compared and that system complexity and implementation cost can be evaluated with respect to the scope of learning that the system achieves. We then propose two constituents of learning scope, density and diversity.

4.1.4 Enabling Technology

Eliminating the Combinatorics of Match

In [Tambe 91a] we analyzed several alternates the existing pattern matching technology that lies at the heart of the architecture. Match complexity is one of the single most important factors in determining the efficiency of Soar. We have found that match combinatorics can be eliminated by restricting the representation in various ways (for example, by forcing all sets in the underlying representation of attributes and values to be structured). The resulting uni-attribute representation made possible Uni-Rete, a recasting of Rete technology. This new match algorithm performs up to ten times faster than Soar5's standard matcher. Alternate schemes that provide significant increases in speed while requiring only small sacrifices in expressiveness are also explored.

Scaling Up: The Very Large Systems Effort (VLSE)

An initial exploration into very large learning systems is reported in [Doorenbos et al. 92]. The paper examines the performance of Dispatcher-Soar as it grew from its initial size of 1819 productions to 12,931 productions via Soar's chunking mechanism. This size represented, at the time, one of the largest production systems in existence and the largest number of rules ever learned by an AI system. This initial work demonstrated a number of remarkable features. For example previous data from EBL systems such as Prodigy (and including some smaller Soar systems) predict that a large number of chunks will result in a large match cost. This increase is called the average growth effect (AGE); it seems to demand a tradeoff between size and efficiency. Yet, despite a seven-fold increase in number of productions and a corresponding eight-fold increase in the Rete network, Dispatcher-Soar displayed no average growth effect.

In [Tambe et al. 92] we examine the remarkable lack of average growth effect in Dispatcher-Soar in the larger context of rule-based systems in general. Two conflicting views exist in the literature. Research in explanation-based learning (EBL) indicates that learned rules add to the match cost of a production system. Thus, as the system size increases with learning, the match cost will increase as well, with the overall performance of the system eventually degrading. Research in parallel production systems, on the other hand, concludes that the match effort in a production system is limited and so increase in size will not result in increased match cost. There is a great deal of literature supporting both views. One of the problems in evaluating the claims is the lack of comparability between systems in each paradigm. We have now built two very large systems within the common framework of Soar: one displaying the average growth effect and one not. The use of a common framework increases the potential for resolving the issue in a qualitative way, enabling us to predict under what circumstances (for what kinds of domains, knowledge representations, match techniques, etc.) the average growth effect will or will not appear.

[Doorenbos 93] extends our previous work with very large systems begun in [Doorenbos et al. 92], this time by examining several systems which learn a large number of productions, including one which learns 113,938 rules — once again the largest number ever learned by an AI system and the largest number in any existing production system. As argued above, it is important to match these rules efficiently, in order to avoid the machine learning *utility problem*. Moreover, examination of such large systems reveals new phenomena and calls into question some common assumptions based on previous observations of smaller systems. We first show that standard match algorithms do not scale well with the number of rules in our systems, in part because the number of rules affected by a change to working memory increases with the total number of rules in these systems. We also show that the sharing of nodes in the Rete network (Soar's underlying match technology) becomes more and more important as the number of rules increases. Finally, we describe and evaluate a new optimization for Rete which improves its scalability and allows two of our systems to learn over 100,000 rules without significant performance degradation.

As part of comparing and contrasting the approaches to intelligence in Soar and Prodigy, [Doorenbos and Veloso 93a] examines results in three speedup learning systems: Soar, Prodigy/Analogy, and Prodigy/EBL. Speedup learning involves the acquisition of new knowledge from experience in order to improve the future performance of a problem solver. The paper differentiates between three aspects of this learning process: the construction of new knowledge, its storage and retrieval in the knowledge base, and finally, its later reuse. The tradeoff between the benefits of blindly increasing the size of the knowledge base and the costs of retrieving (or matching) its contents often leads to the *utility problem* (where the cost of learned knowledge exceeds the cost of the search it replaces). To avoid the utility problem the speedup learning community has paid much attention to construction and reuse but has often overlooked the storage and retrieval aspects. Our recent work has studied the use of organization and indexing techniques at storage time, together with efficient matching or locating methods at retrieval time, to reduce or avoid the utility problem.

Soar6

In [Milnes et al. 92], we make available the formal specification of the Soar architecture that was constructed to elucidate and clarify the definition of Soar and to guide its re-implementation in **C**. The specification is given in Z, a model theoretic specification language based in set theory that has syntax and type checking programs available. The specification has a complete coverage of the architecture, a low level of abstraction and a considerable implementation bias. Soar6 is now in general release to the academic and government research communities. Initial estimates of speed up range from a factor of 5 to a factor of 20.

Documentation of Soar

A number of more general works document the Soar architecture, and, in particular, its status as a unified theory of cognition. These include [Newell 90], [Newell 92a], [Newell 92b], and [Newell 93].

4.2 Rapid External Reactivity in Theo

The goal of the Theo architecture is to incorporate machine learning mechanisms within a problem-solving architecture using a frame-based representation. During this contract we have significantly expanded Theo's learning capabilities, completely reimplemented the system (producing an order of magnitude improvement in performance), and explored the application of Theo's learning methods to problems in robot control, database learning, and interactive learning apprentices. The five major milestones achieved during the contract period are:

- Extensions and improvements to the Theo architecture.
- Explanation-based neural network learning.
- Learning robot control strategies.
- Learning apprentice for office assistance.
- Database mining.

4.2.1 Theo System Architecture

We further developed the Theo architecture: a problem solving system built on a frame-based representation with attached inference and learning methods. All problem solving corresponds to inferring values of slots of frames, and all learning involves acquiring new procedures for inferring slot values. For example, to formulate the problem of robot control in Theo, a ROBOT frame is defined that contains slots such as OBSERVED-SONAR-READINGS, CURRENT-GOAL, and NEXT-ACTION. The problem of computing the correct robot action is cast as the problem of inferring the correct value for the NEXT-ACTION slot. Learning methods can contribute new Prolog rules, decision trees, or neural networks to infer slot values, and can be applied uniformly to any slot within the system. Thus, once a problem solving system is developed within Theo, it is easy to incorporate learning methods for any aspect of the task. Main accomplishments in the basic Theo system in the past three years included:

- Expansion of Theo's learning methods. Learning methods now include caching, explanation-based learning (EBG), decision tree learning (ID3/C4.5), learning of first order rules (MFOIL), induction of neural networks (Backpropagation), and explanation-based neural network learning (EBNN). All of these methods can be applied to learn rules or networks for any slot of any Theo frame.
- Complete reimplementation and simplification of the Theo system, resulting in a faster, smaller implementation called TheoGT. TheoGT is one to two orders of magnitude more efficient than the original Theo implementation, and is efficient and robust enough to serve as the platform for our fielded calendar manager assistant (described in 4.2.4).
- Improved documentation and dissemination. Theo and TheoGT have now been distributed to a variety of university and industrial research labs inside and outside the US. TheoGT is available free of charge from Carnegie Mellon.

4.2.2 Explanation-Based Neural-Network Learning

A major accomplishment of our research is the development of a new learning technique that unifies neural-network, inductive learning with symbolic, explanation-based learning. Inductive methods such as neural network learning have been shown to be effective for learning fairly simple functions from noisy data. In contrast symbolic explanation-based learning methods are able to learn much more complex functions from less training data but require significant prior knowledge on the part of the learner. One of the key open problems in machine learning is to unify these two paradigms in order to obtain the best features of both: ability to learn from little prior knowledge, plus ability to scale up to more complex learning tasks as more prior knowledge becomes available.

We have further developed and experimented with a new algorithm, EBNN, which successfully combines neural-network and explanation-based learning. We also incorporated EBNN into the Theo machine-learning framework, making it accessible to Theo applications such as database learning and learning apprentice applications. We presented the algorithm and described its application to reinforcement learning problems in [Thrun and Mitchell 93] and published a comparative analysis of the relationship between this EBNN algorithm (based on neural network representations) and previous explanation-based learning methods (based on symbolic representations) in [Mitchell and Thrun 93a]. While both approaches use prior knowledge to generalize more correctly from less training data, EBNN has several significant advantages, including:

- It can use approximate, self-acquired prior knowledge, whereas previous symbolic approaches are not robust to errors in prior knowledge.

- EBNN is immune to the slowdown experienced by symbolic approaches, since its learned information is represented in a fixed size network with fixed execution cost, rather than a growing collection of rules with growing match cost.

Finally, we have begun initial studies of potential application of EBNN to additional task domains, including robot control, engineering design and chess.

The potential impact of this algorithm is to expand significantly the capabilities of machine learning techniques in situations where only *approximate* prior knowledge is available to guide learning. We expect to continue exploring this algorithm and its application as part of newly funded research.

4.2.3 Learning Robot-Control Strategies

Learning to control a mobile robot has been one application area in which we have applied Theo and its learning methods. In particular, we explored two different approaches:

- Use of explanation-based learning to speed up robot decision making.
- The application of the EBNN algorithm to reinforcement learning of robot control behaviors.

We developed a Theo program to control a simple mobile robot in our laboratory and used Theo's explanation-based learning mechanism to automatically compile out stimulus-response rules. These rules produce the same decisions as Theo's general planning methods, but much more efficiently. For example, whereas planning certain actions required several minutes of CPU time, the learned rules were able to make exactly the same decisions in a few hundred milliseconds, thereby improving substantially the ability of the robot to react to changes in its environment [Mitchell 90]. The main limitation of this approach is its requirement for correct prior knowledge of the effect of robot actions. Therefore, we did not continue pursuing this approach beyond our initial investigations.

Unlike symbolic explanation-based learning methods, the EBNN method described above does not require perfect prior knowledge on the part of the learner. Instead, it itself learns a model of the effects of its actions, then uses this approximate learned knowledge to explain and generalize from successful and unsuccessful control choices. We demonstrated in a simulated robot the ability to learn significantly better strategies from less training data than competing reinforcement learning methods. We are currently transferring these algorithms to a real robot with vision, sonar, and laser range sensors to test the EBNN method's applicability with real sensor data. This line of research looks quite promising, and we are continuing it under new funding.

4.2.4 Learning Apprentice for Office Assistance

There are numerous opportunities for personalized knowledge-based assistants to improve worker productivity in many domains. For example a personal assistant for managerial/secretarial tasks might assist the user in scheduling meetings, processing purchase orders, arranging travel, searching for information on the Internet, sorting electronic mail, etc. One key practical bottleneck to obtaining such personalized assistance is the high cost of software development and maintenance for systems that must be carefully customized to each worker's tasks, preferences, and workplace.

Our approach to overcoming this overwhelming software development cost is to produce just one learning program, rather than a customized program for each user. The learning program should automatically customize its behavior to each individual user, through experience gained while in routine use. We have developed one case study: a program called CAP, which assists in managing a personal calendar. CAP automatically learns the scheduling preferences of individual users through experience. It is currently in routine use by a handful of users and has learned thousands of rules that characterize individual scheduling preferences (e.g., "IF the meeting is with a GRADUATE STUDENT and FACULTY, and this student is in COMPUTER-SCIENCE department, then the meeting will be in the FACULTY's office.")

We collected significant experimental data on CAP's ability to self-customize to several users who routinely use the system. This reflects the effect of hundreds of nightly learning sessions from about three man-years of system use.

A technical paper (in preparation) will report these results. Our main conclusions are:

- CAP is able to learn a significant number of useful scheduling rules, from a variety of users, without imposing any additional user burden over their normal use of the system as an editor.
- CAP also successfully learns to estimate its confidence in these learned rules, enabling it to selectively provide advice only when it is able to meet a user-specified confidence level.
- The main limitations include the difficulty in determining which candidate attributes should be considered by the learner (i.e., which attributes are likely to be good predictors of meeting time, location, length), the bounded ability of the system to *observe* relevant attributes (such as which seminars are scheduled by others, which people are officemates, which building locations are near each other), the fact that it requires approximately 100 training meetings before reasonable regularities are found (approximately one month of use at five meetings per day), and a high variance in the quality of regularities that can be learned for different aspects of this task (i.e., learning to predict meeting location is easier than learning to predict meeting time).

In addition to this study of the CAP system, we have begun design of a follow-on software secretary agent intended to make greater use of network resources such as online databases, Internet information sources, and email. For example, we have begun to explore the use of CAP's learned knowledge for email negotiation to schedule meetings [Bocionek and Mitchell 93]. We have begun to explore the task of learning which netnews and electronic bboard messages would be of interest to individual users, based on data such as which articles they have read in the past, which other users have read similar articles, and which articles those other users are currently reading (for this effort we have obtained a grant from the Digital Equipment Corporation to support one Ph.D. student, and are collaborating with their Jeeves newsgroup research project). Whereas our initial efforts with CAP focused on learning from passive observation of user behavior, we are now beginning to examine approaches for more active interaction, allowing users to teach the system new tasks and strategies. [Bocionek and Sassin 93] describes the preliminary design of such a system and its possible use in teaching the system how to conduct email negotiations to reserve meeting rooms.

Our ''calendar apprentice'' is the first example of a fielded knowledge-based system that learns automatically from users in routine use. It is significant as a first demonstration that learning can be of practical value for the task of automatically constructing and maintaining user-customized systems. While our preliminary results leave many issues to be studied, this technology, if successful, should be extendible to many tasks in the workplace for which site-specific and user-specific knowledge is needed.

4.2.5 Database Mining

We have demonstrated that by using Theo as an architecture for storing databases, we are able to apply its inductive learning methods to discover useful regularities in the data. In particular experiments with part of a large computer equipment database, provided by Digital Equipment Corporation, have resulted in Theo inferring thousands of rules which predict the value of one database attribute from other attributes of the same database entity. These rules were then used to identify database values that are *inconsistent* with these discovered regularities. In one experiment using a subset of the database containing descriptions of 89 different computer CPUs, these rules were able to identify a number of database errors which have since been corrected in the database in use at Digital. We are interested in continuing this line of research, especially in exploring the ''mining'' of engineering design databases. However, to date we have found it difficult to obtain very large design databases due to the proprietary concerns of companies that have them.

4.2.6 Follow On

Research supported by this grant is being followed up in several directions, primarily through two new ARPA grants in the areas of Machine Learning and Intelligent Information Assistants.

4.3 High-Performance Planning and Problem Solving with Prodigy

During the three-year contract period, we achieved significant breakthroughs in automated planning and machine learning and demonstrated these results in realistic domains (logistics, process planning). The Prodigy software architecture is today a powerful, general-purpose planning and learning system that, given basic axiomatic knowledge of a domain, can become an expert problem solver.

Our work was characterized by three major thrusts:

- Developing a robust planning and learning system.
- Completing several research directions that culminated in three PhD theses, in the areas of graphical knowledge acquisition, learning by experimentation, and learning by analogical/case-based reasoning.
- Initiating and pursuing other challenging research problems, such as learning plan quality, anytime planning, and learning by observation.

4.3.1 High-Performance Planning and Problem Solving

During the entire contract period, we designed and constructed a new Prodigy architecture, Prodigy4.0, which we completed, tested, and documented on schedule.

Taken together with the learning modules, Prodigy4.0 is a powerful planner that we offer to the ARPA community along with documentation and sample problem domains. We have tested Prodigy4.0 successfully in a number of domains, including factory process-planning and scheduling, transportation and logistics planning, autonomous reactive agent planning, and the usual suite of simpler AI planning tasks in the spirit of regression testing in software engineering.

The Prodigy4.0 substrate planner is a nonlinear planner with full interleaving of goal sets and individual plans into more efficient combined multigoal plans. Prodigy4.0 integrates several robust functionalities which we describe below.

Nonlinear Planning

Prodigy4.0 can generate a partial-order forest of subplans when multiple ways of planning for one problem are offered. This is particularly useful for symbiotic man/machine planning, where the human military strategist, for example, may have reasons to abandon one plan and must select another immediately — or where external considerations generate a preference for one line of planning over others.

The planner is a domain-independent, nonlinear planner with a rich action representation language. Each operator has a precondition expression that must be satisfied before the operator can be applied, and a list of effects that describe how applying the operator changes the world. Preconditions are expressed in a typed first order predicate logic encompassing negation, conjunction, disjunction, and existential and universal quantification. Variables in the operators may be constrained by arbitrary functions. The effects are atomic formulas that describe the conditions added or deleted from the current state when the operator is applied. Operators may also contain conditional effects, which represent changes to the world that are dependent on the state in which the operator is applied. A class (type) hierarchy organizes the objects of the world. These language constructs are important for representing complex and interesting domains.

The nonlinear planner follows a means-ends analysis, backward chaining, search procedure that reasons about multiple goals and multiple alternative operators relevant to the goals. This variety of operators amounts to multiple ways of trying to achieve the same goal. Therefore, in addition to searching in the space of multiple goal orderings, as most of the standard nonlinear planners do, the Prodigy4.0 planner searches equally in the space of multiple different approaches to solving a problem. Dynamic goal selection from the set of pending goals enables the planner to interleave plans thus exploiting common subgoals and addressing issues of resource contention. The planner returns a partially ordered plan as a result of analyzing dependencies among steps in the totally ordered solution searched [Veloso et al. 90, Blythe and Veloso 92].

- *Hierarchical Planning*— Planning at multiple levels of abstraction and aggregation was a capability offered earlier in Prodigy and now retained and integrated with the full nonlinear planner.
- *Intelligent-Commitment*— Previous Prodigy implementations used a ‘‘casual commitment’’ approach to decide which goal to work on next, which operator to apply, which subplan to select and expand, etc. This method fed the learning modules, which gradually accumulated knowledge to make the decisions more effective (less casual). However, for any new domain, this method requires a significant training period. Least-Commitment approaches (such as in SIPE) can be computationally inefficient and do not permit dynamic replanning, but neither do they require training. To get the best of both worlds, Prodigy4.0 incorporates general-purpose decision rules (based on plan-dependency analysis, interaction-analysis, and constraint unification) that generate relatively intelligent decisions without training. These are further enhanced during the training phase to produce even better decisions in a planner that permits dynamic replanning. The new method is called ‘‘intelligent commitment.’’
- *Constraint-based matcher*— In order to scale Prodigy up to large domains, we had to make it far more efficient. The largest component in efficiency enhancements was a much more powerful constraint-based matcher for rules and operators. Other efficiency improvements come directly from learning methods that use prior experience to reduce search for effective plans.

  Our constraint-based matcher is a first step towards reducing matching cost and integrates matching for forward control rules and unification of backward rules (operators). The new matcher exploits constraints from multiple sources, such as goals, type hierarchy, functions, static properties, and control rules, and combines them in the process of matching operators and control rules. We present theoretical analysis and empirical results and show that in general, the more complex the domain and the more relevant the control rules, the greater the advantage of constraint-based matching [Wang 92a].

Multiagent Cooperative Planning

The Prodigy architecture has been extended to generate parallel plans in multiagent domains. The approach that we followed is a centralized one in which a single planner generates a plan for each resource and monitors its execution, thereby guaranteeing conflict avoidance while synchronizing agent plans. In this context the term ‘‘resources’’ refers to agents, such as robots, trucks or airplanes in a logistics transportation domain, or machines in a process-planning domain. The goal is to generate a parallel plan that tries to minimize first the execution time and then the resources used in solving the problem. This work exploits the parallelism embedded in the partial order graph that *NoLimit*, the Prodigy nonlinear planner, generates from a single, totally ordered plan. Partial order constraints, conflicts among operators and resources, and synchronization necessary among operators requiring more than one agent all limit the parallelism that can be achieved. An execution monitor directs the parallel execution of the agents plans. We have demonstrated this approach in three different domains: transportation logistics, an extended version of the STRIPS robot domain, and a process planning domain [Perez 91].

Anytime Planning

Autonomous agents that respond intelligently in dynamic, complex environments need to display diverse capabilities that have yet to be combined in a single system. We explored the integration of a reactive planning system with a classical deliberative planner that has exploited some of the strengths of both systems. In order to plan in a resource-limited situation, or to replan when the original plans do not unfold as predicted, the capability known as *Anytime planning* is required. In essence Anytime planning provides for:

- *Time-bounded planning*— When a planner does not know how much time is available for planning, a rough plan is generated and gradually refined or replaced by a better set of plans. At any time, if the planner is interrupted, it can produce the best plan(s) generated so far. This ability is crucial in time-bounded replanning situations.

  We realized that an agent with the desired properties can be achieved by a reactive system using a slightly modified deliberative planner as a subroutine. We then modified our deliberative planning system to be an instance of an Anytime algorithm. The reactive system presents the planner with an abstract version of the problem, and the planner can then produce a more flexible plan. Since we do not make a fundamental change to the planner, we are able to take advantage of the body of work that has been done with classical planners, such as the use of abstraction, machine learning to improve planning performance and derivational analogy. Our preliminary results show that Prodigy communicates very successfully with a simulated household robot built in the Oz system.

- *Planning under uncertainty*— If subsequent circumstances invalidate part of a plan, the planner then focuses its efforts on recovering only that part of the plan, leaving the rest as intact as possible, except where resource contention or other interactions require changes to propagate.

Time-bounded planning required significant research on interrupting plans, on making tentative decisions that require re-examination and possible retraction, and on progressive-quality planning. The Prodigy anytime planning component was tested in the Oz simulated agent environment where unexpected events can cause plans to be modified or invalidated at execution time. The net effect was to produce a more effective (accurate) planning mechanism than the simple reactive planner that is part of the Oz system itself — yet one that can react just as quickly when required and reflects longer on improving plans when the situation permits.

Unexpectedly, planning under uncertainty proved to be a simple extension of the case-based derivational analogy mechanism, where replanning from similar past situations also requires localized plan recovery.

4.3.2 Acquisition of Domain Knowledge

Many planning tasks can be represented using mental models in which an expert manipulates objects from one state to another—delivery route planning (trucks, buildings, packages, routes, etc.) or part machining (parts, drill, mill, drill-bit, etc.). This suggests a highly graphical knowledge-acquisition tool, where the expert is able to capture the visual intuition of the problem solving to facilitate the encoding of a domain knowledge base. By exploring knowledge acquisition for object manipulation domains, we expect to gain insight in how knowledge is acquired and represented for such visually oriented tasks.

We developed a sophisticated, graphical knowledge-acquisition tool, the *Apprentice* module of the Prodigy architecture. A user interface initiates an apprentice-like dialogue, enabling the user to evaluate and guide the system's problem solving and learning. The interface is graphic-based and tied directly to the problem solver, so that it can both acquire domain knowledge or accept advice as it solves a problem.

The interface allows users to define domains graphically, better reflecting how the domain is thought about. The user draws graphical representations of domain objects and their relationships with one another. States are represented graphically by building multiple objects and relationships. Finally, operators are defined graphically by pre-state and post-state specifications. From these graphical images the corresponding Prodigy code is automatically generated. Note that nongraphical information can also be added to the state or operator definitions. The graphical interface is tied directly into the problem solver, so that during problem solving the solution is animated. This animation enables users to understand easily what the problem solver is doing and to detect and correct erroneous knowledge.

The user can also specify, in a coaching fashion, the correct solution for a problem. During specification the system verifies that the operators can be applied at the appropriate time. This checking helps users identify erroneous or incomplete operators.

The Apprentice system was designed as a knowledge acquisition tool to facilitate the construction of Prodigy domains. We evaluated Apprentice's performance, and have shown the system to be versatile, flexible, and productive [Joseph 92].

Our initial Apprentice evaluation used students in an advanced AI class and involved multiple users in numerous domains. Thirty-two students built their own predefined domains using the Apprentice system. The students were diverse, ranging from sophomores to first year graduate students and their major areas of study varied from computer science to music. The domains developed were as diverse as the students' background, ranging between an eight-tile puzzle to the construction of DNA molecules. Using Apprentice, students successfully developed domains one to four hours. The students even found the system fun and easy to use. Thus Prodigy becomes usable in little time by relative novices — the very first AI planner to have achieved such acceptance.

Our second study compared the development and understanding of domains in the Apprentice system against domains developed and presented textually. Subjects included Prodigy experts, Carnegie Mellon's AI workers, industrial AI employees and non-technical people. Two subjects from each group were used. Each person had to build one domain using Apprentice and one using Emacs. Results clearly show that all but the most seasoned Prodigy users develop domains faster using Apprentice. The subjects were then asked questions about different domains presented in textual form and in graphical Apprentice form. Subject responses revealed that users understand domains in Apprentice better than those presented textually.

A third evaluation was a learning study in which a subject developed several domains in Apprentice. Finally, a fourth Apprentice investigation demonstrated the ability to develop a larger domain in the system. Apprentice and its techniques proved to be usable, flexible, and extensible.

4.3.3 Implementation of a Large-Scale Domain

We have two main reasons for interest in this topic. First, we wished to demonstrate the feasibility of effective implementations of realistic, large-scale, complex domains in a general-purpose architecture. We also wanted to use these knowledge-rich domains to investigate issues related to learning domain knowledge for planning.

Much research is being done on how to automate manufacturing processes. The planning component in the production stage is significant, due to the variety of alternative processes, their complexity, and their interactions. We defined a specification of some manufacturing processes, including the machining, joining, and finishing of parts. This specification is not meant to be comprehensive nor detailed, but to present the AI community with a model of a complex and realistic application. Another goal is to use this model to demonstrate the feasibility of effective implementations of large-scale complex domains in a general-purpose architecture. This specification has been successfully demonstrated in the Prodigy architecture and is one of the largest domains available for general-purpose planners [Gil 91a].

We have used this domain in work on acquiring domain knowledge by experimenting with the environment. When the expectations suggested by the domain knowledge and the observations differ, there is need and opportunity for learning. Since there are usually several possible ways to correct the domain, the system must experiment to gather additional information. We have extracted general-purpose heuristics for guiding the experimentation process by static analysis of the existing domain knowledge. These heuristics are used to identify the most plausible hypotheses and avoid costly experiments.

Unlike explanation-based learning or learning by analogy, which learn to plan more *efficiently*, learning by experimentation addresses the issue of learning to plan more *correctly*. If domain knowledge is missing, as typically occurs in incomplete domain specifications provided by human experts prior to laborious iterative debugging, the experimentation process interacts with the external world to repair and expand existing knowledge gradually by deliberative experimentation, isolating and measuring each possible cause of planning failure. Recovery from 10%, 20%, and up to 50% incomplete domain knowledge was demonstrated, though recovery rates were much faster for the smaller situations.

In order for autonomous systems to interact with their environment in an intelligent way, they must be given the ability to adapt and learn incrementally and deliberately. It is virtually impossible to devise and hand-code all potentially relevant domain knowledge for complex dynamic tasks. We developed a framework to acquire domain knowledge for planning by failure-driven experimentation with the environment.

The initial domain knowledge in the system is an approximate model for planning in the environment, defining system expectations. The framework exploits the characteristics of planning domains in order to search the space of plausible hypotheses without the need for additional background knowledge to build causal explanations for expectation failures. Plans are executed while the external environment is monitored, and differences between the internal state and external observations are detected by various methods each correlated with a typical cause for the expectation failure.

The methods also construct a set of concrete hypotheses to repair the knowledge deficit. After being heuristically filtered, each hypothesis is tested in turn with an experiment. After the experiment is designed, a plan is constructed to achieve the situation required to carry out the experiment. The experiment plan must meet constraints such as minimizing plan length and negative interference with the main goals. We developed a set of domain-independent constraints for experiments and their incorporation in the planning search space. After the execution of the plan and the experiment, observations are collected to conclude if the experiment was successful or not. Upon success, the hypothesis is confirmed and the domain knowledge is adjusted. Upon failure, the experimentation process is iterated on the remaining hypotheses until success or until no more hypotheses are left to be considered.

This framework has shown to be an effective way to address incomplete planning knowledge and is demonstrated in a system called EXPO, implemented on the Prodigy planning architecture. The effectiveness and efficiency of EXPO's methods is empirically demonstrated in several domains, including a large-scale process planning task, where the planner can recover from situations missing up to 50% of domain knowledge through repeated experimentation [Carbonell and Gil 90, Gil 91b, Gil 92].

4.3.4 Learning by Analogical Reasoning

Prodigy's general problem solver (PS), which uses different domain theories, typically incurs extensive search costs. Case-based Reasoning (CBR), by contrast, relies on adapting previous solutions. However, CBR requires accurate similarity metrics and costly retrieval to guarantee successful adaptation. In large, complex domains the costs of searching become crucial, but so do those of organizing memory and retrieving appropriate experience. We explored ways to optimize the tradeoff between PS and CBR, which amounts to optimizing the tradeoff between search and memory. Prodigy/Analogy is a complete integration of both problem-solving paradigms within the Prodigy architecture [Veloso 92a, Veloso and Carbonell 93a].

We demonstrated that planning in complex domains can be largely improved by adding to the planner the ability to learn from simple problems and to project the planner's experience by analogy into more complex problems. The learning approach we pursued involves (1) training the planner on simple problems in the domain and accumulating exemplary solutions from the derivational trace of the episodic planning experience; (2) organizing a case library to allow the efficient retrieval of the learned plans, and (3) reusing multiple stored plans found conjunctively similar to a new planning situation.

Case generation

Automatic case generation occurs by extending the general planner with the ability to introspect into its internal decision cycle, recording the justifications for each decision during its extensive search process. Examples of these justifications are links between choices capturing the subgoaling structure, records of explored failed alternatives, and pointers to applied control guidance. A stored planning episode consists of the successful solution trace augmented with these annotations, i.e. the derivational trace [Veloso 91].

Essentially, to generate cases as planning episodes automatically, we:

- Identify the decision points in the search procedure where guidance may prove useful to provide memory of the justifications for the choices made.
- Use a clear language to capture these justifications at planning time and associate a meaning so that they can be used at replay time.

Storage and Retrieval of Plans

We extract the indices for a case from the initial state and the goal statement that define a planning problem. From exploring the search space and following the subgoaling links in the derivational trace of the plan generated, the system identifies the set of initial state literals that contributed to achieving each goal. The connected components of the partially ordered plan determine the independent fragments of the case, each corresponding to a set of interacting goals. Each case is multiply indexed by the sets of interacting goals and the relevant initial state literals.

When a new problem is presented to the system, the retrieval procedure matches the new initial state and goal statement against the indices of the cases in the case library.

The organization of the case library is done using appropriate data structures to store the set of indices such that the set of candidate analogs at retrieval time can be pruned as efficiently as possible. We use two levels of indexing — a hash table and a discrimination network — to store the features in the goal statement and in the initial state shared among the cases. There are many problem solving situations for which the parameterized goal statements are identical and the initial states are different. These different initial states are organized into a discrimination network to index efficiently these cases that completely share the goal statement, but differ in the relevant initial state [Veloso and Carbonell 91a, Veloso and Carbonell 93b, Doorenbos and Veloso 93b].

Replay of Multiple Similar Plans

Prodigy/Analogy core functionality consists of a sophisticated replay mechanism that is able to reconstruct solutions from the retrieved past cases when only a partial match exits between the new and past situations. The replay mechanism coordinates the set of multiple retrieved cases and uses the annotated justifications to guide the reconstruction of the solution for problem solving situations where equivalent justifications hold true.

The replay algorithm is implemented as an extension to the base-level planner. It consists mostly of interrupting the planner at its decision points so it may make choices similar to the ones taken in the past guiding plans. The replay functionality transforms the planner into a module that tests the validity of the choices proposed by the past experience and follows equivalent search directions. Without the replay functionality the planner generates, at high cost, possible goal orderings and operators to achieve the goals and searches through the space of alternatives generated [Veloso 92b, Veloso 93].

In summary, the replay procedure provides the following benefits to the planning procedure:

- Proposal and validation of choices versus generation and search of possible alternatives operators and goal orderings.
- Reduction of the number of plausible alternatives — past failed alternatives are pruned up front by validating the failures recorded in the past cases.

Empirical Evaluation and Results

In Artificial Intelligence it is particularly challenging and important to validate the algorithmic solutions designed, as most proposed approaches look equally plausible. Although there are situations where analytical validation may be possible, empirical experimentation is generally the only appropriate method to demonstrate the power of a designed hypothesis.

To support our hypothesis that learning by analogical reasoning would greatly improve the performance of a planning system in complex domains, we performed extensive empirical tests demonstrating our algorithms [Veloso and Carbonell 91b, Veloso and Carbonell 93c]. We ran a set of 1000 problems in a logistics transportation domain, in which packages are to be moved among different cities. Packages are carried within the same city in trucks and between cities in airplanes. At each city there are several locations, e.g., post offices and airports. The problems have up to 20 conjuncts in the goal statement and more than 100 literals in the initial state. The case library is accumulated incrementally as the system solves problems with an increasing number of goal conjuncts. There are more than 1000 cases of up to 240 steps each, stored in the case library. This system indicates the scope and complexity of planning in the ARPA logistics/transportation planning domain, to be addressed hands-on as an application area of Prodigy in the subsequent contract period.

The analogical reasoner increased the solvability horizon of the base-level planner considerably. An unguided exploration of the search space drives the problem solver easily into a chain of inconvenient or wrong decisions from which it is hard to recover, since there is a large number of open planning alternatives to search from. Therefore although all the problems are solvable theoretically, in practice they become rapidly unsolvable within a bounded running time when their complexity increases. Prodigy/Analogy solves the complete set of 1000 problems with a running time limit of up to 350 seconds. Without analogy the base-level nonlinear planner solves only 458 problems out of the 1000 problems, even when the search time limit is increased up to 350 seconds.

Similarly, the cumulative running time for the set of 1000 problems is significantly improved when using analogical reasoning. The 1000 problems solved by analogy correspond to a total of approximately 40,000 seconds, while the total running time effort of the base level problem solver corresponds to approximately 211,000 seconds. This represents a speedup factor of approximately 5.3 and also means that the cumulative savings in running time for analogy is approximately 81%. The maximum individual speedup for problems solved in both configurations is approximately 40×.

4.3.5 Other Active Research Directions

Learning Plan Quality

We developed an initial model of decision-rule formalism for learning how to plan more accurately (as opposed to simply learning to plan faster).

Most of the work to date on automated control-knowledge acquisition has been aimed at improving the *efficiency of planning* — ‘‘speedup learning.’’ We focus in this work on the semiautomated acquisition of control knowledge to guide a planner towards better solutions (i.e. to improve the *quality of plans* produced by the planner) as its problem solving experience increases. To date, no work has focused on automatically acquiring knowledge to improve plan quality in planning systems. We built a first prototype that partially automates the task of acquiring quality-enhancing control knowledge for Prodigy [Perez and Veloso 93].

Learning by Observation

We completed a study determining that learning operators from state-sequence observations is a viable method of automating domain knowledge acquisition.

We have been investigating techniques to learn action models simply by observing other agents performing actions. Autonomous agents should have the ability to learn from the environment. The actions of other agents in the environment provide a useful source for such learning. We observe sequences of changes in the environment caused by other agents, and hypothesize operators of the domain based on the observations. While the system is given new problems to solve, it uses, modifies, and confirms the hypothesized operators during problem solving [Wang 92b].

4.4 Bibliography

[Altmann 93a] Altmann, E.M.
Learning Scope, Task Analysis, and Sharable Components.
In *Proceedings of the Second International Workshop on Multistrategy Learning*. 1993.

[Altmann 93b] Altmann, E M.
Learning scope at the human scale.
1993
In Second International Workshop on Multistrategy Learning (1993).

[Altmann and Yost 92]
Altmann, E., and Yost, G.R.
Expert-system development in Soar: a tutorial.
Technical Report CMU-CS-92-151, Carnegie Mellon University, School of Computer Science and Digital Equipment Corporation, June, 1992.

[Altmann et al. 90] Altmann, E., A. Newell, and G.R. Yost.
Mapping domains to computational models: Shared concerns of problem spaces and object-oriented systems (extended abstract).
Presented at the *Finding the Object* workshop, Object Oriented Programming Systems, Languages, and Applications conference, Ottawa.
October, 1990

[Blythe and Reilly 93]
Blythe, J. and Reilly, W.S.
Integrating reactive and deliberative planning for agents.
Technical Report, School of Computer Science, Carnegie Mellon University, Forthcoming in 1993.
A shorter version is submitted to AAAI-93.

[Blythe and Veloso 92]
Blythe, J., and Veloso, M.M.
An Analysis of Search Techniques for a Totally-Ordered Nonlinear Planner.
In *Proceedings of the First International Conference on AI Planning Systems*. College Park, MD, June, 1992.

[Bocionek and Mitchell 93]
Bocionek, S., and Mitchell, T.M.
Office Automation Systems that are Programmed by their Users.
In *Proceedings of the 23rd Annual Conference of the German Association of Computer Science*. September, 1993.

[Bocionek and Sassin 93]
Bocionek, S., and Sassin, M.
Dialog-Based Learning for Adaptive Interface Agents and Programming-by-Demonstration Systems.
Technical Report CMU-CS-93-175, School of Computer Science, Carnegie Mellon University, 1993.

[Borrajo and Veloso 93]
Borrajo, D., and Veloso, M.M.
Bounded Explanation and Inductive Refinement for Acquiring Control Knowledge.
In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, pages 21-27. Amherst, MA, June, 1993.

[Carbonell and Gil 90]
Carbonell, J.G., and Gil, Y.
Learning by Experimentation: The Operator Refinement Method.
In R. S. Michalski and Y. Kodratoff (editors), *Machine Learning: An Artificial Intelligence Approach, Volume III*, pages 191-213. Morgan Kaufmann, Palo Alto, CA, 1990.

[Carbonell et al. 92]
Carbonell, J.G. and the Prodigy Research Group.
PRODIGY4.0: The Manual and Tutorial.
Technical Report CMU-CS-92-150, cmu-scs, June, 1992.

[Cho et al. 91] Cho, B., P.S. Rosenbloom, and C.P. Dolan.
Neuro-Soar: A Neural-Network Architecture for Goal-Oriented Behavior.
In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*. CSS, August, 1991.

[Dent et al. 92] Dent, L., Boticario, J., McDermott, J., Mitchell, T., and Zabowski, D.
A personal learning apprentice.
In *1992 National Conference on AI*. July, 1992.

[Doorenbos 93] Doorenbos, R.
Matching 100,000 learned rules.
In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 290-296. 1993.

[Doorenbos and Veloso 93a]
Doorenbos, R.B., and Veloso, M.M.
Knowledge Organization and the Utility Problem.
In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, pages 28-34. 1993.

[Doorenbos and Veloso 93b]
Doorenbos, R.B., and Veloso, M.M.
Knowledge Organization and the Utility Problem.
In *Proceedings of the Third International Workshop on Knowledge Compilation and Speedup Learning*, pages 28-34. Amherst, MA, June, 1993.

[Doorenbos et al. 92]
Doorenbos, R., M. Tambe, and A. Newell.
Learning 10,000 chunks: What's it like out there?
In *Proceedings, Tenth National Conference on Artificial Intelligence*. 1992.

[Gil 91a] Gil, Y.
A Specification of Process Planning for Prodigy.
Technical Report CMU-CS-91-179, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, August, 1991.

[Gil 91b] Gil, Y.
A Domain-Independent Framework for Effective Experimentation in Planning.
In *Proceedings of the Eight International Workshop on Machine Learning*. 1991.

[Gil 92] Gil, Y.
Acquiring Domain Knowledge for Planning by Experimentation.
PhD thesis, Carnegie Mellon University, School of Computer Science, August, 1992.
Available as Technical Report CMU-CS-92-175.

[Haigh and Veloso 93]
Haigh, K., and Veloso, M.M.
Combining Search and Analogical Reasoning in Path Planning from Road Maps.
In *Preprints of the AAAI Workshop on Case-based Reasoning*, pages 79-85. AAAI Press, Menlo Park, CA, Washington, DC, July, 1993.

[John et al. 90] John, B.J., Vera, Alonso and A. Newell.
Toward Real-Time GOMS.
Technical Report CMU-CS-90-195, School of Computer Science, Carnegie Mellon University, December, 1990.

[John et al. 91] John, B.J., R.W. Remington, and D.M. Steier.
An Analysis of Space Shuttle Countdown Activities:Preliminaries to a Computational Model of the NASA Test Director.
Technical Report CMU-CS-91-138, School of Computer Science, Carnegie Mellon University, May, 1991.

[Joseph 92] Joseph, R.L.
Graphical knowledge acquisition for visually-oriented planning Domains.
PhD thesis, School of Computer Science, Carnegie Mellon University, August, 1992.
Available as technical report CMU-CS-92-188.

[Laird et al. 90] Laird, J.E., C.B. Congdon, C.B. Altmann, and K. Swedlow.
Soar User's Manual: Version 5.2.
Technical Report CMU-CS-90-179, School of Computer Science, Carnegie Mellon University, October, 1990.

[Lehman et al. 91a]
Lehman, J.F., R.L. Lewis, and A. Newell.
Natural Language Comprehension in Soar: Spring 1991.
Technical Report CMU-CS-91-117, School of Computer Science, Carnegie Mellon University, March, 1991.

[Lehman et al. 91b]
Lehman, J.F., R. Lewis, and A. Newell.
Integrating Knowledge Sources in Language Comprehension.
In *Proceedings of the Thirteenth Annual Conferences of the Cognitive Science Society*. Cognitive Science Society, 1991.

[Lehman et al. 92a]
Lehman, J.F., A. Newell, T. Polk, and R. Lewis.
The Role of Language in Cognition: A Computational Inquiry.
In Harman, G. (editor), *Conceptions of the Human Mind*. Lawrence Erlbaum Associates, Inc, 1992.

[Lehman et al. 92b]
Lehman, J.F., R. Lewis, and A. Newell.
NL-Soar: Architectural Influence on Language Comprehension.
In Pylyshyn, Z. (editor), *Cognitive Architecture*. Ablex Press, 1992.

[Lewis 92] Lewis, R.L.
Recent developments in the NL-Soar garden path theory.
Technical Report CMU-CS-92-141, School of Computer Science, Carnegie Mellon University, 1992.

[Lewis 93] Lewis, R.L.
An Architecturally-Based Theory of Human Sentence Comprehension.
In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*. 1993.

[Milnes et al. 92] Milnes, B.G., Pelton, G., Doorenbos, R., Hucka, M., Laird, J., Rosenbloom, P., and Newell, A.
A specification of the Soar cognitive architecture in Z.
Technical Report CMU-CS-92-169, School of Computer Science, Carnegie Mellon University, August, 1992.

[Mitchell 90] Mitchell, T.M.
Becoming Increasingly Reactive.
In *Proceedings of AAAI '90*. AAAI, 1990.

[Mitchell and Thrun 93a]
Mitchell, T.M., and Thrun, S.B.
Explanation-Based Learning: A Comparison of Symbolic and Neural Network Approaches.
In *Proceedings of the Tenth International Conference on Machine Learning*. June, 1993.

[Mitchell and Thrun 93b]
Mitchell, T.M., and Thrun, S.B.
Explanation-based neural network learning for robot control.
In *Advances in Neural Information Processing Systems 5*. Morgan-Kaufmann Press, 1993.

[Newell 90] Newell, A.
Unified Theories of Cognition.
Harvard University Press, Cambridge, Massachusetts, 1990.

[Newell 92a] Newell, A.
Precis of *Unified Theories of Cognition*.
Behavioral and Brain Sciences 15(3):425-492, 1992.

[Newell 92b] Newell, A.
Unified Theories of Cognition and the Role of Soar.
Soar: A Cognitive Architecture in Perspective.
In Michon, J.A., and Akyurek, A.,
Kluwer, 1992.

[Newell 93] Newell, A.
Reflections on the Knowledge Level.
Artificial Intelligence 59:31-38, 1993.

[Newell and Steier 91]
Newell, A., and D.M. Steier.
Intelligent Control of External Software Systems.
Technical Report EDRC05-55-91, School of Computer Science, Carnegie Mellon University, April, 1991.

[Newell et al. 91] Newell, A., G.R. Yost, J.E. Laird, P.S. Rosenbloom, and C.B. Altmann.
Formulating the problem space computational model.
Carnegie Mellon Computer Science: A 25-Year Commemorative.
In Rashid,
ACM-Press:Addison-Wesley, 1991.

[Perez 91] Perez, M. A.
Multiagent Planning in Prodigy.
Technical Report CMU-CS-91-139, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[Perez 92] Perez, M.A.
Learning from expert knowledge to improve the quality of plans.
Thesis proposal, School of Computer Science, Carnegie Mellon University, 1992.

[Perez and Etzioni 92]
Perez, M.A., Etzioni, O.
DYNAMIC: A New Role for Training Problems in EBL.
In D. Sleeman and P. Edwards (editor), *Machine Learning: Proceedings of the Ninth International Conference (ML92)*. Morgan Kaufmann, San Mateo, CA., 1992.
Also available in the Working Notes of the 1992 Spring Symposium Series, Symposium on Knowledge Assimilation. Extended version available as Technical Report CMU-CS-92-124.

[Perez and Veloso 93]
Perez, M.A., and Veloso, M.M.
Goal Interactions and Plan Quality.
In *Preprints of the AAAI 1993 Spring Symposium Series, Workshop on Foundations of Automatic Planning: The Classical Approach and Beyond*, pages 117-121. Stanford University, CA, March, 1993.

[Rosenbloom and Newell 91]
Rosenbloom, P.S., and A. Newell.
Symbolic Architectures: Organization of Intelligence I.
In *Exploring Brain Functions: Models in Neuroscience Workshop*. , 1991.

[Rosenbloom et al. 91a]
Rosenbloom, P.S., A. Newell, and J.E. Laird.
Toward the knowledge level in Soar: The role of the architecture in the use of knowledge.
Architectures for Intelligence.
In VanLehn,
Lawrence Erlbaum Associates, 1991.

[Rosenbloom et al. 91b]
Rosenbloom, P.S., J.E. Laird, and R. McCarl.
A preliminary analysis of the Soar architecture as a basis for general intelligence.
Artificial Intelligence 47(1-3):289-325, 1991.

[Simon et al. 91] Simon, T., A. Newell, and D. Klahr.
Q-Soar: A computational account of children's learning about number conservation.
Working Models of Human Perception.
In Fisher and Pazzani,
Morgan Kaufmann, 1991.

[Steier 90] Steier, D.M.
Intelligent architecture for integration.
In *Proceedings of the IEEE Conference on Systems Integration*. , August, 1990.

[Steier et al. 92] Steier, D.M., R.L. Lewis, J. Fain, and A. Zacherl.
Combining multiple sources of knowledge in an integrated intelligent system.
IEEE Expert , 1992.

[Tambe 91a] Tambe, M.
Eliminating Combinatorics from Production Buf Match.
PhD thesis, School of Computer Science, Carnegie Mellon University, May, 1991.

[Tambe 91b] Tambe, M., D. Kalp, and R. Rosenbloom.
Uni-Rete: Specializing the Rete Match Algorithm for the Unique Attribute Representation.
Technical Report CMU-CS-91-180, School of Computer Science, Carnegie Mellon University, 1991.

[Tambe and Rosenbloom 90]
Tambe, M., and P.S. Rosenbloom.
A Framework for investigating production system formulations with polynominally bounded match.
In *Proceedings of the National Conference on Artificial Intelligence*. NCAI, August, 1990.

[Tambe et al. 90] Tambe, M., A. Newell and P.S. Rosenbloom.
The problem of expensive chunks and its solution by restricting expressiveness.
Machine Learning (5):299-348, 1990.

[Tambe et al. 92] Tambe, M., B. Doorenbos, and A. Newell.
The match cost of adding a new rule: A clash of views.
Technical Report CMU-CS-92-158, School of Computer Science, Carnegie Mellon University, June, 1992.

[Thrun and Mitchell 93]
Thrun, S.B., and Mitchell, T.M.
Integrating Inductive Neural Network Learning and Explanation-Based Learning.
In *Proceedings of the 1993 International Joint Conference on Artificial Intelligence*. August, 1993.

[Veloso 91] Veloso, M.M.
Efficient Nonlinear Planning using Casual Commitment and Analogical Reasoning.
In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 938-943. August, 1991.

[Veloso 92a] Veloso, M.M.
Learning by analogical reasoning in general problem solving.
PhD thesis, School of Computer Science, Carnegie Mellon University, August, 1992.
Available as Technical Report CMU-CS-92-174.

[Veloso 92b] Veloso, M.M.
Automatic Storage, Retrieval, and Replay of Multiple Cases.
In *Preprints of the AAAI 1992 Spring Symposium Series, Workshop on Computational Considerations in Supporting Incremental Modification and Reuse*. Stanford University, CA, March, 1992.

[Veloso 93] Veloso, M.M.
Planning for Complex Tasks: Replay and Merging of Multiple Simple Plans.
In *Preprints of the AAAI 1993 Spring Symposium Series, Workshop on Foundations of Automatic Planning: The Classical Approach and Beyond*, pages 146-150. Stanford University, CA, March, 1993.

[Veloso and Carbonell 91a]
Veloso, M.M., and Carbonell, J.G.
Variable-Precision Case Retrieval in Analogical Problem Solving.
In *Proceedings of the 1991 DARPA Workshop on Case-Based Reasoning*, pages 93-106. Morgan Kaufmann, Washington, DC, May, 1991.

[Veloso and Carbonell 91b]
Veloso, M.M. and J.G. Carbonell.
Learning by analogical replay in Prodigy: First results.
In *Proceedings of the European Working Session on Learning*. Springer-Verlag, March, 1991.

[Veloso and Carbonell 93a]
Veloso, M. and J.G. Carbonell.
Derivational analogy in Prodigy: Automating case acquisition, storage, and utilization.
Machine Learning 10, 1993.

[Veloso and Carbonell 93b]
Veloso, M.M., and Carbonell, J.G.
Automatic Case Generation, Storage, and Retrieval in Prodigy.
In R.S. Michalski (editor), *Machine Learning: A Multistrategy Approach, Volume IV*. Morgan Kaufmann, 1993.
(A reduced version of this paper is also available in the Proceedings of the First International Workshop on Multistrategy Learning, November 1991).

[Veloso and Carbonell 93c]
Veloso, M.M., and Carbonell, J.G.
Towards Scaling Up Machine Learning: A Case Study with Derivational Analogy in Prodigy.
In S. Minton (editor), *Machine Learning Methods for Planning and Scheduling*, pages 233-272. Morgan Kaufmann, 1993.

[Veloso et al. 90] Veloso, M.M., M.A. Perez, and J.G. Carbonell.
Nonlinear planning with parallel resource allocation.
In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*. November, 1990.

[Vera et al. 93] Vera, A,H., Lewis, R.L., and Lerch, F.J. .
Situated Decision-Making and Recognition-Based Learning: Applying Symbolic Theories to Interactive Tasks.
In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*. 1993.

[Wang 92a] Xuemei Wang.
Constraint-Based Efficient Matching in Prodigy.
Technical Report CMU-CS-92-128, School of Computer Science, April, 1992.

[Wang 92b] Wang, X.
Learning action models by observing other agents.
Technical Report, School of Computer Science, Carnegie Mellon University, 1992.

[Ward 91] Ward, B.
ET-Soar: Toward an ITS for Theory-Based Representations.
PhD thesis, School of Computer Science, Carnegie Mellon University, May, 1991.

[Yost 92] Yost, G.R.
TAQL: A Problem Space Tool for Expert System Development.
PhD thesis, School of Computer Science, Carnegie Mellon University, May, 1992.

5. RESEARCH IN CREATING GRAPHICAL APPLICATIONS

Carnegie Mellon's research in this area is embodied in the Garnet project and has built an interface development environment in which most aspects of graphical, highly interactive interfaces can be specified without programming. Instead, user interfaces are specified using direct-manipulation graphical techniques and a very-high-level specification language, enabling the achievement of rapid prototyping of user interfaces.

Systems such as Garnet have demonstrated that the time it takes to develop the graphics and basic interaction objects can be reduced from months to days. However, it can still take months to develop the other parts of the user interface.

During this contract period, we worked on:

- Determining how to specify application-specific behaviors by demonstration, and implementing these techniques in graphical user interface editors.
- Designing and developing a constraint language that supports the relationships needed by all user interface objects. These constraints are then solved in real time.
- Developing methods to provide existing gesture recognition technology in an application-independent manner.

In addition, we have documented all parts of the developed system to maximize their utility to other projects. We have offered continuous interface to and support for other projects in their usage of the system, to ensure Garnet's acceptance.

5.1 The Need for Tools

Creating user interface software is a difficult and expensive task, and often leads to approximately 50% of the program code [Myers and Rosson 92]. Unfortunately, as user interfaces become easier to use for the end user, they usually become more complex and harder for the interface designer to create. The easy-to-use direct graphic manipulation interfaces are among the most difficult to implement. Our research is explicitly aimed at easing the creation of direct manipulation interfaces.

5.2 Look-and-Feel Independence

Few existing user interface tools allow the designer to experiment with the ‘‘look-and-feel’’ of the interface; that is, different graphical appearances and different responses to the input devices. Most modern systems come with a predefined look-and-feel, usually embodied in a library, or ‘‘toolkit’’ of widgets (user interface elements such as menus, buttons, and scroll bars).

If the designer is interested in creating a new look-and-feel, however, there are few tools to help. In addition, the toolkits themselves are often expensive to create. Our research in creating graphical applications is specifically designed to make it easy to create new, customized looks-and-feels. For example, we developed two complete widget sets: one with the Motif look-and-feel and one with a custom look-and-feel. The relative ease with which we accomplished this demonstrates we can quickly generate alternate appearances and behaviors.

5.3 Emphasis on Appearance and Behavior

Many existing tools for user interface construction require the designer to write programs in conventional or special-purpose programming languages. Many of these languages emphasize the syntax, or sequencing, of events, actions and modes in the user interface. However, a central part of the design for graphical user interfaces is the appearance and behavior of the components, which these languages cannot easily specify. Also, experience with special languages for defining user interfaces shows that designers are reluctant to learn and use them. Therefore, Garnet emphasizes interactive graphical tools where the designer can *draw pictures* of the objects and *demonstrate* the behaviors of those objects.

5.4 Gesture Recognition

Gesture recognition is the drawing of a shape using an input device such as a mouse, stylus, or finger, and its subsequent recognition as data or a command. An example is circling a group of objects in a graphics editor, drawing a "C" to choose a copy operation, and then moving the cursor to the desired destination. This action can all be performed in one quick, continuous movement.

Despite extensive research on gesture recognition, the technology is just now beginning to be practical. Existing systems, however, have been laboriously constructed in a particular application without high-level tools. No other system provides gesture recognition capabilities as a toolkit item that can be used by different applications and easily combined with conventional direct manipulation techniques.

5.5 Garnet Toolkit

The Garnet system [Myers et al. 90] can be divided into two layers: the low-level Garnet toolkit and the high-level interactive tools. We have tried to innovate at every level because we are investigating novel ways to construct all parts of the user interface software. Garnet is implemented in Lisp and uses the X window system.

The toolkit level provides a prototype-instance object system, constraints, new input and output models, widgets, debugging tools, and other utilities and demonstration programs.

5.5.1 The Garnet Object System

The Garnet object system, KR, supports a prototype instance model, rather than the conventional class-instance model used by Smalltalk and **C**$^{++}$. In a prototype instance model, there is no concept of a ‘‘class,’’ and any object can serve as a prototype for other objects. When a prototype is changed, all instances are automatically updated by the system.

The primary innovation in the object system is the support for ‘‘structural inheritance,’’ which means that if the prototype is an ‘‘aggregate’’ (or group) of objects, adding or removing objects from the prototype is reflected in all instances.

Another innovation is the demonstration that a prototype-instance system can be implemented efficiently. Through various versions, we decreased the size of the binaries by 40% and increased the runtime speed tenfold.

5.5.2 Constraints

Another important aspect of the Garnet object model is that properties of objects can be computed by ‘‘constraints,’’ which means that the value is updated automatically whenever anything it depends on changes [Vander Zanden and Myers 90]. Constraints are commonly used to tie graphical objects together. For instance, the programmer can declare that lines must stay attached to circles and Garnet maintains this as the circles are moved by the program or user. Constraints in Garnet can be arbitrary Lisp expressions so they can also be used for any application computation and to connect the graphics to applications.

An important feature in the Garnet constraint system is the use of indirect references [Vander Zanden et al. 91], which enable a constraint to refer to objects through variables. For example, a feedback object might have constraints that say ‘‘I am the same size as whatever I'm over,’’ and then at runtime, the program will set the referent object.

Another characteristic is automatic elimination of unneeded constraints. The constraint mechanism in Garnet is ubiquitously used to connect objects together, and a typical Garnet application will have thousands of constraints in it. However, we have discovered that a significant portion, often up to 95%, of these constraints are not needed at runtime. Therefore, we have devised a mechanism that removes these unnecessary constraints, thereby saving significant space and time.

We are collaborating with colleagues at the University of Washington on a multiway constraint solver. The current Garnet constraint system is ‘‘one-way,’’ which means that object B can depend on object C, so if C changes, B will also, but then if B changes, C will not. In a multiway constraint system, there can be multiple constraints attached to each property of the object, and the system automatically determines which one to use. Multiway constraints are clearly more powerful than one-way constraints, but research is required on how to integrate them with some other innovations in Garnet, such as inheritance of constraints and indirect constraints using pointer variables.

5.5.3 Output Model

Garnet's output system uses a ''retained object model,'' sometimes called a ''structured graphics model.'' This means that there is an object in memory for every graphical object on the screen. When the programmer or user changes a property of the object, Garnet automatically redraws the object and any other objects that are covering it. This ability makes it much more convenient for programmers since the system automatically determines which objects to redraw. Also, Garnet takes care of refreshing windows when they are scrolled, become uncovered, or change size.

Garnet is mainly designed to support graphical interfaces, but most users still need some text editing. Therefore, we also provide a sophisticated WYSIWYG multifont text editing capability, including support for filling lines and wrapping words in real time as the user types. The multifont text object also allows the embedding of arbitrary graphical objects in the string. These objects can even be buttons and other widgets, and can therefore support Hypertext systems. Furthermore, there is a Lisp editing mode that automatically indents and matches parentheses.

Garnet's built-in layout routines can be used when output objects should be laid out in columns, rows, tables, trees or connected graphs. Many kinds of applications have data that can be naturally shown this way, and with Garnet these applications do not have to implement their own layout code. The primary innovation here is that the designer has complete control over the way the nodes and arcs look. Arbitrary graphics can be defined for them, and the system will use the pictures in the appropriate places.

An important feature in Garnet is the demonstration that the retained object model can be used ubiquitously throughout an application. In most other systems that provide this model, the programmer is free to go around the object model and draw directly on the screen. This ability to bypass the object model negates the advantages cited above.

To make this work for maps and visualizations with thousands of objects, we added a special feature to eliminate most of the per-object overhead when there are lots of similar objects. We have developed ''virtual aggregates'' that pretend to allocate objects to the rest of the system but actually only maintain a small amount of state information.

Another technique we investigated would allow the programmer to still create composites by listing the components, but the system would combine the draw methods for all the objects into one draw method, and then allocate a single composite object for the entire collection. Preliminary results suggest that this technique might reduce drawing time and space allocation significantly.

5.5.4 Input Model

Garnet uses a novel input model where common direct manipulation behaviors are supplied as built-in ‘‘Interactor’’ objects that can be attached to graphics [Myers 90a]. The key importance of this model is that the interactors are parameterized so that all of the standard direct manipulation behaviors can be easily achieved using only a few basic interactor types. Each type handles a standard kind of behavior, such as selecting one or more of a set of objects, moving objects with the mouse, rotating objects, or text editing. The interactors are independent of the particular graphics used, so, for example, the same choice interactor can be used for buttons, palettes, menus, or even to select objects in a graphics editor. Parameters to the interactors include which mouse buttons cause them to start and stop, what the feedback looks like, and behavior-specific properties like how many objects can be selected or the minimum sizes for growing.

In keeping with our commitment to support state-of-the-art user interfaces, we have added the ability to recognize ‘‘gestures.’’ A gesture places importance on the *path* of the mouse, not just its start and end positions. For example, the user might draw an "X" over an object to delete it. Gestures have been well-recognized as an effective and interesting user interface technique, but they are usually difficult for designers to implement. Garnet provides an application-independent mechanism for using gestures in user interfaces. Building on the gesture recognition algorithms being developed at Carnegie Mellon in other research, Garnet contains high-level tools which report the interpretation of the gestures to the applications. These tools integrate gesture recognition with other, more conventional user interface elements, such as menus. Also, since we were able to add gestures to the existing Garnet input model, we have demonstrated that the input model can be extended to support innovative interactive techniques. In order to make the gesture recognition system fit in with the Garnet philosophy, we developed the Agate tool for specifying gestures by example [Landay and Myers 93].

The input model of Garnet was also extended to support animations. A special ‘‘animator-interactor’’ object can be attached to a graphical object and cause any property of the object (most commonly its position) to change at a fixed rate. This functionality is implemented using multiple processes. The same mechanism is also used to allow the arrows on scroll bars and sliders to repeatedly scroll at a fixed rate if the mouse button is held down on them.

5.5.5 Widgets

Since its beginning Garnet has contained a set of widgets with a custom look-and-feel. However, it became clear that Motif was an emerging standard as the way that user interfaces should look. Motif is also more attractive than the Garnet look-and-feel on color screens, which are becoming more common. Therefore, we developed a complete set of widgets that look like Motif’s, implemented on top of the Garnet low-level facilities described above. This makes it easy for developers to create custom widgets, and to demonstrate how easy the creation of new widgets is.

We are continually adding new widgets to the widget set. For example, we added

- Pulldown menus like those on the Macintosh,
- Pop-up option buttons that show the current value and display a menu showing all the values when the button is selected,
- Windows to support reading and saving to a file (which includes a scrolling menu of files in the current directory),
- Sophisticated property sheets that automatically determine which properties of objects to show and which widgets to use for them,
- A ‘‘mouseline’’ widget that provides documentation on the object indicated by the cursor, as positioned by the mouse. The widget displays information either as a single line at a fixed place or as pop-up windows like Macintosh ‘‘Balloon Help.’’

All widgets in the toolkit, as well as the underlying mechanism they are based on, support the inclusion of arbitrary Garnet objects as labels, instead of just text strings. Therefore, menus and buttons can contain pictures in place of textual labels, demonstrating the flexibility of the Garnet object system.

An important feature of the Garnet widget set is the support for the *insides* of application windows. Therefore, there is a ‘‘selection-handles’’ widget that can be used by any graphical application to support selection and manipulating of objects. In all other toolkits, programmers have to reimplement this functionality for each application. The handles in Garnet also support moving and growing of *multiple* objects at the same time.

5.5.6 Debugging

Garnet comes with a complete set of debugging tools. To make errors easier to catch, we added full support for type-checking of the slots of objects. This support makes Garnet code more testable and also provides information that can be used at run time to inspect objects using appropriate widgets. For example, a check box might be used for Booleans, a slider for numbers in a range, etc.

The ‘‘Inspector’’ is a powerful debugging tool that views and edits the properties of objects. It will also trace or trap into the debugger when any slot of an object is modified, either because it was directly set or because the value was computed by a constraint which changed. Other views in the Inspector show the object hierarchies and the dependencies of constraints.

5.5.7 Utilities

Originally, the only way to capture Garnet windows on paper was to use a ‘‘screen dump’’ utility, which simply saves the screen bitmap to a file. The problem with this method is that the picture looks grainy and has ‘‘jaggies’’ because the resolution of the screen (72 DPI) is typically much lower than the printer’s (300 DPI). Therefore, we developed a subsystem that generates a Postscript description of the objects on the screen. This approach means that the picture appears at the printer’s maximum possible resolution. An important implication of this feature is that applications that need a ‘‘print’’ command can simply use the built-in Garnet routine instead of writing their own code. The printing system supports printing of multiple windows, including the window manager’s headers of the windows.

We also provide functions to support reading and writing bitmaps and color pixmap pictures to files in the standard formats. This functionality allows interchange of pictures with other systems.

We provide a complete drawing tool, called Garnetdraw, that allows drawings to be created, saved to a file, read back in, and printed. This tool contains most of the functionality of MacDraw on the Macintosh, yet it is only 1700 lines long. That there is so little code demonstrates the high level of support provided by the Garnet toolkit for graphical applications.

We observed that several editing functions are shared by most graphical editors. Unlike other application frameworks (such as MacApp and InterViews) in which programmers have to subclass these operations, Garnet provides these routines in a manner that should be usable by most graphical editors *without change*. The functions support such operations such as cut, copy, paste, delete, duplicate, group, ungroup, refresh, to-top, to-bottom, print, save, open, etc.

5.6 Higher-Level Tools

One of the most important developments in Garnet is its complete set of interactive user interface construction tools. The ultimate goal of these is to allow *all* of the user interface to be created by demonstration. Instead of writing programs, the designer simply draws the graphics and gives examples of what the end user will do. [Myers 93a] is a book chapter summarizing how demonstrational techniques are used in various Garnet tools.

Of course, the high-level tools are written using the low-level Garnet toolkit. In [Myers and Vander Zanden 92] we discuss why the features of the Garnet toolkit make this task easier. For example, the original version of Gilt took only about three man-months to complete.

5.6.1 Gilt

One recognized problem with using widgets is that it can be difficult to create an attractive layout for dialog boxes and property sheets. Our Gilt interface builder [Myers 93b] allows users to select widgets they want from a palette of choices and place them into a window. Gilt provides access to the full set of widgets, including pop-up widgets such as menubars and option buttons. When complete, the interface design can be tested and saved to a file.

There are two important innovations in Gilt: dependencies among widgets and filtering of values can be demonstrated by the interface designer, and graphical styles can be defined so that widgets will look consistent across multiple dialog boxes and applications.

Demonstrating Dependencies

Conventional graphics toolkits today require the programmer to attach *callback* procedures to most widgets in the interface. These procedures are called by the system when the user operates the widget in order to notify the application of the user's actions. Unfortunately, real interfaces contain hundreds or thousands of widgets and associated callback routines, most of which perform trivial tasks.

We investigated various ways to minimize the number of callbacks in users' programs [Myers 91a]. In most cases, the required tasks can be specified by *demonstrating* the desired behavior. Gilt will watch the demonstration and create appropriate code. Other tasks can be entered by selecting menu items. As a last resort the designer can directly edit the generated code. The goal is to minimize the amount of code that the user must actually generate himself to achieve the required transformation.

If a call to an application function is necessary in the code, Gilt will make sure that the procedure is called with appropriate high-level parameters, rather than such things as a widget pointer or the string labels. For example, a routine to generate a color from red, green and blue values entered using sliders might be called in Gilt as (Make-Color 0.2 0.1 0.9). Other systems, for example, might use a separate call-back procedure on each slider, and would have to know the slider widgets' structure in order to access their values. Gilt will completely insulate any unavoidable callbacks from the particular widgets used.

Implementing Graphical Style Decisions

Often a user or a company will have a particular style in mind for their user interfaces. Even with a predefined look and feel, such as Motif has, many options remain. For example, the user must choose text fonts, colors, arrangements for buttons (horizontal, vertical, tabular), etc. It can be quite tedious to specify the appropriate properties for each widget as it is created.

We investigated ways to define *graphical style sheets* that allow easy editing and implementation of design decisions [Hashimoto and Myers 92]. When a new widget is created, it will automatically be assigned an appropriate style, based on its position and type. For example, a text label at the top of a window might have the ‘‘main-title’’ style and have a large font, whereas a text label in the center of the window might have the ‘‘subtitle’’ style and a bold font. A rule based system checks each new widget’s location and applies the appropriate style. Styles can be edited, and then all widgets using that style change immediately. This system also automatically lays out the components neatly.

5.6.2 Lapidary

The Lapidary tool is used to create new widgets, such as new kinds of buttons, and also application-specific graphical objects, such as the nodes for a boxes and arrows editor [Vander Zanden and Myers 91a, Vander Zanden and Myers 91b]. The designer draws the graphical objects and uses iconic menus to specify the constraints among them. The menus provide the most common graphical constraints found in user interfaces. If the objects are edited, the constraints are maintained. Next, behaviors can be attached to the graphics by filling in dialog boxes. The responses to the end user’s actions can also be demonstrated. In this way, components of a graphical application, or even entirely new widgets, can be created without programming.

5.6.3 C32

When the constraints available from the iconic menus in Lapidary are not sufficient, the C32 tool [Myers 91b] can be used. C32 allows complex constraints to be entered using a spreadsheet metaphor. C32 can be used stand-alone or from an interactive design tool to specify these constraints. It provides significant support to make it easier to specify constraints, including automatically generating references to objects, menus for inserting common functions, and automatic balancing of parentheses.

5.6.4 Marquise

The Marquise tool [Myers et al. 93a] allows the overall interactive behaviors of an application to be described by demonstration. Other Garnet tools allow pieces of an application to be created (Gilt handles dialog boxes, Lapidary handles application-specific graphics, C32 handles constraints), but a tool to put it all together was missing. With Marquise, the user can demonstrate the overall design of application windows and how the various widgets control the behaviors. Features in Marquise include:

- The ability to demonstrate graphical palettes that control what kinds of objects are created and the properties of objects (color, font, etc.).
- Control over when behaviors will start. In a direct manipulation interface, pressing down the mouse button might do many different things, depending on where the mouse cursor is placed and the current global mode.
- Special icons that support the demonstration of what happens when the user presses the mouse button, moves the mouse, and then releases the button.

5.7 Distributing Garnet

In order to demonstrate that Garnet is a useful system, help transfer the ideas into wider use, and serve the computer science community, we remain committed to making Garnet available for general use. Although originally distributed under a restrictive license, Garnet is now in the public domain. Currently, there are over 50 active projects using Garnet all over the world. There is an Internet bulletin board for Garnet, called `comp.windows.garnet`, on which over 300 readers across the world post and read messages daily.

In support of the users we continuously revise the Garnet manual [Myers 92a], and port Garnet to different versions of Lisp for different machines. Garnet currently works in virtually any Common Lisp environment, including Allegro, Lucid, CMU, Harlequin, and CLISP Common Lisps on Sun, DEC, HP, Apollo, SGI, IBM 6000, and many other machines.

We have also performed comparisons of Garnet with other systems. [Myers et al. 92] discusses why the programming style in Garnet is different from systems that use classical object-oriented models. Because Garnet uses constraints, a retained object model, and a separate specification of behaviors, much of the program is written in a declarative manner. This also makes it easier for interactive tools to create interfaces.

This paper also reports on a small experiment where we compared the time to create a particular application using various toolkits (without using any interactive tools such as Lapidary or Gilt). We were happy to see that Garnet appears more effective than others:

System	Language/Obj. Sys.	Time	Lines of Code
Garnet	Common Lisp/Garnet	2.5 hrs	183 lines
CLIM	Common Lisp/CLOS	4.5 hrs	331 lines
MacApp	Object Pascal	9 hrs	1026 lines
GINA++	C++	16 hrs	550 lines
LispView	Common Lisp/CLOS	2 days	500 lines
CLM, GINA	Common Lisp/CLOS	2 to 3 days	350 lines

Unsolicited comments from users have supported these numbers. For example:

> Garnet is awesome!! I have tried out various other GUI's, and Garnet blows them away in terms of ease of use, power, performance, and documentation. It is a real pleasure to use. — *Andy Golding, Mitsubishi Electric Research Laboratories, Inc.*

> Our application is almost done and it took only a fraction of the time that we initially expected, thanks mostly to Garnet. — *George Kutty, Electrical & Computer Engineering, University of California at Santa Barbara*

> I've used Garnet for the last couple of years and I'm very happy with it. Using it its possible to build highly interactive interfaces very quickly. You can do things in Garnet that you wouldn't even even attempt in CLUE and CLIM. — *Roderick Williams, University of Leeds*

> I love Garnet. It's one of the easiest to learn pieces of software I've ever seen, and I'm considering using it for some introductory programming training; combined with the LISP interactive environment, it allows people to see the results of what they do very quickly. The tutorial was excellent. — *Peter Dudey, Oregon State University, Dept. of Computer Science*

We have continually endeavored to write papers about the important ideas in Garnet; in its six-year history, the project has generated 32 articles accepted in refereed journals and conferences, three book chapters, and 13 tech reports. Some recent articles are collected in two technical reports [Myers 90b, Myers 93c]. The latter [Myers 93c] also contains a collection of pictures from some interesting applications written using Garnet.

5.8 Other Contributions

In addition to the direct research on the Garnet system, we worked on a number of important surveys and articles during this project.

5.8.1 User Interface Survey

Garnet is designed to help make graphical, direct manipulation interfaces significantly easier to build. Conventional wisdom rests on little real evidence, but asserts that such interfaces are difficult to program. In fact, many papers in the field were still using old survey data from 1978 when user interface styles were radically different. Therefore, in a collaborative effort with IBM, we designed and conducted a survey to discover how people are actually programming user interface software today [Myers 91c].

Results [Myers and Rosson 92] show that in today's applications, an average of 48% of the *code* is devoted to the user interface portion. The average *time* spent on the user interface portion is 45% during the design phase, 50% during the implementation phase, and 37% during the maintenance phase. 34% of the systems were implemented using a toolkit, 27% used a user interface management system (UIMS), 14% used an interface builder, and 26% used no tools. One of the systems in the survey was using Garnet. Projects using only toolkits spent the largest percentage of the time and code on the user interface (around 60%) compared to around 45% for those with no tools. This apparent discrepancy seems to result from the toolkit systems having more sophisticated user interfaces. The projects using UIMSs or interface builders spent the least percent of time and code on the user interface (around 41%) suggesting that these tools are effective.

5.8.2 Languages for Developing User Interfaces

System builders can choose among many approaches to programming user interface software. In [Myers 92b] we show the results of our May 1991 workshop on programming languages for user interfaces, comparing the techniques that the many people working on this problem have developed. In addition to writing an introductory chapter to this book, we wrote a chapter that demonstrates how the features of Garnet can be considered an advanced programming language, embedded in Lisp [Myers 92c].

5.8.3 Survey of User Interface Tools

We also prepared a survey of user interface tools in general, for publication in another book [Myers 93d]. This article provides more formal definitions of various user interface software terms, like *toolkit*, *user interface management system*, *interface builder*, etc., and will help people understand the current state of the art.

5.8.4 Other Articles

As part of Carnegie Mellon's initiative to form an area for the study of Human-Computer Interaction, we helped write a technical report [John et al. 92] which examines HCI in Carnegie Mellon's School of Computer Science.

[Myers 93e] discusses why user interfaces are important and why they are hard to design and implement. [Jacob et al. 93] and [Olsen et al. 93] are parts of the special issue on an agenda for future HCI research that grew out of an NSF-sponsored workshop.

5.9 Status and Conclusion

In summary, the Garnet research project is exploring innovative toolkit organizations and high-level, demonstrational, interactive tools. In particular, the project is widely recognized as the world-leader in interactive techniques for specifying the dynamic behavior of user interfaces. In addition to its significant research contributions, Garnet is also being widely used at many sites to develop real user interfaces and has been amply shown to be a practical and effective way to create user interface software.

5.10 Bibliography

[Hashimoto and Myers 92]
Hashimoto, O., and B.A. Myers.
Graphical styles for building user interfaces by demonstration.
In *ACM Symposium on User Interface Software and Technology*, pages 117-124. ACM, November, 1992.

[Jacob et al. 93] Jacob, R.J.K., Leggett, J.J., Myers, B.A., and Pausch, R.
Interaction Styles and Input/Output Devices.
Behaviour and Information Technology 12(2):69-79, 1993.

[John et al. 92] John, B.E., Miller, P.L., Myers, B.A., Neuwirth, C.M., and Shafer, S.A.
Human-computer interaction in the School of Computer Science.
Technical Report CMU-CS-92-193, Computer Science Department, Carnegie Mellon University, October, 1992.

[Landay and Myers 93]
Landay, J.A., and Myers, B.A.
Extending an Existing User Interface Toolkit to Support Gesture Recognition.
In *Adjunct Proceedings of INTERCHI'93*. April, 1993.

[Myers 90a] Myers, B.A.
A New Model for Handling Input.
ACM Transactions on Information Systems 8(3):289-320, July, 1990.

[Myers 90b] Myers, B.A.
The Garnet Compendium: Collected Papers, 1989-1990.
Technical Report CMU-CS-90-154, School of Computer Science, Carnegie Mellon University, August, 1990.
This is a collection of previously published papers, released as a CMU technical report.

[Myers 91a] Myers, B.A.
Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-Backs.
In *ACM Symposium on User Interface Software and Technology*, pages 211-220. ACM, November, 1991.

[Myers 91b] Myers, B.A.
Graphical Techniques in a Spreadsheet for Specifying User Interfaces.
In *Proceedings SIGCHI'91: Human Factors in Computing Systems*. SIGCHI, April-May, 1991.

[Myers 91c] Brad A. Myers and Mary Beth Rosson.
User Interface Programming Survey.
SIGPLAN Notices 26(8):19-22, August, 1991.
Also appeared in SIGCHI Bulletin, April 1991, vol. 23, no. 3, pp. 27-30.

[Myers 92a] B. A. Myers, D. Giuse, R.B. Dannenberg, B. Vander Zanden, D. Kosbie, P. Marchal, E. Pervin, A. Mickish, J. A. Landay, R. McDaniel, and D. Hopkins.
The Garnet Reference Manuals: Revised for Version 2.1.
Technical Report CMU-CS-90-117-R3, Carnegie Mellon University Computer Science Department, November, 1992.
Earlier versions were CMU-CS-90-117-R2, May 1992; CMU-CS-90-117-R, June 1991; CMU-CS-90-117, March, 1990; and CMU-CS-89-196, Nov. 1989.

[Myers 92b] B.A. Myers (editor).
Languages for Developing User Interfaces.
Jones and Bartlett, Boston, MA, 1992.

[Myers 92c] Myers, B.A.
Ideas from Garnet for Future User Interface Programming Languages.
Languages for Developing User Interfaces.
In B.A. Myers,
Jones and Bartlett, 1992, pages 147-157.

[Myers 93a] Myers, B.A.
Garnet: Uses of Demonstrational Techniques.
Watch What I Do: Programming by Demonstration.
In Cypher, A., et al.,
The MIT Press, 1993, pages 219-236.

[Myers 93b] Myers, B.A.
The Garnet Gilt Interface Builder: Graphical Styles and Tabs and Techniques for Reducing Call-Back Procedures.
In *Application Builder Session, Seventh Annual X Technical Conference*. January, 1993.

[Myers 93c] Myers, B.A. (ed.).
The Second Garnet Compendium: Collected Papers, 1990-1992.
Technical Report CMU-CS-93-108, School of Computer Science, Carnegie Mellon University, February, 1993.

[Myers 93d] Myers, B.A.
State of the Art in User Interface Software Tools.
Advances in Human-Computer Interaction, Volume 4.
In Hartson, H.R., and Hix, D.,
Ablex Publishing, 1993, pages 110-150.

[Myers 93e] Myers, B.A.
Why are Human-Computer Interfaces Difficult to Design and Implement?.
Technical Report CMU-CS-93-183, School of Computer Science, Carnegie Mellon University, July, 1993.

[Myers and Rosson 92]
Myers, B.A., and M.B. Rosson.
Survey on User Interface Programming.
In *Proceedings of SIGCHI'92: Human Factors in Computing Systems*. SIGCHI, May, 1992.
Also published as Technical Report CMU-CS-92-113 and IBM Research Report RC17624.

[Myers and Vander Zanden 92]
Myers, B.A., and B. Vander Zanden.
Environment for Rapid Creation of Interactive Design Tools.
The Visual Computer; International Journal of Computer Graphics 8(2):94-116, February, 1992.

[Myers et al. 90] Myers, B.A., D.A. Guise, R.B. Dannenberg, B. Vander Zanden, D.S. Kosbie, E. Pervin, A. Mickish, P. Marchal.
Garnet; Comprehensive Support for Graphical, Highly- Interactive User Interfaces.
IEEE Computer 23(11):71-85, November, 1990.

[Myers et al. 92] Myers, B.A., D.A. Guise, B. Vander Zanden.
Declarative programming in a prototype-instance system: object-oriented programming without writing methods.
In *Proceedings OOPSLA'92: Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 184-200. OOPSLA, October, 1992.

[Myers et al. 93a] Myers, B.A., McDaniel, R.G., and Kosbie, D.S.
Marquise: Creating Complete User Interfaces by Demonstration.
In *Proceedings INTERCHI'93: Human Factors in Computing Systems*, pages 293-300. April, 1993.

[Myers et al. 93b] Myers, B.A., Mickish, A. and Hashimoto, O.
The Garnet Gilt interface builder: graphical styles and tabs and techniques for reducing call-back procedures.
Video.
1993
Presented at the Application Builder Video Session, *Seventh Annual* **X** *Technical Conference.* 1993.

[Olsen et al. 93] Olsen, D.R., Foley, J.D., Hudson, S.E., Miller, J., and Myers, B.A.
Research Directions for User Interface Software Tools.
Behaviour and Information Technology 12(2):80-97, 1993.

[Vander Zanden and Myers 90]
Vander Zanden, B., and Myers, B.A.
A Constraints Primer.
IEEE Computer 23(11):74-75, November, 1990.

[Vander Zanden and Myers 91a]
Vander Zanden, B., and B.A. Myers.
Creating Graphical Interactive Application Objects by Demonstration: The Lapidary Interface Design Tool.
In *SIGGRAPH Video Review*. SIGCHI, April-May, 1991.

[Vander Zanden and Myers 91b]
Vander Zanden, B., and B.A. Myers.
The Lapidary Graphical Interface Design Tool.
In *Proceedings SIGCHI'91:Human Factors in Computing Systems*, pages 465-466. SIGCHI, April-May, 1991.

[Vander Zanden et al. 91]
Vander Zanden, B., B.A. Myers, D.A. Giuse, and P. Szekely.
The Importance of Indirect References in Constraint Models.
In *ACM Symposium on User Interface Software and Technology*, pages 155-164. ACM, November, 1991.

6. RESEARCH ON TYPES IN PROGRAMMING

The Types in Programming (TIP) project seeks to understand languages for programming and program specification by exploring the unifying concept of type structure. This community effort utilizes basic research results on the foundations of programming languages as an enabling technology for more immediately practical systems efforts. Moreover, our research addresses such fundamental issues that many of its results promise a lasting impact on the way we compute.

Unforeseen interactions between programming language features make it practically impossible to determine whether programs conform to their specifications. As a consequence, programmers spend more time ‘‘fighting the language’’ than solving their real problems. So large scale formal verification remains impossible, and trustworthy programs remain an unattainable ideal.

In order to move towards a solution to these problems, the TIP project is taking an innovative approach in which the focus of research is the type structure of languages. Research on types in programming seeks to deepen our understanding of the role of types in both the theory of programming languages and in the practice of program development. The three parts of our approach are:

- Design of type systems for programming languages.
- Development of types as specifications.
- Development of a mathematical theory of types.

Our emphasis in the design of type systems for programming languages is on the investigation of uniform type systems which are more general (i.e., admit more programs as well typed) and more precise (i.e., express more program properties) than systems in use to date. In each case, the expressiveness of the type system must be balanced against the efficiency of type inference, and our work in type system design always goes hand-in-hand with experimental implementations to test practical viability of the ideas. Programming languages with more expressive type systems will lead to more reliable programs, as more program properties can be checked statically by a compiler.

One view is that types form a specification (often partial) of program behavior. Then, type checking is a form of program verification. In development of types as specifications, we are investigating how this view of types may lead to new insights into the nature of verified program development and program transformation.

Results obtained in the fundamental research on a mathematical theory of types plays a central role in the development of better programming and specification languages and thus in the practice of programming in the long run.

6.1 Type Systems for Programming Languages

The dichotomy between typed and untyped languages has been the subject of impassioned debate since the earliest days of programming languages. One side claims that untyped languages preclude compile-time error checking and are succinct to the point of unintelligibility, while the other side claims that typed languages preclude a variety of powerful programming techniques and are verbose to the point of unintelligibility. Years of research, however, have changed the terms of the debate considerably. Languages with type polymorphism, type inclusion, and type inference, which ameliorate or eliminate many of the restrictions of early type systems, are becoming more widely used and implemented.

As we gain experience with advanced type systems, and as the mathematical theory of types develops, new applications of types emerge. One such application, the use of types as specifications, allows the notions of types and type checking to be unified with the notions of formal specification and verification. Essentially, types are specifications of program properties, and type checking a form of program verification. In languages with simple type systems, this amounts to proving ‘‘small theorems about large programs.’’ As the power of the type system is increased, the value and importance of the types-based approach also increases.

Two approaches distinguish languages that combine functional and imperative features. Languages such as Algol and some of its descendants use a *call-by-name* mechanism, evaluating arguments only on executing the procedural body, and restrict assignable values more than those that can be passed as parameters. The languages Scheme and ML, on the other hand, employ a *call-by-value* protocol and can assign to variables the same variety of values passable as parameters. We are investigating how to apply advanced type systems to both kinds of language.

6.1.1 Forsythe

Our work on Algol-like designs employs the Forsythe programming language. While similar in level and basic character to its ancestor, Algol 60, Forsythe’s design reflects a clear understanding of appropriate semantics and offers far more uniformity and generality than its predecessors.

During this contract, we continued to develop the Forsythe programming language, which uses intersection types to combine Algol-like and object-oriented programming paradigms. The semantics of Forsythe has been proved to be coherent, which means that any two derivations of the same typing assertion are semantically equivalent. This ensures that Forsythe programs are semantically unambiguous, a critical property of the language.

Intersection type disciplines, such as the one underlying Forsythe, allow many more properties of a program to be expressed and checked than possible with more traditional type systems. The practicality of the intersection type discipline in a realistic programming language depends in a large part on its theoretical properties, such as coherence. We completed a bottom-up type-checker for the language Forsythe. The design of this language has been described in previous reports.

We studied, from a theoretical as well as practical perspective, a language that arises from the combination of bounded polymorphism and intersection types [Pierce 91]. This work shows that the generality of polymorphism can be combined with the precision of intersection types without creating semantic paradoxes. We also show that for a certain natural formulation of bounded polymorphism (proposed in connection with type systems for object-oriented programming languages) the type-checking problem is undecidable.

6.1.2 ML

Our work on *call-by-value* functional languages has mostly been concerned with extensions and refinements of ML. ML has a formal semantics [Milner et al. 90], high-quality implementations, and an ever-increasing user community. It is thus a good vehicle for transferring theoretical ideas into software development practice.

During this contract, we developed an enhancement of the ML type system with ‘‘refinement types.’’ This enrichment of the ML type system allows the programmer to specify, in considerably greater detail than hitherto possible, the properties of data structures, and hence enables the compiler to catch many more errors at compile time. Thus refinement types narrow the gap between types and full specifications of programs, while retaining the essential properties of a type system (such as decidability of type checking).

We describe the basic refinement type system in [Freeman and Pfenning 91]. In short, refinement types allow the programmer to specify recursively defined subtypes of user-defined datatypes. Type inference now becomes a form of abstract interpretation that employs ideas from the intersection type discipline. However, the type system remains closely tied to ML in that refinement types are given only to programs already well typed in ML.

We have further shown how the system of refinement types can be extended to handle side effects, while preserving the property that no runtime type errors can occur in the resulting language. The prototype implementation of refinement types already incorporates this more general system.

In related work we have given a soundness proof of a variant of ML’s weak polymorphism, which has hitherto been an open problem [Greiner 93]. This allows more programs to be accepted by a compiler as well typed without giving up the guarantee that no type errors will occur at runtime.

6.1.3 Module Interfaces

We developed a general theory of modularity for ML-like languages supporting higher-order modules while maintaining the ‘‘phase distinction’’ (i.e., compile-time type checking) [Harper et al. 90]

The use of types in module interfaces is one of the most important application of type systems. We are making significant progress towards understanding a language of modules based on expressive type systems from a mathematical viewpoint. Only with an appropriate understanding of the theoretical properties of module calculi can we expect to develop sound, general, and practically useful type systems for programming-in-the-large.

6.1.4 Type Systems and Object-Oriented Programming

We developed a calculus of mergeable records capable of supporting many object-oriented programming idioms. This calculus investigates a type system with a form of ‘‘constrained quantification’’ that is capable of expressing the constraints required by a language with multiple inheritance [Harper and Pierce 91].

The interaction between the object-oriented programming paradigm and type systems is not very well understood in that common type systems exclude certain forms of object-oriented programs. In this work, we try to show new ways to extend the generality of type systems to encompass object-oriented programming.

6.1.5 Explicit Polymorphism

Determining the impact of full polymorphism on programming style and productivity will require considerable experimentation. For this purpose we implemented a prototype of Leap, a Scheme-like language with a full polymorphic type discipline. Leap offers an opportunity to transfer into practice recent theoretical insights on parametric polymorphism. In addition, its clear connections to higher-order logic will facilitate building specification languages on that foundation.

We have completed a Leap prototype in Standard ML. Part of this implementation [Michaylov and Pfenning 91] is a newly designed type reconstruction algorithm that appears to solve most practical problems associated with the unrestricted use of polymorphism in higher-order functional languages.

Among the conceptual tools we develop is a notion of observational equivalence, which is used to outline a proof that the compiler preserves observable program behavior. This technique compiles functions of well understood inductive types to non-functional data structures so that computation is no longer just beta-reduction.

We also considered the question of whether a useful notion of metacircularity exists for the polymorphic lambda-calculus [Pfenning and Lee 91]. Even though complete metacircularity seems impossible, we obtain a close approximation to a metacircular interpreter through a representation that has the property that evaluation is definable, and furthermore that only well typed terms can be represented and thus type inference does not have to be programmed explicitly.

6.2 Toward a Unifying Theory: Types and Specifications

Mechanical assistance in developing formally verifiable software requires a programming language with appropriate formal properties *and* a language for formulating the properties of resulting programs. Currently available environments are inadequate because their underlying representation languages lack sufficient expressive power. We are working towards a unifying theory of programming notations and logical systems for reasoning about them. Our strategy, based on combining ideas from the Logical Framework (LF) and constraint logic programming, promises to remedy this lack.

In February 1992 we released a first version of Elf, a metalanguage for the specification, implementation, and correctness proof of programming languages and logics. Elf is available electronically and comes with a tutorial and a number of examples from computer science. These examples include a proof of type soundness for Mini-ML and proofs of a number of other properties of programming languages. Among the published experimental results are a fully formalized proof of a compiler for Mini-ML to an abstract machine [Hannan and Pfenning 92], and an implementation of the mathematically complex proof of the Church-Rosser theorem for the untyped lambda-calculus [Pfenning 92a].

We are currently distributing Version 4 of Elf (released on July 1, 1993), which includes a number of efficiency improvements and an Emacs interface that provides sophisticated help in locating sources of type errors in Elf programs. The rationale behind our implementation techniques is described in [Michaylov and Pfenning 93]. The implementation is in Standard ML and is thus immediately available on a wide range of architectures. The unification algorithm employed in this implementation goes beyond LF in that handles polymorphic type variables [Pfenning 91]. Resulting enhancements to the Elf logic programming language allow significantly greater code reusability.

We worked on the design and implementation of a module system for the LF logical framework and the Elf programming language. The complete design is described in [Harper and Pfenning 93]. The goal of this work is to provide structured means to present definitions of deductive systems, including descriptions of programming languages and their semantics.

We designed a refinement of the type system underlying the LF logical framework that considerably enhances its expressive power in that many practical examples can be formulated much more concisely. This extension combines ideas from the intersection type discipline (the basis for the imperative language Forsythe), refinement types, and dependent types. The design is summarized in [Pfenning 92b]. Three related unification algorithms for a type theory with refinement types are given in [Kohlhase and Pfenning 93]. They will form the basis of a future implementation of refinement types as an extension of Elf.

We have also prepared a case study in the expression of program development knowledge in terms of proof transformations. It demonstrates a systematic method for deriving tail-recursive programs by transforming constructive proofs. The implementation is based on principles from type theory and exploits types as a method for specifying program properties. This case study has also been implemented in Elf [Anderson 93].

6.3 A Mathematical Theory of Types

There is a considerable amount of research activity in the area of the mathematical theory of types. Such research leads to applications such as ‘‘types as specifications,’’ and also to a better understanding of the deep structure of programming languages. Typically, fundamental research of this kind does not lead to immediate changes in the practice of software development. However, the results obtained at this foundational level play a central role in the development of the field.

Early type systems suffered from excess rigidity and thus excluded too many intuitively correct programs or prohibited reuse of code. The introduction of the notion of parametric polymorphism addresses this problem but, 20 years after its inception, its theoretical properties are still poorly understood. Our work is a central contribution towards developing a mathematical theory of type structure that encompasses parametric polymorphism.

We generalized the concept of relations over sets to relations over an arbitrary category, and used this concept to investigate the abstraction (or logical-relations) theorem, the identity extension lemma, and parametric polymorphism for Cartesian-closed-category models of the simply typed lambda calculus and PL-category models of the polymorphic typed lambda calculus. This theory forms the basis for a general treatment of parametricity for arbitrary models of the polymorphic lambda calculus. The main result makes precise the idea that the type system of a language with polymorphism enforces data abstraction. We gave a lecture on this topic at the Mathematical Foundations of Programming Semantics Workshop held in Pittsburgh, March 25-28, 1991. Treatments of Kripke relations and of complete relations were also covered [Ma and Reynolds 92].

In [Reynolds and Ma 92], we prove abstraction or logical-relations theorems for general category-theoretic models of the polymorphic type lambda calculus, and derive a plausible definition of ‘‘parametric’’ polymorphism.

We also gave an advanced tutorial at the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages [Reynolds 93]. The tutorial provided an introduction to logical relations and parametric polymorphism and their use in program proving.

In [Reynolds 91] we discuss the problem of semantic ambiguity in languages with rich type structure. When a programming language has a sufficiently rich type structure, there can be more than one proof of the same typing judgment.

Potentially this situation can lead to semantic ambiguity since the semantics of a typed language is a function of such proofs. When no such ambiguity arises, we say that the language is coherent. In this paper we prove the coherence of a class of lambda-calculus-based languages that use the intersection type discipline, including both a purely functional programming language and the Algol-like programming language Forsythe.

We published a paper on the expressive power of the pure polymorphic lambda calculus [Reynolds and Plotkin 93]. Another paper, on the connection of recursion and iteration in functional languages, has also been accepted for publication [Filinski 93]. We also investigated the problem of solving type equations in Cartesian closed categories, which work has applications in type-indexed retrieval of functions from libraries. An extended abstract announcing the results has appeared in the proceedings of the Symposium on Logic in Computer Science [Narendran et al. 93].

6.4 Bibliography

[Anderson 93] Anderson, P.
Program Development by Proof Transformation.
PhD thesis, School of Computer Science, Carnegie Mellon University, October, 1993.

[Brookes 91] Brookes, C.
Using Fixed Point Theorems to Prove Retiming Lemmas.
July, 1991.

[Brookes and Geva 91a]
Brookes, C., and S. Geva.
Towards a Theory of Parallel Algorithms on Concrete Data Structures.
Theoretical Computer Science , 1991.

[Brookes and Geva 91b]
Brookes, C., and S. Geva.
Continuous Functions and Parallel Algorithms on Concrete Data Structures.
In *Proceedings of the Conference on Mathematical Foundations of Programming Semantics*. March, 1991.
Also appears as Technical Report CMU-CS-91-160.

[Brookes and Geva 91c]
Brookes, C., and S. Geva.
A Cartesian Closed Category of Parallel Algorithms between Scott Domains.
In *Programming Languages Semantics and Model Theory*. June, 1991.

[Brookes and Geva 91d]
Brookes, C., and S. Geva.
A Cartesian Closed Category of Parallel Algorithms between Scott Domains.
Technical Report CMU-CS-91-159, School of Computer Science, Carnegie Mellon University, July, 1991.

[Filinski 93] Filinski, A.
Recursion from Iteration.
Lisp and Symbolic Computation 6(3/4), 1993.

[Freeman and Pfenning 91]
Freeman, T., and F. Pfenning.
Refinement types for ML.
In *Proceedings of the SIGPLAN '91 Symposium on Language Design and Implementation*. SIGPLAN, June, 1991.

[Geva 90] Geva, S.
Towards a theory of Parallel Algorithms on Concrete Data Structures.
Technical Report CMU-CS-91-157, School of Computer Science, Carnegie Mellon University, September, 1990.

[Greiner 93] Greiner, J.
Standard ML Weak Polymorphism Can Be Sound.
Technical Report CMU-CS-93-160, School of Computer Science, Carnegie Mellon University, May, 1993.

[Hannan and Pfenning 92]
Hannan, J. and Pfenning, F.
Compiler Verification in LF.
In Andre Scedrov (editor), *Seventh Annual IEEE Symposium on Logic in Computer Science*, pages 407-418. IEEE Computer Society Press, Santa Cruz, California, June, 1992.

[Harper and Pfenning 92]
Harper, R., and Pfenning, F.
A module system for a programming language based on the LF logical framework.
Technical Report CMU-CS-921-191, School of Computer Science, Carnegie Mellon University, September, 1992.

[Harper and Pfenning 93]
Harper, R., and Pfenning, F.
A Module System for a Programming Language Based on the LF Logical Framework.
Journal of Functional Programming , 1993.

[Harper and Pierce 91]
Harper, R. and B. Pierce.
A record calculus based on symmetric concatenation.
In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages, Orlando FL*, pages 131-142. ACM, January, 1991.
Extended version available as Technical Report CMU-CS-90-157.

[Harper et al. 90] Harper, R., Mitchell, J., and Moggi, E.
Higher Order Modules and the Phase Distinction.
In *Conference Record of the 17th Annual ACM Symposium on Principles of Programming Languages*, pages 341-354. ACM Press, January, 1990.

[Kohlhase and Pfenning 93]
Kohlhase, M., and Pfenning, F.
Unification in a Lambda-Calculus with Intersection Types.
In Miller, D. (editor), *Proceedings of the International Logic Programming Symposium*. MIT Press, Vancouver, Canada, October, 1993.

[Ma and Reynolds 92]
Ma, Q.M., and J.C. Reynolds.
Types, abstraction, and parametric polymorphism, part 2.
In Brookes, S., M. Main, A. Melton, M. Mislove, and D.A. Schmidt (editors), *Proceedings of the 1991 Mathematical Foundations of Programming Semantics Conference*. Lecture Notes in Computer Science, Springer-Verlag, 1992.

[Michaylov and Pfenning 91]
Michaylov, S., and F. Pfenning.
Compiling the polymorphic lambda-calculus.
In *Proceedings of the Symposium on Partial Evaluation and Semantics Based Program Manipulation*. ACM, June, 1991.
Also appears in *SIGPLAN Notices* 26(9), September 1991.

[Michaylov and Pfenning 93]
Michaylov, S.,and Pfenning, F.
Higher-Order Logic Programming as Constraint Logic Programming.
In *Position Papers for the First Workshop on Principles and Practice of Constraint Programming*, pages 221-229. Brown University, Newport, Rhode Island, April, 1993.

[Milner et al. 90] Milner, R. and Tofte, M., and Harper, R.
The Definition of Standard ML.
MIT Press, Cambridge, Massachusetts, 1990.

[Narendran et al. 93]
Narendran, P., Pfenning, F., and Statman, R.
On the Unification Problem for Cartesian Closed Categories.
In Vardi, M. (editor), *Eighth Annual IEEE Symposium on Logic in Computer Science*, pages 57-63. IEEE Computer Society Press, Montreal, Canada, June, 1993.

[Pfenning 91] Pfenning, F.
Logic programming in the LF logical framework.
In Plotkin, G.D. and G. Huet (editor), *Logical Frameworks*. Cambridge University Press, 1991.

[Pfenning 92a] Pfenning, F.
A proof of the Church-Rosser theorem and its representation in a logical framework.
Technical Report CMU-CS-92-186, School of Computer Science, Carnegie Mellon University, September, 1992.

[Pfenning 92b] Pfenning, F.
Intersection types for a logical framework.
Technical Report POP 92-006, School of Computer Science, Carnegie Mellon University, December, 1992.

[Pfenning and Lee 91]
Pfenning, F., and P. Lee.
Metacircularity in the polymorphic lambda-calculus.
Theoretical Computer Science (89):137-159, 1991.
A preliminary version appeared in *TAPSOFT '89*, Proceedings of the International Joint Conference on Theory and Practice in Software Development, Barcelona, Spain, pages 345-359, Springer-Verlag LNCS 352.

[Pierce 91] Pierce, B.C.
Programming with Intersection Types and Bounded Polymorphism.
PhD thesis, School of Computer Science, Carnegie Mellon University, December, 1991.
Also available as Technical Report CMU-CS-91-205.

[Reynolds 91] Reynolds, J.C.
The coherence of languages with intersection types.
Lecture Notes in Computer Science: Theoretical Aspects of Computer Software 526:675-700, 1991.

[Reynolds 93] Reynolds, J.C.
An introduction to logical relations and parametric polymorphism (tutorial).
In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, 1993.
Abstract only.

[Reynolds and Ma 92]
Reynolds, J., and Q.-M. Ma.
Types, abstraction, and parametric polymorphism, part 2.
Mathematical Foundations of Programming Semantics, Springer-Verlag Lecture Notes in Computer Science 598:1-40, 1992.

[Reynolds and Plotkin 93]
Reynolds, J.C., and Plotkin, G.D.
On Functors Expressible in the Polymorphic Typed Lambda Calculus.
Information and Computation 105(1):1-29, July, 1993.

7. GLOSSARY

Agate	A tool in Garnet which allows the specification of gestures by example.
AGE	Average Growth Effect; the prediction that a large number of chunks will result in a large match cost in explanation-based learning systems.
Analytica	A Mathematica-based automatic theorem prover for theorems in elementary analysis.
Apprentice	A knowledge acquisition tool designed to facilitate the construction of Prodigy domains.
ASR	Application-Specific Resolver; a specialized tool in object management that facilitates incorporating application-specific knowledge in an optimistically replicated file system.
AST	Abstract Syntax Tree, used in Venari to search object libraries.
BDD	Binary Decision Diagram; a symbolic represention allowing the analysis of behavior of circuits and finite state machines that far exceed the capabilities of other approaches.
C32	A Garnet tool allowing complex constraints to be specified using a spreadsheet metaphor.
CAP	A personal calendar management program.
CBR	Case-Based Reasoning; an analogical reasoning method employed in Prodigy.
Coda	A file system that manages very large distributed object bases and supports disconnected operation.
Color histogram	A technique for arranging color distribution data in image analysis.
Complex EGI	Complex Extended Gaussian Image; an improved EGI that can determine both altitude and position of curved objects.
Congruent aspects	In Image Understanding; aspects of objects that cannot be distinguished from one another based on the values of available features.
COSMOS	COmpiled Simulator for MOS circuits; a hardware verification tool for VLSI.
Dependencies	Object behaviors in Garnet that depend upon the value of other objects for their definitions.
Dyad	Carnegie Mellon's object management security project, focussing primarily on the applications of secure co-processors and formal models of secure systems.
EBL	Explanation-Based Learning, a machine learning paradigm such as that offered by Prodigy and some smaller Soar systems.
EBNN	Explanation-Based Neural Network learning; one learning method offered in Theo.

EGI	Extended Gaussian Image; an Image Understanding tool that determines the altitude of an object.
Elf	A metalanguage for the specification, implementation, and correctness proof of programming languages and logics.
Factorization	A method of image analysis that combines displacement measurements from many image frames and then directly computes shape and motion, without first having to compute depth as an intermediate step.
First-class store	An object that represents the state of all mutable values at a particular point during program execution.
Flying continuous stereo	Techniques that can obtain dense and accurate depth maps from an image sequence taken by a moving camera with a very small interframe displacement.
Forsythe	A programming language similar in level and basic character to its ancestor Algol 60.
Gilt	An interface builder in Garnet that allows users to select widgets from a palette of choices.
Graphical styles	The appearance of graphical objects that make up a user interface.
Inspector	A powerful debugging tool in Garnet that views and edits the properties of objects.
Interactors	Parameterized objects in the Garnet system which are used to control all standard direct manipulation behaviors.
KR	An object-oriented language used for specifying constraints in Garnet.
Lapidary	A Garnet tool used to create new widgets by drawing their appearance and selecting constraints, without programming.
Leap	A Scheme-like language with full polymorphic type discipline.
LF	Logical Framework; the type and logic specification language used in Carnegie Mellon’s Types in Programming research.
LP	Larch Prover; a powerful tool in Object Management for reasoning about specifications.
LSL	Larch Shared Language; a specification language to describe mathematical abstractions such as sets, relations, and algebras; LSL’s semantics are defined in terms of first-order theories.
LTO	Linguistic Task Operator; a phenomenon inherent in Soar’s language that provides a method for problem solving by deploying the skills of comprehension and generation in service of a task.
Marquise	A Garnet tool allowing the overall interactive behaviors of an application to be specified by demonstration.
NESL	A machine independent programming language for massively parallel computers, based on the idea of nested data-parallelism.

NL-Soar Soar's natural language system, which incorporates a single-path parsing algorithm, top-down and bottom-up knowledge, and a limited repair capability for correcting wrong interpretations.

NoLimit Prodigy's nonlinear planner.

Observational equivalence A conceptual tool used to outline a proof that the compiler preserves observable program behavior.

Oz An environment to let writers and artists create high quality interactive fiction and animated worlds based on AI simulations.

Paraperspective A new model of image perception that captures the most important aspects of perspective projection, yet leads to simple mathematics.

Refinement types An enhancement of the ML type system allowing the programmer to specify the properties of data structures.

Reactive system An approach to formal verification of hardware systems that consists of a number of independent agents communicating and synchronizing according to a specific protocol.

Replication-based copying garbage collection A method of copying garbage collection which does not destroy the original replica when a copy is created, allowing multiple copies to exist.

Rete Soar's underlying match algorithm.

RVM Recoverable Virtual Memory; regions of a virtual address space on which transactional guarantees are offered in Coda.

Sequential processor An approach to formal verification of hardware systems that conceptually executes a single sequence of operations.

Signature matching A technique allowing the reuse of a software component, based on signature information easily derived from the component.

SMV Symbolic Model Verifier; a tool that checks specifications given in the temporal logic CTL and includes a built-in hardware description language.

SourceGroup An open-architecture, selective recompilation system designed for interactive compilers and interpreters.

SQL Structured Query Language; a widely-used database query/interface language.

Symbolic trajectory evaluation An approach to sequential processor verification combining the capabilities of symbolic, switch-level simulation with limited forms of temporal analysis.

SynRGen A synthetic file reference generator for stressing UNIX file systems.

Taskification A process in Soar by which task-specific knowledge can be smoothly integrated into comprehension.

TAQL — An experimental language for rapid task acquisition in Soar5.

Temporal-color space — A four dimensional space, consisting of the red, green, and blue axes and one temporal axis, which allows the analysis of a *sequence* of color images.

Temporal logic model checking — A technique in verification of reactive hardware systems in which specifications are expressed in a propositional temporal logic, and circuits are modeled as state-transition systems.

Texture segmentation — The technique of grouping together regions of an image based on their similar texture.

TheoGT — A faster, smaller reimplementation and simplification of the Theo system.

Trace algebra — An abstract algebra having a set of traces as its carrier, along with operations of projection and renaming on traces.

Trace structure algebra — An abstract algebra in which operations of parallel composition, projection, and renaming are defined on trace structures.

Uni-Rete — A recasting of Rete technology in a new match algorithm in Soar5, allowing performance ten times faster than the standard matcher.

VAC — Vision Algorithm Compiler; an Image Understanding tool that can automatically generate a machine vision program to recognize and locate an object based on a solid model.

Venari — Carnegie Mellon’s persistent object repository management project.

Virtual aggregates — Virtual objects in Garnet that pretend to allocate objects to the rest of the system but actually maintain only a small amount of state information.

Widget — A user interface element in Garnet, such as a menu, a virtual button, or a scroll bar.

Z — A model theoretic specification language based in set theory that has syntax and type checking programs available.

References

Table of Contents