MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

MEMO No. 299 SEPTEMBER 1973

PROPOSAL TO ARPA
FOR
RESEARCH ON INTELLIGENT AUTOMATA
AND
MICRO-AUTOMATION
1974 - 1976

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contracts N00014-70-A-0362-0003 and N00014-70-A-0362-0005.

TABLE OF CONTENTS

1974 PROPOSAL TO ARPA

SUMMARY

The results of a decade of work on Artificial Intelligence have brought us to the threshold of a new phase of knowledge-based programming -- in which we can design computer systems that

(1) react reasonably to significantly complicated situations and

(2) perhaps more important for the future -- interact intelligently with their operators when they encounter limitations, bugs, or insufficient information.

This proposal lays out programmes for bringing several such systems near to the point of useful application. These include:

A physical "micro-automation" system for maintenance and repair of electronic circuits.

A related "expert" problem-solving program for diagnosis and modification of electronic circuits.

A set of advanced "Automatic Programming" techniques and systems for aid in developing and debugging large computer programs.

Some Advanced Natural Language application methods and systems for use with these and other interactive projects.

A series of specific "expert" problem solvers, including Chess analysis.

Steps toward a new generation of more intelligent Information Retrieval and Management Assistance systems.

The application areas are chosen to advance our general competence, clarify dark areas, and provide working prototypes that should be especially helpful in bringing other areas to the same practical stages. The proposal details plans for two years research work, beginning January 1974, and we further draft what we believe must be done to bring each area to the point of major practical applications in the order of five years.

OVERVIEW OF THIS DOCUMENT

This proposal follows more than usually quickly our previous proposal of barely six months ago. Thus, much of this proposal is for continuation of work already under way. However, we can now specify more precisely the problems and milestones we expect to encounter and achieve, and more precise assignments of people and resources in the laboratory to specific subtasks. This is in response to the considerable increase in size and complexity of the projects, as compared to those of the past which were usually attacked by individuals rather than by groups.

SECTION 1 presents our work-plan for the next two years (plus), classifying the tasks into seven areas of concentration. Each area is further broken down into one or two primary topic and a number of secondary topics. Each primary topics has clear-cut milestone goals for the two year period. Those goals mentioned in Section 1 are selected to illustrate the nature of the problem area; they are real goals but not necessarily the most important or most difficult.

These areas have discrete milestones and identifiable personnel with responsibility to those goals. However, to some extent these assignments are nominal, in that there is a great deal of interaction and sharing of expertise within the laboratory. In other words, Section 1 presents a "linear" model of the laboratory; listing seven "terms" as though independent and ignoring the important interactions.

PROJECTS 1 through 7 are the body of the proposal. Each states the goals of one of our application areas, its problems, proposed methods, resources needed, expected "milestones", and the people who will be responsible.

PROJECT 1 describes in detail the proposed Visual Electronic Repairman Project, which includes the goals of our previous work on Micro-Automation and Machine Vision.

PROJECT 2 describes the "Knowledge-Based Electronic Repairman" project.

PROJECT 3 describes our Automatic Programming, Debugging and Self-Documentation research programme.

PROJECT 4 describes our work toward Natural Language interactive semantic systems.

PROJECT 5 describes a number of theoretical projects concerning representation of knowledge, learning, logic, etc.

PROJECT 6 describes the Chess project and several other small "expert problem solving" projects.

PROJECT 7 proposes two study projects concerned with improving the state of the art in Information Retrieval and in Management Assistance.

SECTION 3 discusses the state of A.I., in terms of methods, outstanding problems, and probable time-scales. It also explains the cohesiveness of the entire area of study.

SECTION 5 is the A.I. Laboratory Bibliography.

SECTION 1
SCHEMATIC OUTLINES OF THE PROJECTS

1. Topics with emphasis on VISION AND MANIPULATION

PRIMARY TOPIC: Circuitboard Repair Robot

SECONDARY TOPICS: Manipulating liquids
Seeing irregularly shaped forms
Hand-eye coordination
Micro-Automation development
Inexpensive equipment for research

2. Topics with emphasis on DEBUGGING ELECTRONIC CIRCUITS

PRIMARY TOPIC: Electronic Trouble Shooting

SECONDARY TOPICS: Syntax and Semantics for circuits
Diagnosis of faults
Planning of signal tracing
Planning of repair

3. Topics with emphasis on AUTOMATIC PROGRAMMING AND DEBUGGING

PRIMARY TOPICS: Debugging Electronics and Graphics Programs
Implementation of ACTOR formalism

SECONDARY TOPICS: Classification of common program bugs
Intention-Oriented Automatic Programming
Automatic Annotation and Self-Documentation

4. Topics with an emphasis on NATURAL LANGUAGE

PRIMARY TOPICS: Semantic theories of Syntax (with W. Martin)
V. Pratt's syntax language development

SECONDARY TOPICS: Representation of extended events and scenarios
Interfaces for other projects

5. Topics with a general THEORETICAL EMPHASIS

PRIMARY TOPICS: Representation of knowledge: Frames, Actors
Classification of Common-Sense Knowledge

SECONDARY TOPICS: Inductive Inference (Solomonoff)
Mathematical Complexity
Modal Logics (Geiser)
Physiological Theories (Marr)

6. Topics with an emphasis on EXPERT PROBLEM SOLVING

PRIMARY TOPICS: Chess and Surprise Analysis (Greenblatt)
Geometry (Brown)

SECONDARY TOPICS: Theorem-Proving (Nevins)
Qualitative Physics
Decision under uncertainty
Advanced Learning machines

7. Topics concerned with IDENTIFYING AREAS OF APPLICATION

SUB-TOPICS: Advanced Automation; Micro-Automation
Advanced Information Systems
Heuristic Information Retrieval
Personal Management Assistants
Undersea and Space
Overview and Assessment of Problems in AI

SECTION 2
THE PROJECTS

This section is the body of the technical proposal. It is subdivided into seven PROJECTS. Each subsection begins with a compact overview with, more or less, the following format:

DEFINITION: A brief explanation of what the project is about.

MILESTONES: We divide each project into phases as appropriate. Milestones are given for each phase. We give projections beyond the two-year proposal period to show how we envision these projects coming to fruition over the subsequent few years.

It is understood that this is a very difficult field, our goals are ambitious, and our standards are extremely high. We have tried to be realistic about these estimates but some may take longer. The overall hope is to get each of these areas onto solid foundations -- in the research and prototype sense -- within three years, so that dissemination of techniques and equipment can put them well on the road to practical exploitation within five years.

APPLICATIONS: Each subsection summarizes briefly some applications of the project to important problems.

PROBLEMS: We mention the outstanding difficulties and bottlenecks that appear to be the most serious.

COSTS: The especially expensive aspects of each project. Only a few require "special" equipment, but all except the most theoretical areas require unusually heavy computational services under large-memory, time-shared operating systems.

PERSONNEL: The principal innovators and people responsible for results, and their associates as known at this time.

Acknowledgement: Much of the text of the following sections is adapted from drafts by P. Winston and B.K.P.Horn Project 1; G.J.Sussman, Project 2; I. Goldstein, Project 3.1; C. Hewitt, Project 3.2; V. Pratt, Project 4; R. Greenblatt, Project 6.

PROJECT 1
VISION, MANIPULATION, MICRO-AUTOMATION
The Physical Electronic Repairman

DEFINITION: We want to bring machine vision to the point where one could feel comfortable about saying that the computer "can see". As emphasized in our earlier work, this is not a well-defined, isolated task, because perception in general, and vision in particular, cannot be cut away from general knowledge and intelligence. But within the microworld of the Electronic Repairman, we can define what it means to see, and many useful applications are within reach.

We already can get the computer to scan and "understand" well-controlled scenes involving objects with neat geometric shapes, some aspects of standard printed circuits, and some moderately complicated shadow situations.

Specifically, this project will be concerned with real-world visual analysis of the kinds of scenes found inside electronic assemblies.

MILESTONES:

JULY 1974
- Visual analysis of printed circuit layouts.
- Recognition of electronic components

DEC. 1974
- Mechanical inspection of solder joints.
- Use of test probes at circuit points
- Use of Miniature Hand-Eye system
- Pouring liquids with visual control

DEC. 1975
- Visual inspection of solder joints.
- Identification or verification of physically broken part
- Diagnosis of simple malfunction
- Physical replacement of defective components.
- Connection to electronics debugging programs [Project 2]

DEC. 1977
- Connection with NATURAL LANGUAGE ASSISTANT.
- Connection with Automatic Programming Assistant
- Correlation of physical component with verbal description.
- Assembly of a whole kit.

The latter is also an ambition of the Stanford Project, and close collaboration should be possible by that time.

APPLICATIONS: Assembly, inspection, maintenance, and repair of computers and other electronic equipment.

Generalization to similar functions of small mechanical assemblies.

Servicing of electronic assemblies of larger systems, e.g., electrical systems of engines, antennae, undersea and space devices. (Eventually) Maintenance of sub-miniature systems.

Computer vision has extensive applications to supplementing other systems. For example, such a system could maintain surveillance over the area around a dangerous machine, a secure installation, an intensive-care patient, an object-in-road-ahead warning.

ENGINEERING PROBLEMS:

Curved objects	Representation of components.
Shading and Texture.	Direct range-finding.
Arm and hand design.	Arm dynamics.
Force-feedback sensation.	Motion tracking.
Motion parallax.	

Qualitative physics of components, wires, etc.
Understanding shading and texture
Developing descriptive languages for representing components.
Developing a language for convenience to human operators.
Heterarchical programming for real-time interrupt environment.

COSTS: Use of very large programming systems and memory
Diagnostic display
Completion of Micro-Automation Laboratory.

PERSONNEL: Prof. Winston (vision system development and software)
Prof. Horn (hardware and software development)
Research Staff, Consultants, Students (see below)

TECHNICAL ASPECTS OF PROJECT 1
RESEARCH IN MACHINE VISION

Prof. P.H. Winston ---------- Prof. B.K.P. Horn

T. Finin	J. Lerman	R. Woodham	R. Boberg
J. Hollerbach	M. Dunlavey	S. Fahlman	M. Lavin
R. Waters	M. Billmers	G. Dresher	W. Kornfeld
E. Freuder	S. Siesinger	M. Adler	D. Marr
T. Lozano-Perez	V. Scheinman	C. Flateau	R. Nofteker

INTRODUCTION

Work on machine vision has progressed rapidly in the last three years. Many basic issues are now more sharply defined, permitting us to focus outside the restricted world of carefully prepared simple polyhedra.

At the "performance" level, we can take a collection of flat-sided objects of assorted shapes, pile them in a disorderly heap, and ask the program to analyse, disassemble, and rearrange the objects into another, orderly structure. The latter can be specified by a symbolic description or by presenting a physical example to be analysed by the system. Many "low-level" vision problems had to be solved to reach this level of performance. Many of them are summarized in our January, 1972 Progress Report, and much more detail is available in technical notes and reports.

It is important to note that we have made no compromises in our original long-term goal to set a firm foundation for Monocular machine vision! This vision system works as well on Pictures of a scene as it does on the physcial scene itself. It is not based essentially on the use of physical range-finding methods, tactile-probe exploration, or other "active" sensors.

This is not to say that active sensors are not valuable! We plan many uses of them in our project. We simply want to underline the scientific importance of the systematic work on what one might call picture-vision, because casual onlookers might be unduly impressed with how easily one can get superficially similar practical applications by more application-tailored methods. But the understanding that has come from the study of the pure, monocular vision problem is a more solid and permanent addition to our "general" capabilities, both practical and scientific. This knowledge will always be available when active systems run into difficulty because of (for example) large distances, power limitations, monitoring through TV type sensors, need for not disturbing the scene, etc. And perhaps most important, this work has already made outstanding contributions to modern cognitive psychology in in understanding human vision. See, for example, the

widely used introductory psychology textbook of Donald Norman, or the assessment of Sutherland in the British SRC report on Artificial Intelligence research (see Section 3).

Problems remain, to be sure, but now that the construction of a multi-purpose Blocks-World hand-eye system is behind us, it is time to reorient our efforts towards richer domains.

First, we need to expand our basic features to include texture, color, shading, sharpness of focus, highlights, shadows, and motion.

Second, we want to study visual situations in which perceived context can have a substantive role in analysis.

Third, we plan to extend the interaction with other concentration areas around the laboratory so as to profit from advances in natural language, representation of knowledge, problem solving, advanced programming developments.

B. PROGRESS TO DATE

Before outlining our position with respect to further work, we wish to describe a few recently completed projects that seem likely to support new studies.

1. David Waltz has worked out a semantic theory of polyhedral line drawings that is a major breakthrough in several respects. The theory gives deep insights into the success of earlier work and provides a powerful analysis capability for separating regions into bodies; in identifying edges as convex, concave, obscuring, shadow or crack; in using shadows to determine contact; and in reasoning out the orientation of object faces. That more information should simplify problem solving is obvious; Waltz has gone far beyond the truism and shown how the idea can be worked out using a formalism and representation structure that should contribute to work in advanced systems for both visual and linguistic work. In all early vision projects, shadow boundaries caused malfunctions because they were often interpreted as physical boundaries; in Waltz' system they are exploited in several ways to correct other kinds of errors and ambiguities, even to asserting that a missing line must exist and should be looked for more carefully.

2. Previous vision systems suffered from an artificial division into line-finder/scene-analysis partnerships, communicating only by way of a handed-over line drawing. The new systems of Jerry Lerman and Yoshiaki Shirai show how the barrier can be eliminated and how high level knowledge of physical constraints and partial analysis can guide the filters and trackers that most intimately deal with

low-level intensity information. The systems are thus prime examples of the heterarchical programming concept discussed elsewhere in this proposal.

Briefly, the problem is this. In older systems (as well as in older psychological theories of vision) the process was divided into steps, for example:

1. Find distinctive visual feature points, e.g., large gradients.
2. Aggregate them into higher-level elements, e.g., lines.
3. Aggregate those into, say, regions.
4. Aggregate these into, say, "objects".
5. Identify or "recognize" the objects.
6. Aggregate the objects into familiar structures.

Such systems worked, if at all, only on carefully prepared scenes and "toy" problem demonstrations. The trouble is that there are so many places for errors, and these propagated so mercilessly, that the chance of the whole chain working was too small to be useful. In the new systems we have found ways to use knowledge at a high level -- say, about what kinds of edges occur in shadowed, concave regions, to continually monitor the performance of the low level "line-finders" operating at the primitive "scanning" level.

3. Tim Finin has given the evolving vision system considerable deductive depth through several goal-oriented programs. One of these specializes in using a theory of "perceived groups". Often, some of an object's individual dimensions, position, or orientation parameters are indeterminate because of an obstruction in the line of sight. In these situations the vision system hypothesizes the missing information, using other objects considered similar by virtue of alignment in a stack, a common purpose, or simple proximity. This is one entry into the area of context driven analysis.

4. Finin, Lerman, and Siesinger have completed a visual feedback module that checks the position of a block after positioning by the hand. Then it jiggles it into place if its positional error exceeds a small threshold. This feedback link makes possible exploiting the random-access capability of a programmable image acquisition system by looking only at points lying on a small circle around expected vertex locations. Finin and Lerman have also completed a touch feedback module for use in certain cases when a monocular image is inherently ambiguous.

5. Bob Woodham has done initial complementary work on visual motion tracking. As the first step in a coffee-pouring demonstration, he has worked out and compared several mechanisms for monitoring the rising level of coffee in a stylized cup.

Although this is still in a demonstration phase and not integrated into the system, we believe the mechanism will extend smoothly to such skills as shadow-aided placement of delicate objects.

6. Scott Fahlman has devised a construction planning system which solves problems in two distinct directions. First, three dimensional modelling skill has been developed in the form of sophisticated touch and stability tests. Second, in cooperation with the specialists in CONNIVER language, he has demonstrated the need for and use of advanced control and data base mechanisms. The system can plan fairly complicated constructions requiring temporary scaffolding supports.

7. Rich Boberg has explored the problem of reversing the analysis process, that is, reconstructing a scene from an abstract description. We believe this is the first step toward an automatic design system where the machine contains and uses considerable common sense knowledge about the constraints inherent in a physical world. In the next section we will discuss how Dunlavey's work pushes still further in this direction.

8. John Hollerbach probed the problem of describing complex shapes through work on complicatd, higher order polyhedra. His heuristic theory of projection shows how many objects can be sensibly decomposed into basic shapes, modified by protrusions and indentations.

9. In another domain, Mark Adler has shown how to make progress toward solving the problem of line drawings with curves. In a style reminiscent of initial work on polyhedra, he has outlined an approach to the analysis of some highly constrained kinds of drawings. This should contribute conceptually to work on more general real vision, to diagram reading and manipulating services, and eventually to personal assistant systems in which sketches must supplement natural language commands that are more clearly explained graphically.

C. PROPOSED RESEARCH IN MACHINE VISION

The traditional approach to "low-level" vision has been to bring in familiar mathematics from linear systems theory and elsewhere, and use it in generalized form. Certainly texture has been a primary source of problems for people interested in Fourier transforms and statistics, and in our own laboratory Prof. Horn has applied the mathematics of partial differential equations to the problem of deducing three-dimensional shape from two-dimensional shading information. But while mathematically derived computations on picture data are important to understand, the research does not always couple well with what one hopes to get out of an intelligent machine. Transformations of textured

pictures do not seem to help much in identifying a solid as wood, plaster, or carpet. Horn's methods, while accurate in producing space curves on uniformly painted surfaces, still do not go far toward the hierarchical data structure which on the highest level says that a surface appears to be part of a sphere, cone, or cylinder.

What we are after are common sense qualitative theories of low level vision processing, that can exploit constraints easily expressed only at higher levels of description.

Shirai's paper, "A Heterarchical Program for Finding Objects," is prototypical of what we would like to have come out of our new studies. In that paper he describes a program we believe to be the best available translator of picture arrays into line drawings. It is instructive to see why it is so good.

The program consists of a feature point detector, a line tracker, and a line proposer. Shirai's feature point detector employs only a simple differencing operation used widely before. But Shirai couples that differencing operation with a heuristic filtering program which checks the shape of the contrast curve against a four-point quality check list. Although simple, these tests are not the sort of things one builds into a uniform, position independent, linear filter.

The point is further illustrated in Shirai's exploitation of tracking and proposing programs about which classical knowledge-free mathematical methods have nothing at all to say. Again we find lists of common sense facts about the demonstration universe which translate into goal-oriented programs that get the job done.

It is this qualitative, common sense spirit that we want to extend to larger systems.

In this direction, Horn, Lavin, and Marr have undertaken a new study of color, asking how a machine might exhibit the same insensitivity that man has when naming colors under varying illumination spectra and incident light distribution. The "retinex" theory of Land is regarded as a strong first step. In essence, that theory argues for color naming on the basis of three independently derived lightness orderings, one each in the red, green, and blue. The lightness orderings in turn are determined at region boundaries with no part played by slow drift in absolute intensity. Land argues that the independence of the lightness determinations prevents shift in illumination from distorting perceived color and further that the discounting of slow drift across faces prevents oblique lighting from disturbing the relative lightness measures.

But at a detailed level we believe more work is to be done. Land's lightness measurement algorithm involves an unsatisfactory random walk procedure that wanders about the scene seemingly in search of the

brightest region against which to normalize other regions.

We hope to substitute a more reasonable algorithm that reflects the two dimensional nature of the problem. Horn is working on the details of a theory that involves a successive differencing and thresholding, and a calculation resembling simple solution of resistive networks. The method may lend itself to implementation in simple parallel hardware.

We believe this work will be important to robots, especially where color is often of first importance in identification and where identification must be accurate under all sorts of varied natural and artificial illumination.

Continuing this deliberate program of formulating in machine-usable form common-sense observations about vision, Winston is trying to develop a qualitative theory of how one can determine solid shapes from shading and highlight information. In earlier work, Horn has shown how surface line equations can be arrived at analytically under assumptions of uniform reflectance. What Winston is after is a qualitative theory of shading facts that allows direct hypothesis about the approximate shape suggested by a surface without recourse to that surface's borderline shape. And eventually, one would want to be able to eliminate Horn's uniform surface assumption in favor of using additional knowledge of plausible intrinsic colorations of surfaces.

The theory, like Shirai's heterarchical line finder, is expected to take the form of a collection of procedurally embedded facts relating the relative location of highlights, shading gradients, and light sources to the conclusions about the three-dimensional nature of the sample. Informally, such ideas are well known; the work of J. Gibson in particular is widely acclaimed as important in explaining human perception of three dimensional configurations outside (and often, even, inside) of the range of focus or stereoscopic determination. But although the ingredients of such theories have been available for a very long time, this will be the first attempt, we believe, to weld them together into a coherent (and useful) system.

John Hollerbach is working on the complementary problem of devising a description system rich enough to represent real curved surface objects. He expects to build on his success with polyhedra in which description segments divide complex objects nicely into protrusions, indentations, and basic projections of simple plane figures. He is nearly finished with the first generalization, that of describing the myriad jugs, bowls, crocks, and amphorae that make up the world of archeological pottery. When finished, his system will describe these objects in terms like "high shoulders, "flared base," and "narrow neck," just as does a human specialist in the field.

We have two problems in learning how to introduce textured objects to the machine. The first is to separate texture boundaries from object

boundaries between objects with the same texture. The second problem is how to use texture to determine the orientation of a surface. This is an old idea, as seen in many papers by J. Gibson, but earlier work has not included detailed theories demonstrated by or potentially demonstrable by a machine. Once again, we expect the description problem to be a key focus in our approach. Without taking image space into Fourier space we will want to study several potential processes for 1) determining texture granule boundaries, 2) calculating texture granule features such as area, shape, and boundary characteristics, and 3) noting differences between two textures as a prelude to relative orientation conclusions. This works with a project in which Eugene Freuder wants to bring the most sophisticated knowledge to bear on identifying real objects. His focus is on the interface between image information and world knowledge. He cites as inspirational the work of Hewitt on PLANNER, Sussman and McDermott on CONNIVER, Winograd on the semantic interface, and Waltz on constraint exploitation. We believe this work will uncover general ideas about problem solving in a heterarchical system and force a step forward in real world vision capabilities. Michael Dunlavey has the equally difficult job of understanding what he calls "interface knowledge": that knowledge which is required to understand how general concepts interact by delving into the details of their descriptions until the level is reached where interaction takes place. Specifically, Dunlavey has chosen the demonstration world of construction with toy bricks. The general concepts are notions like "wall," "door," "chimney," and "roof." The interface knowledge concerns the description of each of these in terms o their constituent repeated brick patterns. Generalization to deal with an important part of the design of real houses, buildings, ships, and other constructions seems smooth and continuous.

D. VISION MILESTONES

The chart below summarizes our major vision activities.

color	Correctly name the colors of randomly arrayed colored papers under a variety of illuminants. (Well under way)
	Use color in conjunction with established image-processing techniques to identify boundaries. (late 1974)
texture	Separate and name wood, metal, cloth, paper, plaster, and a few other surfaces. (1975)
	Use texture as an aid in determining approximate surface orientation. (1974)

shading	Devise a qualitative shading and highlight theory sufficient to correctly suggest flat, cylindrical, spherical, and conical sections without recourse to directly measured depth information or the numerical integration method. (1975)
description	Complete first order theory of tiered; hierarchical descriptions and accompanying program for planning toy houses from bricks. (Ph.D. thesis in progress)
	Complete a theory of curved object description and accompanying program capable of describing vases in humanly acceptable form. (Thesis in progress)
	Extend work in curved object description to deal with objects commonly found in a kitchen or other complex room. (1974-75)
	Marry description and analysis tools into a system able to describe the kitchen or other class of objects from camera input (1975, but some results already)
Heterarchical systems and context driven analysis	Search for and identify a specified object in a clutter of tools notwithstanding dirt, reasonable obstruction, and considerable shape abberation. (Long-range goal)
Hand-eye coordination	Cause two objects to touch gently using shadow information. (1974)
	Devise monitor capable of efficiently monitoring a large area for motion.

E. THE MODULAR VISION AND MANIPULATION LABORATORY

International economic problems, environmental questions, worker satisfaction and a host of other issues argue strongly for the development of an advanced productivity technology. Our modular "mini-robot" laboratory effort responds to this need through its two primary goals:

1) To develop a modular set of vision and manipulation tools suitable for substantive research that costs less than $75,000. We believe that such a laboratory kit will widely stimulate research on advanced productivity. We measure our success here in proportion to the degree of acceptance of our laboratory kit as adopted by other research groups.

2) To reorient a substantial portion of our own laboratory's efforts toward applied work. We believe our own theoretical work in vision places us in a strong position for speeding toward early application achievements. Progress here can be equated with the use of machines in industry whose existence can be credited to our work.

Our plan to achieve these goals has three major parts: 1) assembly of the hardware by purchase or construction, 2) programming of basic software support programs, and 3) successful demonstration of the equipment on a specific application task.

Hardware for the Vision and Manipulation System

Hardware development and selection has been a major focus during the first years. We now have selected and acquired a computer configuration, a vidicon system, and a digitally driven x-y table. A new arm designed for us is under construction for December 1973 delivery. A 512 linear array and mirror drivers are here and the design of a new camera using them is being completed.

Image Input

Both the image dissector and the vidicon image sensors suffer from about a dozen major defects each. It is reasonable to build a simple and inexpensive image input device using a now-available low-noise, high-sensitivity P.I.N. photo-diode, an F.E.T. op-amp and a fast mirror deflection system. Interfacing requires only two D/A's and one A/D. Its speed should be comparable to that of the image dissectors of older vintage running at a comparable signal/noise ratio. The device would have lower geometric distortion, scatter, and better uniformity of response. (The use of the same sensing device for each point is an important feature.)

In a related effort, we will evaluate a system which eliminates one dimension of mirror scanning by using a Reticon linear array and a single scanning mirror. (These devices are normally used for binary inputs only. We must check out their uniformity, bloom, noise, range, and meaningful intensity resolution.)

Should neither of these "random access" schemes work out, our backup plan is to work with an existing image dissector camera head. Our plan is to use a mini-computer rather than a special design such as the video processor in recognition of the low prices now associated with mini-computer processors.

At the other end of the speed spectrum, vidicon rates are too high for any real time mini-computer processing. One way of lowering the data rate and reducing the expense of digitizers and memory space is to read only one point per raster line - a whole image now takes about 10 to 15 seconds, which is compatible with the time it takes to process the image. Such a system is expected to be a primary image source during the next year.

Image Output

We do not at the moment have a satisfactory output device for presenting intensity modulated images of acceptable resolution. Such presentation is important both for monitoring how the image input devices perform and for presentiing the results of processing on the image. The usual display devices suffer from a lack of dynamic range, insufficient quantization of intensity, and in some cases from insufficient resolution and flicker. For hard copy, we can use conventional devices with multiple exposure.

Range Finding

No entirely satisfactory range-finding method has been developed yet. The slit type range-finder using a laser source is adequate for many purposes. Time-of-flight techniques are under study at the Draper Laboratory, and we feel inclined not to explore this technique ourselves although we continue to follow the Draper Laboratory's progress.

We plan to use random access linear image sensor coupled with a spot light-source, reducing by a large factor the power required. Suitable tracking algorithms should not be hard to develop. Techniques for use of visual range information have been explored already by the vision group at Stanford and at SRI, by our associates in Japan, and at General Motors, and there is quite a lot of experience to draw on.

Manipulation

More versatile finger configurations have to be worked out. Gripping objects which do not have parallel sides requires more complicated fingers, possibly pliant and with more degrees of freedom. We are investigating a multi-finger arrangement which allows the force applied to be resolved by strain-gauges built into each finger.

We continue our interest in truly small manipulators. Carl Flateau, a consultant, is examining scaling laws in connection with a new hand he is designing for us. Another consultant, Russell Seitz, has advised us about scaling problems with physical materials and about further study of biological systems on this scale.

To complement the hand, wrist, and arm work we plan to develop suitable tools and the means to transfer power to them. Programming must consider the tools to be extensions of the hand with respect to dynamics and force sensing. Tools are needed for: voltage-probing, cutting wires, soldering, desoldering, cleaning, holding in fixed orientation, bending.

The Scheinman arm, which is 2/3 human scale, is nearing completion. It was designed for us while he was in residence during 1972. We may contract for construction of Flateau's 1/4 scale arm. These arms will facilitate assessment of the usefulness of tachometer feedback, direct computer servoing, advantage of counterbalancing and the like. More attention can now be turned to wrist and hand considerations. Small semi-conductor strain-guage wrists, preferably of one-piece construction, will be needed. We need electronics to make it sensitive, accurate and drift free. Some work will have to be done on correctly resolving the three forces and three moments from the measurements. Touch sensors for the fingers will be developed with more spatial resolution. Two devices will be particularly investigated. One, seen in Japan, is an array of metallic buttons buried in some elastic material. The other involves the use of fiber-optic devices. Light traveling down a light pipe is reflected in proportion to the compression of elastic transparent material at its end. This promises to allow a certain amount of force resolution while being very small at the sensing end. Bradford Howland, of Lincoln Lab., is working on such devices.

Software

Our overall approach involves splitting the computation requirements between the mini-robot's processor and a larger remote machine with the ARPA network serving as the communication medium. The high-level knowledge-rich portions of a robot experiment can thus be developed in the friendly environment of the large machine with its greater file system and more powerful languages. Meanwhile the local processor handles straightforward programs which are too data-time and I/O-dependent to work well over the ARPA net.

This gives high priority to the creation of a command interpreter capable of interfacing commands from and information toward the larger machine. Meyer Billmers is in charge of developing the language. It is to accept commands in the form of character strings consisting of a

function name followed by a sequence of numerical or symbolic arguments. This syntax is smoothly compatible with LISP and simplifies interface programming at both ends.

An additional consideration is compatibility with respect to the possible development of a powerful PDP/11 stand-alone LISP system. If such a system were developed, the mini-robot itself could support all of the software development and execution, thereby moving network interfacing to the role of program and picture data exchange.

In many applications, knowledge-based programs determine access points in a small area. Subsequent accesses are likely to be nearby in the picture and therefore in core on the same page. A paging arrangement is designed to recognize this kind of use and make relatively few accesses to the disk.

We therefore plan a software picture dumping system. The system will transfer data from vidicon and solid state arrays into picture files on the local disk. Further interfacing will allow transfer of files to bulk storage on the parent machine and to the ARPA network community at large. This will allow low budget groups to work with expensively procured images without having the physical image device in house.

We plan to pattern the mini-robot picture facility after that already existing on our PDP/10 system. Picture arrays will be stored as collections of subarrays which in turn can be paged in and out of core memory.

Experience shows conclusively that disk stored picture files are essential for scientific vision research. Without such files large complex programs become impossible to debug and two programs can never be compared with any satisfaction.

We have singled out dynamic arm control as a particularly sensitive area in which to begin development of basic user primitives because considerable theoretical work must be done before satisfactory control programs can be written.

Some studies have been made in our group and elsewhere (ref: Gresser, Whitney, Stanford) but problems remain in that the straightforward manipulation of the arm dynamics equations result in solution formulas with far too much computation for real time use. We expect eventually to find efficient symbolic and interpolation/table look-up solutions to such problems.

Richard Waters is making good progress toward a set of arm control programs and basic commands. He hopes to continue during the next year to approach the computational problem with a combination of heuristic and mathematical ideas that already has made considerable progress toward reducing the real time load from impossible to manageable.

Demonstration Problem

We believe that a successful piece of hard-core applied research will be necessary in order to sell the specific idea of the mini- robot laboratory and the general idea of advanced productivity technology. We consequently have embarked on the development necessary to enact the following scenario:

1) A technician specifies an interest in the waveform at a particular pin on a given component, perhaps a DIP integrated circuit.

2) The robot locates the part visually.

3) The robot realizes that the specified pin lies in an awkward place. Wires on the foil side are traced to a more accessible place. The arm clips on a test probe.

4) The technician decides the component is bad.

5) The robot would then clip off the component's connecting lead,

6) Desolder the clipped free leads from the board using a force sensitive tug to pull them out.

7) It would visually inspect the holes to be sure they are free of solder.

8) Next, it would insert the new part, guiding it to the correct position with a combination of visual and tactile feedback, then

9) Solder in the new part, and finally

10) Inspect the newly soldered joints and resolder if necessary.

We should be able to do this around the end of 1975, probably sooner. It should be understood that this demonstration will not be versatile enough, however, to be considered a prototype for a real production repair facility. The mechanical activities should proceed at approximately human speed, limited primarily by the mechanical hardware rather than the computer processing. At that time, we could decide whether a determined effort should be made to make faster, smaller equipment.

At that point or, perhaps somewhat earlier, we should assemble a conference of people concerned with delicate assemblies to decide on

priorities for development of more capable micro-automation equipment. We will attempt to maintain liason with agencies involved in such concerns.

Work Underway

Timothy Finin and Thomas Lozano have already begun a study of circuit board images. Since a skeleton system is just now together, we have begun preliminary studies. Our image dissector is not entirely satisfactory here because it is both insensitive in general and susceptible to damage from the highlights that are common with circuit boards. Progress, nevertheless, has been made:

1) We now have one good printed circuit wire tracker that creates drawings of circuit boards from images. Three alternate algorithms have been blocked out and are being programmed so as to compare their perfomance against the first. Thus some progress has been made in scenario problem 3.

2) We have a program that searches for resistors. The program has a model for resistor reflection characteristics and is not fooled by other components of the same size and shape. We will couple this program together with those derived from our general color studies in order to read color codes. This will allow a user to easily direct the machine's attention to a particular resistor. Since our printed circuit wire tracker is running, we can couple in a facility that will identify components connected to previously located resistors. This addresses scenario problem 2.

3) Using a prototype force sensitive arm-gripper combination, David Silver demonstrated a force-sensing, non-visual program that turns a crank and spins nuts onto bolts. We will generalize this to the problem of inserting component wires into holes and the problem of pulling bad parts loose in desoldering. This has preliminary impact on scenario problems 6 and 8.

The arm control language of Richard Waters will also contribute to all activities requiring arm motion. The concentrated work on the elements of the scenario will follow the completion of the skeletal system assembly with the minimal eye, arm, and software. The entire skeleton system should be in useful operation early in 1974.

PROJECT 2
THE ELECTRONIC CIRCUIT DEBUGGER

DEFINITION: The goal is to develop programs that understand the principles of ordinary electronic circuits well enough to be able to analyse malfunctions in ordinary equipment. Operated in conjunction with the PHYSICAL ELECTRONIC ASSISTANT, such systems could perform routine maintenance, diagnosis, and repairs. They could be equally valuable in checking out manufactured products or servicing in the field.

When combined with the physical system of Project 1, we believe that this project will lead to a useable maintenance and repair capability that, starting in about four years, would demonstrate how to make systems that can repair electronic boards, automotive electrical systems, and other circuits of similar complexity.

MILESTONES:

July 1974: Develop representations for a sample collection of well-understood circuit "blocks" -- amplifiers, detectors, etc., annotated with comments concerning electrical functions and functional roles.

Construct informal trouble-shooting scenarios, for (say) simple transmitter and receiver.

Dec. 1974: Build formal grammars for the structures in phase 1.
"Parse" into understandable parts a simple digital logic circuit.
"Parse" complex schematic diagram of a transmitter into blocks.
Diagnose and explain why simple faults cause failures.
Understand design well enough to plan signal-tracing for particular circuits.
Select good sequence of test-probe points and
Predict some property of wave-forms at those points.

Dec. 1975: Extend analyser ("parser") semantics to deal with detailed functional analysis of symbolic circuit diagram.
Understand design principles well enough to use debugging theory to propose repair.
Perform physical sequence of signal-tracing test-probe operations
Connect to Electronic Repairman (Project 1) in demonstration to locate (visually) broken part and replace.

Dec. 1977: Correlate visually-scanned circuit diagram with physical circuit board, to locate components in symbolic diagram.
Connect to Natural Language system
Connect to Automatic Programming Assistant
Repair real circuit with genuine (field) malfunction.

APPLICATIONS: The choice of specifically electronics-oriented intelligent problem-solving is motivated by our auxiliary concern with MICRO-AUTOMATION. In the electronics field, the alternative of using human skill to make repairs and modifications will gradually become unavailable as assemblies become smaller and more complex.

Most work on "robotics" has been focused on initial assembly of devices. This is very natural, because there are fewer complications in working with perfect, new components. However, we feel that the most valuable applications of intelligent automata will be in the areas of maintenance and repair, and this area is untouched so far. The visual and logical problems are harder, but we should begin to work on them now so that there will not be a very long time lag when the hand-eye hardware becomes adequate for these applications.

PROBLEMS: Understanding complex circuits is not a well-developed formal area. We emphasize this because many readers will recognize that "circuit theory" has been thoroughly formalized! But circuit theory does not mean circuit-understanding. We will first have to develop some "scenarios" describing what happens in some simple analog and digital circuits. This entails representations of circuit causal functions and circuit representations with semantic meaning. To describe the function of a component, we will probably have to talk about "Difference" representations for explaining circuit changes. And on top of these high-level functional semantics we will need a parallel system of "physical semantics" for relating the component descriptions to test "waveforms".

In developing such a system, in which one has to work with three or more different kinds of representations, important technical problems are shared with those of Project 3 (the Programming Assistant) and point not only to the Electronic Repairman application but also to tools for perfecting all sorts of large systems. There is no sharp line between repair and design. Even in simple digital circuits, debugging problems range from simple broken and shorted connections to subtle symptoms of marginal design breakdown -- excessive fan-in and fan-out, imperfect synchronizing provisions, etc.

Many problems in this area are new. Of course, previous "computer problem solving" programs had to be debugged, but this was never treated systematically as a technical problem in itself. Circuits have complex, non-serial causalities and depend on intricate "side-effects" outside the main "signal path". Fortunately, the kind of thinking needed to handle such interactions seem generally similar to the kind needed to understand ordinary programs.

Conventional circuit theory will not occupy center stage. Electronic technicians and servicemen do not analyse whole systems as electric networks (nor would this be feasible even if they knew the appropriate theory). The real problem is to understand the local situation well

enough to represent it by a simplified circuit, and analyse that.

COSTS: The project uses the equipment of Project 1 for physical hardware, and we are already funded for that. However the project has major computational costs as well.

PERSONNEL: G. Sussman, I. Goldstein, Scott Fahlman.
With M. Minsky, S. Papert, P. Winston,
C. Reeve, T. Knight, J. Holloway.

TECHNICAL ASPECTS OF PROJECT 2
THE ELECTRONIC CIRCUIT DEBUGGER

A. INTRODUCTION

DEFINITION: The goal is to develop a system that understands the principles of ordinary electronic circuits well enough to be able to analyze malfunctions in ordinary equipment. We want it to be applicable to a wide variety of devices. Also, one would like to be able to extend it to new classes of problems, without too much effort. Therefore the system must be based on good principles of causal and teleological reasoning. All special knowledge of devices and components should be modular and extensible.

It is not necessary to make the "learning" as easy as it is for a human technician -- to make the system have practical value. An advantage of a computer-based system is that once the skill is learned it can be transferred swiftly to other machines.

Or so it would seem! What computer scientists have learned, however, is that one can rarely "add" two skills together to make one larger, better program! One might even say that a basic problem in A.I. is to find representations for skills in which the interactions due to such merging can be easily isolated and made compatible.

In order to avoid escape into toy problems, we intend to develop this system in several parallel domains of real devices. Eventually these will include analog devices ranging from consumer radio, T.V., and hi-fi through sophisticated communications equipment and digital devices ranging from the pocket calculator through the mini-computer. The underlying reasoning processes should eventually be knowledgeable enough to cover this range and be quickly adaptable to new kinds of components (provided the basic circuit concepts are not changed much).

We expect this kind of system to be useful in augmenting the intellectual powers of (human) maintenance technicians. When combined with the Physical Electronics Repairman (Project 1) it could perform routine maintainance, diagnosis, and repairs by itself. At what level of skill? It is hard to say at this time. Clearly, it will be very hard to approach the general common sense of a skilled technician. On the other side, the machine should be able to exploit special kinds of expertise in using conventional circuit theory, correlation of simultaneous measurements, etc. Of course, we expect all this research to work directly toward improving our position vis-a-vis general knowledge, so perhaps this is too conservative a position.

B. GOALS

PROBLEM: Suppose that we are given an inoperative electronic device of known correct design. Deduce, from its symptoms and from a few well chosen experiments, the cause of difficulty. (You have to choose the experiments, first, by understanding the symptoms!) Determine the action to be taken (e.g., the components to be replaced) to restore the device to working order.

Why is this a good problem?

SCIENTIFIC VALUE: It is particularly important to study qualitative causal and teleological reasoning. This is one of the weakest areas of AI. This task requires sensible and purposeful experimental and exploratory behavior. It attacks head-on the problem of dealing with the unexpected in real world situations.

ECONOMIC VALUE: There is a lack of highly trained technicians to maintain modern complex electronic hardware. If coupled with a robot in the future it could be valuable for repairs in hostile environments (space, undersea etc.). With quickly evolving equipment, as in modern electronics, the problems are unusually severe.

The problem also engages another important modern focus in our work. The repair problem contrasts with the harder kinds of design and puzzle problems because

> In repair, one is usually given a description of how things are supposed to be; how the device should work. One doesn't have to figure that out. Nonetheless, one has to understand the explanation! So the problem meets the condition that before one can create a "designer" or program-writer, one should know how to build a repairman, i.e., an understander - annotator - debugger.

> The program needs less knowledge, in the sense that one can understand how a device works without knowing all the design considerations of how to invent it.

> The apparent solution to the problem seems to fit in well with our current concept of the "frame" or "scenario"-type recognition and explanation theory.

Thus, we need only determinehy the device presented does not operate according to an understood model of its operation.

C. A MODEL-DRIVEN TROUBLE-SHOOTING SCENARIO

Imagine an AM superheterodyne radio receiver -- say one of the standard 5-tube AC-DC circuits that was once the most common of all electronic devices -- with the symptom that loud signals are distorted. Suppose further that the problem is in fact caused by a shorted AVC filter capacitor. How does our repairman determine the cause of failure?

When the problem is posed, our repairman pulls out a fairly abstract model of a superheterodyne radio receiver (See Figure 1. How our repairman assimilated the model is discussed elsewhere). This "general" model actually covers a large class of radio receivers, including most transistor portables as well as the "all-American 5". (It does not, however, cover basically different designs, e.g., the simpler Tuned - Radio - Frequency receiver.) Of course, this is not the only possible model for a superhet; one might conceivably find a quite different way to analyse it into modules. In any case, this model does distinguish submodules, each of which can be further specified as a module with specific ports and some internal structure. We see that the model also distinguishes the basic signal path from control signal paths and power paths. Each submodule is likewise specified. We show two possible converter modules in Figures 2 and 3.. The first one is appropriate to most broadcast receivers, but the second might be found in fancier communications equipment.

There are analogies both with physical scene-analysis and with a "generative grammar" of electronic equipment. The top level of grammar -- like the "kernel sentences" -- are the devices that people use. The lowest level -- the "words" or "terminal symbols" -- are the atomic components: resistor, capacitor, inductor, transistor, etc. This grammar is matched against the schematic diagram of the equipment to be debugged, to establish the correspondence of parts of the model to parts in the diagram. The result of this "parse" is a hierarchically annotated diagram, with the module boundaries laid out and each module (down to the atomic components, if necessary) annotated with its purpose.

It will be interesting to see whether we can usefully model such analyses within the proposed ACTOR system [see Project 3] and obtain an analogy to conventional signal-space "simulation". In the Actor application, the circuit blocks might converse by means of messages describing the "wave-forms"! In conventional simulation, no one has gone outside the basic time-signal space. Each component module (at all levels: amplifier to resistor) can be described extrinsically and intrinsically. Intrinsic descriptions -- that is, looking down the tree -- describe what the module is; examples are:

0.1 ufd. capacitor
2 MegOhm resistor
455 KHz tuned circuit
Fixed-tuned radio-frequency amplifier.

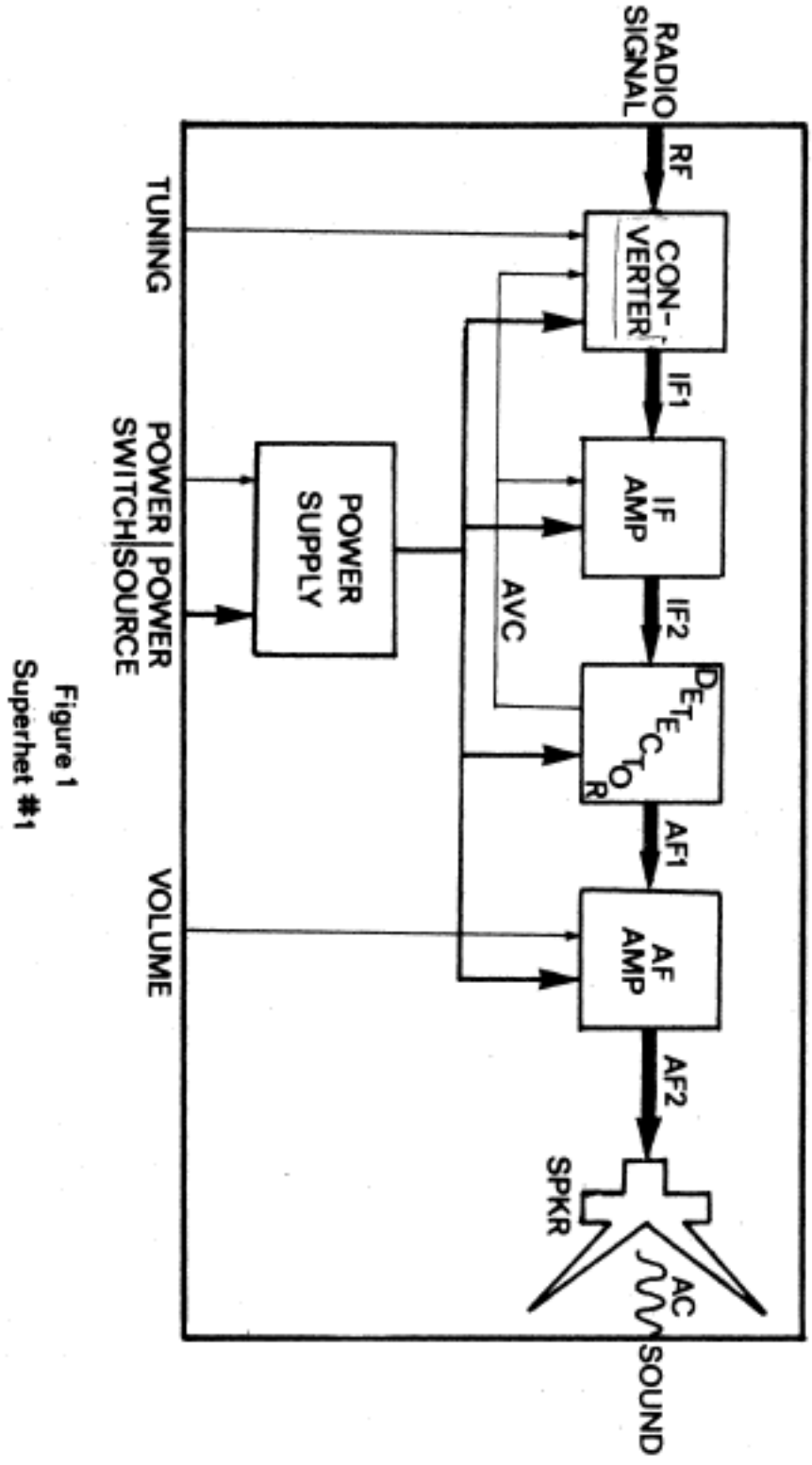


Figure 1
Superhet #1

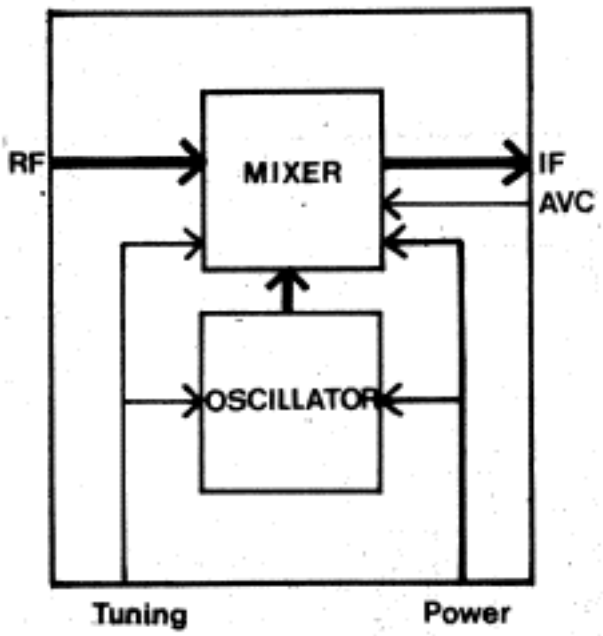


Figure 2
Simple Converter

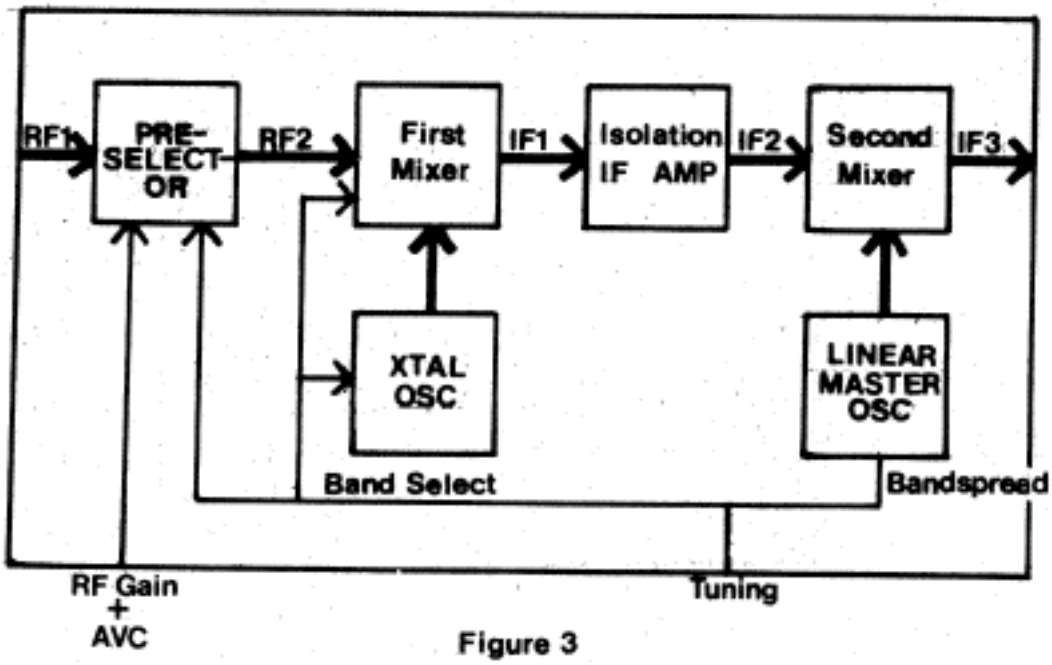


Figure 3

Communications Receiver Converter

(power connections ignored for clarity)

An extrinsic description explains the use or purpose of that module in the overall circuit -- in terms of higher-level nodes of the tree. Corresponding extrinsic descriptions of the same objects are:

cathode bypass capacitor
grid-leak resistor
IF amplifier input tank
IF amplifier.

Usually, there is only one intrinsic description of a particular pile of parts but the extrinsic descriptions depend upon context -- their relation to each other.

A device is broken if the behavior of that device is not as advertised in its intrinsic description. Since the devices we will deal with are correctly designed this means that some subcomponents do not live up to their intrinsic descriptions. The problem is then to determine which atomic components are at fault (without testing them all individually).

Let us return to the given problem and consider the reasoning of our repairman. The perception of distorted sound on loud signals means that some module in the basic signal path is not linear with respect to the audio component of the signal. Where is the distortion introduced? The repairman program instructs us to set up the radio on the workbench under operating conditions. The radio is tuned to the output of an amplitude-modulated signal generator which is adjusted to produce a signal strong enough to cause the problem. The program now traces back the signal path looking for the place where the distortion occurs. It asks us to "scope" the audio output and determine if the distortion is there. We answer yes. It then asks about the output of the detector. The answer is still yes. In fact, we find that the distortion originates in the IF amplifier because its output is distorted but its input is not. Is the problem in the IF amplifier? Let's look at the other inputs (prerequisites) to the IF to check if they are reasonable. The power supplied is OK but we find that the AVC line is at 0 Volts and that it is independent of the RF input (from the signal generator). Something is wrong, then, with the AVC bus. By considering the consequences of the AVC being held at 0 Volts we can see that this could cause the problem. The AVC controls the gain of the converter and the IF amp. At 0 Volts everything is at maximum gain. A strong signal would then be amplified enough to drive the IF amplifier into non-linear operation, hence the distortion.

Now, what is the problem with the AVC bus? Is the problem that the AVC voltage is not being generated at the detector, or is it being bypassed to ground in the converter or IF amplifier? The way to determine this is to disconnect the AVC line from the converter and IF amp and then measure its voltage when isolated. Still zero! Thus the voltage is not being generated. Let's look at the detector in more detail (Figure 4.

Figure 4 matches Figure 5 in the actual circuit). Since we get an audio signal out of the audio filter (with a DC component) the problem must be in the DC filter. But that leaves only two components to test: the 3.3 Megohm resistor and the .1ufd capacitor. We find that the capacitor is shorted, hence the problem is solved!

In the preceding scenario of the operation of a competent repairman we get a glimpse of the general approach. The device is broken if it does not behave as advertised by its intrinsic description. It may not provide the expected output for a specific input. The question is, "How should the correct output be generated?" We then look at the structure of the device, as described by the parse tree to determine what submodules are directly responsible for generating the output. In an AM radio this is the audio amplifier. (Recursing down, if the device is an audio amplifier, its ***main step*** is the output stage. If the output stage is push-pull then there are multiple main steps -- components which contribute directly to the generation of output of the next higher level module.) If the device is correctly designed (as we are assuming) either one of these main steps is incorrectly operating or it is getting bad inputs. We use this idea to trace back along the main signal path for the first place where the signal appears good. The problem is then either in this stage, the auxiliary inputs to this stage, or the interface to the next stage (the output of the current stage may be overloaded). We check each of the possibilities and then, when we have found an inoperative submodule, we recursively apply this analysis until the bad atomic components are isolated.

This is of course a sketchy idea and it must be refined. How does it relate to other kinds of troubleshooting, like the debugging of computer programs? Can these ideas be extended to debugging of design errors as well as component failures? The answers to these questions are basic research goals.

D. MILESTONES

In attacking a problem such as this, it is important to thoroughly understand a variety of instances before designing a "general" method. Thus an important first step is to work out detailed scenarios (far more detailed than the one shown here) for a number of electronic troubleshooting tasks in devices ranging from transistor portable radios to test equipment. Much can be learned by discussion with competent technicians as they perform maintenance in the AI lab. As this evolves, we have to formulate representations of ways to "explain" how the circuits work -- as seen by repair technicians. This is very different from the electric network theories learned in academic electrical engineering courses; it is qualitative common sense.

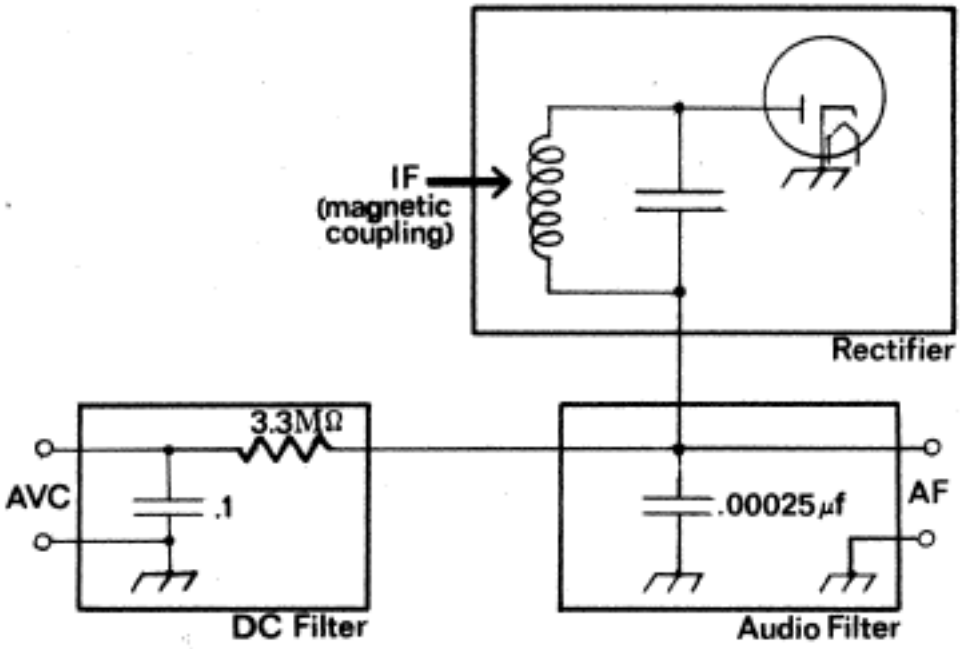


Figure 5

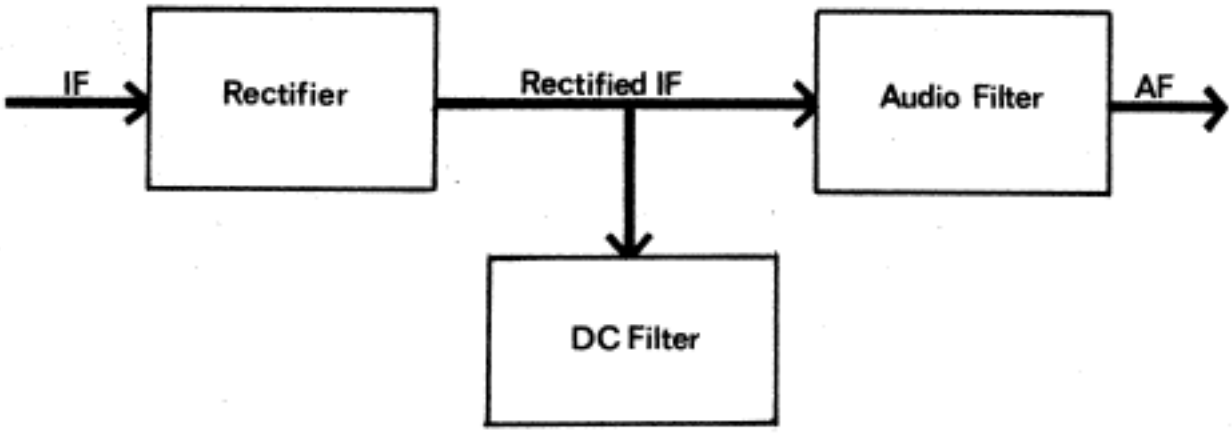


Figure 4

After compiling some set of scenarios, the next step is to generalize the results. At this point, it sould begin to be clear just what are the general techniques of electronic troubleshooting. We now begin to design a program which can parse a schematic diagram with respect to an electronics grammar assigning to each part in the device its purpose in the circuit.

By the end of the first year we should have a comprehensive grammar covering some small class of electronic devices such as commonly available consumer radio receivers. This would include, of course, many of the building blocks of more complex equipment; we would have quite a zoo of oscillators, amplifiers, etc., which are also found in television sets, radars, and communications equipment.

Shortly after the first year we should also have a program capable of parsing a schematic diagram with respect to this grammar and answering questons about the result, such as:

1. What kind of local oscillator is used in the converter of this device? Answer: A Hartley Oscillator.

2. What is the purpose of C10 in this device?
   Answer: It is an emitter bypass capacitor.

By the end of the second year, we should actually have a program running which can use the information provided by the program written in the first year to perform simple troubleshooting tasks similar to the ones collected in the scenarios. We expect that by this time many scenarios will have been developed. During this second year, we expect to increase the size of our grammar to cover many new device types such as transmitters.

We wish we could state at this time how hard it will be to add knowledge about new circuits to the system. But this is a new kind of problem, and no one has had any experience with the pseudo-linguistic problem of meaningful circuit grammars. Within a well-defined category such as communication receivers and transmitters, we would not expect much difficulty in passing from one model to another -- except that each "set" is likely to contain some designer's favorite idiosyncracy, requiring special attention. Designers are prone to obtaining a bias voltage from some unusually stable side-effect. They will insist on using some device's internal resistance as a feedback common element. These conflict with generally followed design heuristics and can cause trouble in diagnosis. In any case, by the end of the first year we should be able to assess the potential difficulties.

PROJECT 3
AUTOMATIC PROGRAMMING, DEBUGGING, and DOCUMENTATION

DEFINITION: We will attempt to make systems to facilitate development of complex computer programs. The goals involve automatic analysis, automatic documentation and debugging schemes. Because the performance will be defined by exterior goals, these systems will have to include advanced learning abilities.

There are really two projects here with two approaches; one is concerned with representing commonsense reasoning knowledge, as exemplified by the theses of Sussman and Goldstein, Project 3.1; the other is concerned with developing a programming formalism and technology, the ACTOR system of Hewitt and his associates, in which implicit and undesirable interactions are (we hope) unlikely to arise accidentally -- Project 3.2.

MILESTONES:

July 1974: Formulation of intention formalisms in several microworlds:
Blocks world debugging: extensions of Sussman's Thesis
Semantics of graphics: extensions of Goldstein's Thesis
Semantics of electric circuit simulation programs

Dec. 1974: Preliminary ACTOR formalism realization
First attempts to apply automatic debugging methods to electronic circuit problems.

Dec. 1975: First Applications to Electronics Diagnosis and Repair programs (Note -- programs, not problems!)
First Natural Language Interfaces
Programming Assistant for the Personal Management Assistant, if that develops into an AI Lab project.

APPLICATIONS: Work on Artificial Intelligence, over the past few years, has created an environment in which we can ask much more from computer programs than was previously reasonable. At least, this is so "in theory". But the much more complex programs required for this are harder to understand, modify, and debug. Thus, this progress will not pay off as well as it should until there are corresponding improvements in:

Responsibility -- Ability of the programs themselves to justify their results; to explain what was done to get the result.

Accountability -- Ability to explain the assumptions the results are based on.

Debuggability -- Programs can provide much better information to make it easier to detect programming mistakes.

Extendability -- Programs should have enough modularity and "transparency" to details to make extensions possible without requiring the programmer to understand all the fine details.

We believe that it is possible to make major advances in these areas now, because of recently developing capabilities to incorporate into programs a new kind of knowledge. To be specific, we mean the KNOWLEDGE ABOUT HOW THE PROGRAM WORKS -- in addition to what programs traditionally contain: procedures designed specifically to solve the target problems.

PROBLEMS: In our approach, the main directions are already outlined in the prototypes presented in the recent theses of Sussman, Goldstein, and in the Actor-Intention paper of Hewitt et al. These all appear to give rather definite and rather promising outlines of what must be done. As the different kinds of principles in each of these are clarified and applied to different microworlds, we can expect many difficulties, interactions, and conflicting priorities. It is too early to guess which problems will be most serious.

PERSONNEL: This area is of concern to two distinct groups.

I. Goldstein	C. Hewitt
G. Sussman	P. Bishop
et al.	et al.

The results so far have captured the imagination of many students, and we expect that quite a few more people will become involved in this project.

COSTS: This "project" requires large computational support. It also involves resources of the sort found in general computer language development, systems programmers and maintenance people, documentation, etc.

The systems will use features related to LISP, MICRO-PLANNER, PLANNER, CONNIVER, LLOGO, and the new ACTOR system.

PROJECT 3.1
PRINCIPLES OF REPAIR AND DEBUGGING

A. INTRODUCTION

What makes an individual a competent repairman? Is there a set of skills which are common to the expert television repairman, computer programmer or digital logic troubleshooter? We assert that there is indeed an important core of knowledge common to these activities. It consists of expertise in debugging, planning, skill acquisition and modular knowledge representation. We propose to develop this area by building "repairmen" for the above domains. We hope to do this in ways that both produce "expert" programs for each area as well as reveal sound principles applicable to other domains.

The Conceptual Framework

To design a programming assistant or a circuit repairman is useful in its own right. The project takes on further interest if one believes that there are important skills and knowledge that are "generally" useful in repairing things. For, if such knowledge could be accumulated in a heuristic program, the design or adaptation of repair-programs for different domains would become progressively easier.

Readers who followed closely the development of AI in its early years -- but not in the last 3 or 4 -- might be inclined to say: "That sounds like the early schemes of separating knowledge about problem-solving in general from knowledge about particular areas of expertize. Didn't it turn out that this line led to mediocre results? And doesn't it turn away from the principles of Heterarchy that you have been advocating, in which different kinds of knowledge have to work together?"

For the last question, the reply is simply that the kinds of knowledge do indeed have to work closely together. But we claim that in the earlier "general" problem solvers inadequate attention was given to questions of repair and debugging -- and that is really why they turned out to be relatively weak!

This debugging knowledge is precisely what is needed to close the gap between planning and execution. Some of the early programs did indeed have some "general" knowledge about how to solve hard problems by breaking them down into simpler sub-problems. But then they faltered. For it is in the nature of a "general" plan that it does not take details into account. Most plans fall down, in fact, when it comes to details. The only hope is to REPAIR the most plausible plan. And this is what we propose to study systematically.

B. THEORY OF DEBUGGING

Debugging is an essential component of reasoning. Plans rarely work the first time; and even when they do, the world may change. Furthermore, debugging is an essential component of self-improvement. We often learn by debugging our own knowledge. One can regard Winston's learning program (AI TR-76) as an error-diagnosis error-correction debugging system.

One might at first believe that the study of planning and debugging is by its nature informal -- mere common sense. In our recent work, however, this area has made progress towards becoming a systematic theoretical subject. Perhaps the best developed illustration of what has happened is exhibited in the recent thesis of Sussman (soon published as AI TR-292), in which we see the start of an effective classification of types of bugs, how to detect them, and how to make plans to repair them. The interested reader should consult the thesis, which should be available by the time this proposal is in circulation.

The following discussion highlights important concepts of this embryonic debugging theory. To illustrate the ideas, we use an example drawn from the thesis of Goldstein (soon published as AI TR-294) which is nearly completed; it is from the world of "turtle" programs.

A "turtle" program is a sequence of commands, to a mobile robot, that cause it to draw a graphic figure on the floor. Alternatively, the commands can refer to a display simulation wherein the turtle behaves like a pen. In the graphics programming language, one can tell the "turtle" to go forward or back ("FORWARD 60" or ""BACK 100") or to turn about its center through some angle ("RIGHT 90" or "LEFT 359"). The turtle primitives are embedded in a high level symbolic language capable of interpretive evaluation, recursion and iteration. For the purposes of this discussion, the programs are given in LOGO syntax. This is for clarity: the programs could equally well be expressed in LISP.

This world of turtle graphics provides a particularly clear distinction between imperative method (local movements of the turtle) and declarative intent (static description of the final picture). However, examples of powerful planning and debugging could just as easily have been drawn from the robot's block world.

Goldstein's thesis is concerned with repairing turtle programs which fail to draw an extended picture. An example of a typical problem is shown in the following way:

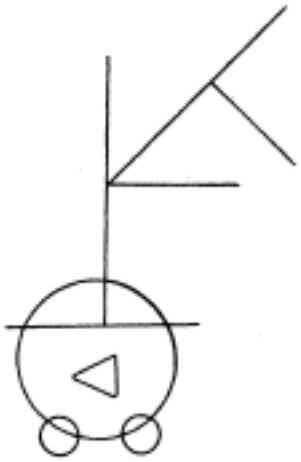


Figure 6. DEBUGGING FACEMAN

To gain insight into the process by which the system debugs programs, we shall draw on the following simpler procedure for a stick figure:

```
TO STICKMAN
10 VEE
20 FORWARD 100
30 VEE
40 FORWARD 100
50 RIGHT 90
60 CIRCLE
END
```

```
TO VEE
10 RIGHT 45
20 BACK 100
30 FORWARD 100
40 LEFT 90
50 BACK 100
60 FORWARD 100
END
```

This program was intended to draw the following picture.

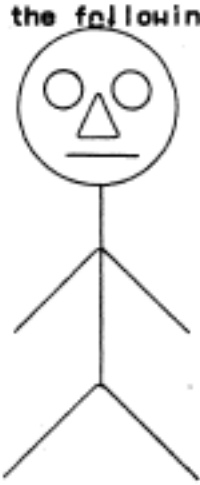


Figure 7. THE INTENDED PICTURE

But the program has a bug. It actually draws:

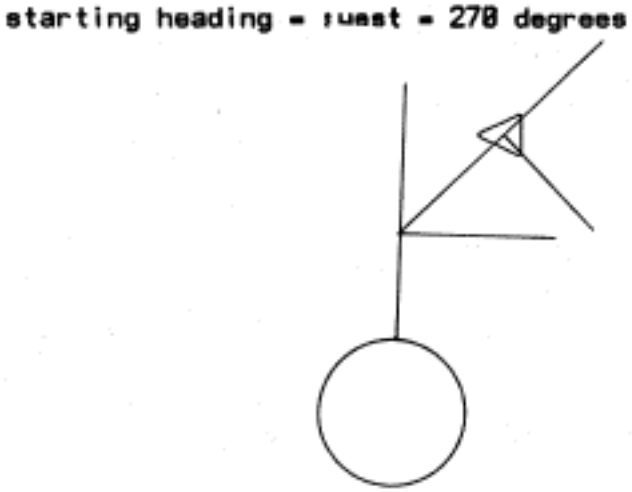


Figure 8. THE BUGGED PICTURE

Debugging requires descriptions of Intentions

To repair the program, a debugging system must initially be provided with a model of the program's intent. Such a model is necessary if the system is even to recognize that the program was unsuccessful.

For our stick figure, it will be sufficient for the "model" to be a set of geometric predicates describing the desired picture.

```
MODEL STICKMAN
      M1 PARTS HEAD BODY ARMS LEGS
      M2 CIRCLE HEAD
      M3 LINE BODY
      M4 VEE ARMS
      M5 VEE LEGS

      M6 CONNECTED HEAD BODY, CONNECTED BODY ARMS,  CONNECTED BODY LEGS

      M7 BELOW LEGS ARMS
      M8 BELOW ARMS HEAD
END
```

This model is underspecified. But it tells enough of the story to recognize that the program fails to accomplish its task. Now the reader is to imagine a program, called INTERPRET, that compares the program with the model:

```
(INTERPRET STICKMAN)
        ;The system is asked to interpret the picture
        ;drawn by the program in terms of the model.
```

```
(*VIOLATIONS    ;the picture fails to satisfy the model because:
  (M5 (NOT (LINE BODY)))           ;The body is not a line.
  (M7 (NOT (BELOW LEGS ARMS)))     ;The legs are not below the arms.
  (M8 (NOT (BELOW ARMS HEAD)))))   ;The arms are not below the head.
```

Different domains will require different model languages. But it will always be necessary to specify the purposes of major parts and the relationships between these parts.

Descriptions of Programs' Intentions: Annotations

Annotation is the process of building a detailed description of intention and effect. The "intentions" describe the "why" or reasons for the object's structure; the "effects" describe a causal chain documenting its performance. Part of the explanation is specific to the particular domain. But important constituents are universal. This would include structuring into main steps and state interfaces, assigning responsibility for parts of the task to parts of the mechanism and documenting interactions accidental to the main purpose.

The "Plan" --- the top level of Annotation

Deriving the plan supplies the "why" of the system, whether program or circuit. The plan describes how the goals of the model are to be accomplished. It is an assignment of local responsibility between statements in the model and modules or interfaces in the code. Deducing the plan is based on general knowledge that processes consist of main steps interleaved with setups, interfaces, and cleanups.

The Idea of "state"

Of course, ANNOTATION requires understanding domain-dependent, non-universal knowledge also. For turtle-program graphics, cognizance of the STATE - position and heading - is important. For circuits, the state will be in terms of electrical primitives -- it might be an instantaneous list of all currents and voltages. But in both cases, analysis proceeds in determining the purpose and success of each stage in terms of its effect on the state.

Our point is the modest one; that there may also be "general" principles about how to handle particular problems. The idea of defining a "state" might be critical for success in many areas, even though each requires different kinds of states, with different kinds of transformation properties.

Rational Design and First-Order Theories

Here is another very general piece of knowledge (or "advice"). Finding a plan is simplified by assuming that the system under study was designed according to the following "rational design" principle.

> The modules interact only over explicit interfaces. There are no accidental side effects. This is the "First-Order" theory; it is surely false in some way, but it can guide the initial attempt at understanding.

Later, as difficulties are isolated, the bug may indeed be discovered as due to a violation of "first-order" principles such as an unexpected interaction between supposedly independent sub-systems.

We cannot give details here, but a pattern-matching procedure based on this principle yields the following plan for the STICKMAN program:

```
TO STICKMAN
10 VEE                  (ACCOMPLISH LEGS)
20 FORWARD 100          (ACCOMPLISH (PART1 BODY))
30 VEE                  (INSERT ARMS BODY)
40 FORWARD 100          (ACCOMPLISH (PART2 BODY))
50 RIGHT 90             (SETUP (STATE HEADING) (FOR HEAD))
60 CIRCLE               (ACCOMPLISH HEAD)
END
```

> NOTE: This plan is the RESULT of the running of a phase of Goldstein's program. Thus, it might be considered to be an Automatic Program Annotator. In the programs of Sussman, "comments" are provided ab initio to the programs to be debugged; in the course of operation more comments are generated. In Hewitt's proposal (Project 3.2) the actor implementation is envisioned to detect incompleteness of the intention structures, and interrogate the programmer. Thus, these are all different threads of the same weave -- all trying to formalize the relations between intentions and the programs written to achieve them.

More on Plans

The plan for STICKMAN is almost linear. The legs, body and head are accomplished one after the next in a natural sequence appropriate to their relative position. However, note that the purpose comment for the arms declares an "insertion". This is a common and useful type of abstract planning structure to which the system is sensitive.

While accomplishing one part of a model, the program may be in the appropriate entry state for another part. In this case, it is natural to "accomplish" the arms in the midst of drawing the body. If the program for the inserted part is state transparent, then the system can expect that the intrusion will cause no harm. Of course, the inserted procedure may interact in unexpected ways with the main program or simply not be state transparent. However, by being sensitive to this abstract plan format, the system is in a position to recognize such bugs and fix them accordingly.

C. DEBUGGING

Isolating a bug is accomplished by finding inconsistencies between intention and effect. Debugging is accomplished by describing the type of discrepancy and making the appropriate patch.

Commom Underlying Causes

The underlying cause of the disaster can often be described with sufficient abstractness to apply to many domains. For example, such causes as CONFLICTING-BROTHER-GOALS, UNEXPECTED-SIDE-EFFECT, MODULE-FAILURE and IMPROPER-INTERFACE are universal causes of failure. (NOTE: these terms mean pretty much what they seem to mean. For details, they are defined precisely in the Goldstein and Sussman theses.)

Common Methods of Repair

Similarly, there are important common elements in strategies for repairing bugs. CONFLICTING-BROTHER-GOALS can sometimes be fixed simply by reordering. Interface problems are simplified by maximizing state transparency. MODULE-FAILURE, whether of a sub-procedure, tube or chip, suggests the obvious correction of recursing the system and repairing (or replacing) the module.

Ordering the Attack

Multiple bugs can be difficult to correct. Hence, guidelines are necessary in the order of debugging. A heuristic of wide applicability is to debug the parts before attempting to correct the relations between them. For the STICKMAN, this would mean debugging the body before worrying about the "above" relations. The basis of this ordering is the standard scientific notion of beginning with a first-order linear theory of a problem before attempting a second-order explanation which handles interactions.

The plan indicates which steps are responsible for the body. Domain dependent knowledge defines a line as composed of colinear vectors, where FORWARD's are understood to produce vectors. "Colinear" is a constraint on the direction of vectors. Now, domain independent knowledge is applied. An abstract pattern-match of the process is made to discover the state-interfaces between the main steps responsible for the body. In this case, the interface is line 30 wherein the arms are drawn. The bug is classified as an UNEXPECTED-SIDE-EFFECT of VEE. The fix is to insert a patch returning the turtle to the correct heading:

INSERT LINE 35 "RIGHT 45"

Aesthetic Interrupt

An aesthetic interrupt occurs when the criteria of good design are violated. These criteria are based on considerations of efficiency, clarity and resistance to future bugs.

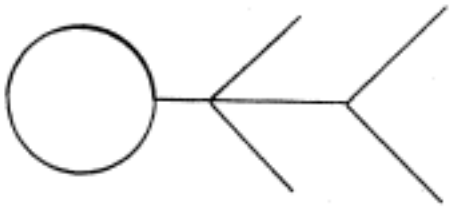
> In this example, the sub-procedure VEE already returns the turtle to its entry position. Inserting the interface in line 35 of STICKMAN requires that "heading" also be restored. The result is an abstract pattern match on procedures in which "state-transparency" is required. The result of this match is to remove the "RIGHT 45" from STICKMAN and, instead, insert it as the final line of VEE. This restores the original heading and VEE becomes fully transparent with respect to both position and direction.

INSERT LINE 65 OF VEE "RIGHT 45"

State-transparency is an important characteristic for achieving modularity. Thus, this edit serves to make the VEE sub-procedure simpler to use in future applications.

Recursion

Having fixed the "body", the system now recurses and debugs the remaining difficulties. With VEE edited, the STICKMAN now has the appearance:



The failure of the above relations can be classified as having one of two possible underlying causes. The first is that the interfaces between the parts are in error. The second is that the global state upon entry is not as expected. Under the former assumption, each interface must be debugged. Under the latter, only one change need be made.

Minimal Change

A repairman should make minimal changes to the system. Its goal is to fix the system, not redesign it. More important, it does not fully know the designer's intent. Hence, it should be hesitant to make major revisions in his plan. Thus, a single edit to the entry interface is clearly preferable to many edits to internal interfaces.

The resulting change to STICKMAN is:

```
INSERT  LINE 5 OF STICKMAN "LEFT 90"
        EXPECTATION (ENTRY STICKMAN) (HEADING = 270)
```

Expectations

An important side effect of this edit is the insertion of an expectation. The expectation checks the entry state to STICKMAN. In the event that it is not 90 degrees, the system is immediately cognizant of an anomaly. It is not necessary for it to repeat again the entire analysis it first performed. It learns from experience by adding commentary to the user's code.

D. KNOWLEDGE -- SPECIAL AND GENERAL

Specialized knowledge structures for electronic circuitry, programming, and digital logic will be required to interact with the basic repairman module. This will force the system to deal with basic issues in the management and use of large collections of knowledge. Several recent ideas in Artificial Intelligence make this difficult task seem

manageable.

Demons

The first is the use of demons. Old-fashioned programs required an explicit control structure, dispatching in sequence to a prepared list of sub-procedures. This becomes inadequate when the number of situations that the system may encounter grows large. Demons are programs that are automatically activated when a datum or request matching their pattern becomes current.

Procedural Knowledge

A second tool is the representation of knowledge as procedures. Repair know-how is not a collection of facts but a set of directions. A uniform reasoning program is too inefficient. It does not exhibit skill or expertise. Demons can be procedures of arbitrary complexity.

Virtual Databases

Some knowledge is most clearly thought of as facts; for example, the state of a circuit or computer process. This may, however, be too burdensome in terms of space. The solution is the use of a "virtual" data base. Most of the state is not of interest and is, hence, not ordinarily computed. The repairman, however, never knows this. Upon request, invisible demons compute and assert the needed data. Hence, space is used only when needed but redundant computation is avoided. This management system provides important conceptual clarity.

For turtle programs, this structure is used for asking geometric questions of the picture. The asker expects to find the answer in the data base. It may well be there explicitly as a result of analyzing the performance of the program in careful mode. This would include statements of connectivity between sequentially drawn vectors. However, the answer may not be present. A "global" connection due to the turtle crossing a previously drawn vector is not easy to notice while running the program. Upon asking for such information, however, demons are activated which compute the answer and place it in the data base.

The same structure is found in BLOCKS-WORLD programs (see Fahlman's thesis, AI TR-283) where it is even more time consuming to recompute three-dimensional geometric predicates.

Hierarchy

Representing CONFLICTING-BROTHER-GOALS and its associated patches abstractly allows the same knowledge to be applied to many domains. Programs or circuits in conflict for the same resources can be handled by similar plans. Hence, an important epistemological goal of this research is to represent knowledge hierarchically in increasing levels of abstraction.

NOTE: This does not conflict with the notion of a heterarchical analysis. Knowledge at different levels must communicate flexibly. A linear flow of control is not adequate.

The belief that basic debugging, planning and learning skills can be abstracted to a general but powerful form is encouraged by the success with which the LOGO project has taught such general skills to children.

E. LEARNING

The design of a repairman would not be satisfactory without a non-trivial learning component. This is required not only by the desire to see the system generally applicable to new domains but also to facilitate its development even for the specific mini-worlds chosen. Large systems that exhibit no learning are extremely burdensome to program. It is difficult to predict which lines of research will contribute most to this goal. However, here are several promising ideas under development.

Declarative Programming -- "Building Programs From Advice"

Declarative programming is an important type of learning which no systems currently exhibit. This is the specification of the task or edit via simple declarative advice. The system is responsible for making the appropriate compilation or modification of its own code.

This style of programming is important for several reasons. For one, it allows the system to exercise (and the creators to judge) its planning and debugging skills. For another, it is easier for the system to debug itself, since in expanding declaratives into code, the system can fully comment the intended purposes. But the most essential reason is that without such capability, only a person familiar with the entire system could possibly make any improvements.

An essential ingredient to support such capability is debugging skill. The first expansion of declaratives into code may well have bugs. All of the contingencies may not have been considered. Declaratives do not

specify interactions. But if the system is capable of debugging, then unforeseen difficulties can be fixed when encountered.

In the above STICKMAN example, declarative programming is illustrated by the fashion in which the system fills in the details, in the form of a "plan", to supplement the program-independent specification of the model. Indeed, it is clear that such a system, with minor extensions, is capable of writing turtle programs given only an initial model of intent.

Advice Taking

Even if the system is unable to debug itself, it understands a language, i.e. a set of concepts, in which it can be given advice about its own processes. This set of concepts is simply the ideas about control, planning, types of bugs, and methods of solution that it must know anyway to debug programs. Thus, if the turtle monitor is unable to discover the plan from the model and program, it can ask the user for statements of purpose. The concept of "purpose" is part of its understanding.

Skill Acquisition -- Improving from Example

Improving from examples is a second important characteristic. Such improvement can be categorized as debugging oneself in the light of new knowledge. Again, this capability is important if the repairman is to be readily extendable to new domains.

Dead-End Analysis

A common element to recent work in Artificial Intelligence is the ability of a system to learn from errors. The metaphor of a search for the right path is replaced by an exploration, which is sensitive to learning from errors. Debugging systems have this characteristic since their very purpose is to correct an unsuccessful system. But Fahlman's "BUILD" program" also shares this characteristic. A plan to build a tower should not be discarded if the tower falls. Rather the best option may be to fix it by using scaffolds or counter-weights.

Sussman's program is sufficiently sensitive to dead-ends that it compiles critiques to prevent repetition of unfruitful lines of development.

Initially, Sussman's HACKER knows about planning and debugging but not about "gravity". It does not know that towers must be built from the bottom up. However, HACKER, in examining bugs due to towers falling, learns that building upwards is essential. Interestingly, while the

program learns this initially for towers of two blocks, it immediately generalizes to towers of any size. It is comfortable with the concepts of "input" and "recursion".

Exploration, experimentation and improvement from mistakes will be the guidelines for a repairman rather than simple heuristic search.

F. PLANNING

Planning, whether to build a tower or to fix a program, often procedes in a top-down fashion. The general classification of the problem is made. The top level set of goals to solve the problem are generated. Then the system examines each goal and decides how to satisfy it in turn. Difficulties occur when a sub-goal proves recalcitrant. The plan must be debugged. A system that has expertise in repair is in a position to perform such a recursion, applying its skill to its own plan. Thus, the repairman is expected to exhibit powerful planning capabilites, which simpler non-self-critical programs intrinsically lack.

Fahlman has designed an expert BUILDER for the BLOCKS-WORLD. This program is able to employ scaffolds and counterweights to aid in its construction of towers, arches and other structures. The important characteristic is that it is capable of first deriving a simple plan and later, when faced with "bugs", improving and augmenting it. The program is less an expert on programs and more a specialist in the physics of construction than Sussman's HACKER program. However, both projects reveal how expertise in planning and debugging greatly increases the power of a problem-solver.

G. BOOTSTRAPPING

The specialized expert, though the possessor of detailed and difficult knowledge, is often incapable of changing domains. He lacks basic problem solving expertise in planning and debugging, expertise that has been structured abstractly so as to be generally applicable. The possibility that by abstracting the skills of debugging, bootstrapping will be possible in the design of a general repairman is fascinating.

PROJECT 3.2
AN AUTOMATIC PROGRAMMING APPRENTICE
For Software Production and Validation

Carl Hewitt, Peter Bishop, Irene Greif,
Brian Smith, Todd Matson, Richard Steiger

A. INTRODUCTION

This proposal discusses the development of a programming system which understands what it is doing. We mean this in a surprisingly literal sense: we propose to develop a system that understands not only the steps of a program but what they are supposed to do and why. Such a system will help in construction of more intelligent and powerful programs, will also serve in further understanding of reasoning, knowledge embedding, and intelligence in programming. Although a highly intelligent system cannot be built today, it is a plausible long-term goal because we can begin now with presently understood concepts.

Recent research on Artificial Intelligence has given us the tools and concepts for this, including:

- Goal Oriented Formalisms
- Natural Language Semantics
- Source Language Interactive Debugging Systems
- Ability to confirm that a program satisfies its intentions
- Ability to make simple patches to procedures that fail to satisfy their intentions

These ideas need synthesizing into an integrated system. At our Laboratory the PLANNER project [PLANNER Technical Report 3: "A Universal Modular ACTOR Formalism for Artificial Intelligence"] has recently developed a coherent semantics which integrates many of these abilities.

The apprentice we propose is conceived as an initially "diligent but moderately stupid" apprentice to help in writing large programs. It will take the form of an integrated editor-interpeter-debugger-problem solver. Existing interactive debuggers (with the exception of Teitelman's PILOT system, developed in our Laboratory) can only deal with syntactic aspects of programs. They can catch misspelled words, correct the format of functional arguments, keep track of the use of functions, balance parentheses, etc. but can do little with semantic content. Our initial goal is to be able to interactively answer such questions as "Who called this function with a negative number?" or "Who can come here with a list?". A subsequent stage will include a semantic model of the programming domain as well.

B. AN INITIAL APPLICATION DOMAIN

We are developing a language-system based on the idea of "ACTORS". ACTORS are universal modular computing units that have a number of important advantages over conventional ways of organizing programs and data. They communicate only through sending messages in specific ways. This is a sharp restriction -- compared to the free-for-all of interactions permitted in ordinary programming systems. Our thesis is that we gain much and lose little by using them. The implementation of actors on a conventional computer is a good initial domain in which to try out our ideas about automatic programming. Such a system will have to do the following:

> Trace implications of proposed changes in a configuration of actors.
>
> Determine the behavioral properties of other actors that a configuration of actors relies on.
>
> Trace the dependencies of some aspect of the behavior of a configuration of actors.

Our apprentice must understand both programming and the domain dependent knowledge for which the program is being written. First we must teach our apprentice about programming in the area of ACTOR implementation. (The implementation of actors on a conventional computer is a large problem, not a toy one.) We have a number of experts on this domain who are very interested in formalizing and extending their knowledge. These experts are good programmers and have the time, motivation, and ability to embed their knowledge and intentions in the formalism. Once the experts put in some of their intentions, they find that they have to put in a great deal more knowledge to convince the auditor of the consistency of their intentions and procedures. In this way we hope to make explicit ***all*** the ***behavioral assumptions*** that our implementation is relying upon.

This will require new representations for description of the system. Fortunately, this domain is "closed" in the sense that the questions that can reasonably be asked do not lead to a vast body of other knowledge which would have to be formalized as well. It is possible to start with a small superficial model of actors and build up incrementally. The task is simplified by excluding such complicated software engineering practices as the use of "go-tos", interrupts, or semaphores.

C. KNOWLEDGE BASED PROGRAMMING

"Knowledge based programming" is writing programs that have direct access to a substantial knowledge base in the application area for which the programs are intended. The actor formalism will aid knowledge-based programming in the following ways:

> ***PROCEDURAL EMBEDDING*** of ***KNOWLEDGE*** has gained new impetus in recent years from the development of PLANNER-like problem solving systems. Procedural knowledge enables us to put knowledge into a computer in a form such that it can be effectively used as intended.
>
> ***TRACING BEHAVIORAL DEPENDENCIES*** can be done by analyzing the intentions of programs and keeping track of each intention that is relied on in another procedure.
>
> ***SUBSTANTIATING*** that ***ACTORS SATISFY*** their ***INTENTIONS*** can be done by binding procedures to their intentions and meta-evaluating the programs. Meta-evaluation is the process of reading the program abstractly to make sure that all of the intentions of the actors that it calls are satisfied.
>
> "Testing examples shows the presence, not the absence, of bugs."

In particular, our apprentice must be able to understand the following:

> The intentions that we express for our programs.
>
> The justification we give for believing that these intentions are satisfied by the programs.
>
> The behavior that our programs will exhibit if executed.

Our apprentice digests this information as it is incrementally presented. It sometimes asks questions when it doesn't understand. When we believe that we have finished writing the code and intentions for a configuration of actors we can ask our apprentice to execute it "abstractly" to see if it has a chance to work in general. Where it can't understand the code it will try to give us high level feedback pointing to a place in the code and describing the nature of its difficulty.

As a theoretical basis, we plan to try to use Scott's lattice theory approach to recursion. In this theory, the meaning of a program can be understood to be the least fixed point of a functional on a lattice of functions. Due to specific properties of these functionals, this fixed point can be constructed in a particularly nice manner. Induction over the stages in this construction is then the natural means of confirming facts about the fixed point and, thus, about the behavior of the program.

Irene Greif is investigating how this interpretation of recursive programs can be extended to "actor programs" or systems of actors. There is a natural minimal behavioral fixed point to any actor definition, but it is only understood informally. An extension of Scott logic to a theory of actors would not only formalize these concepts for actors but would show, we hope, the value of taking a Scott logic viewpoint for more general systems. Actors can be used to define indeterminate systems as well as determinate ones and at present it is not known whether these systems can be studied productively within the lattice theory framework.

One problem is to identify the lattice over which we will be defining actors. Since behavior may be time-dependent, it may be necessary to try to account not only for input-output pairs but also for some relation over time. This will be a departure from the systems previously handled in this logic.

This extension will also yield rules of deduction for the new logic. The rules of deduction to establish that actors satisfy their intentions essentially take the form of a high level interpreter for abstractly evaluating the program in the context of its intentions. This process [called META-EVALUATION] can be justified by a form of induction. To substantiate a property of the behavior of an actor system, some form of induction will be needed. At present, actor induction for an actor configuration with audience E can be tentatively described in the following manner:

> If the actors in the audience E satisfy the intentions of the actors to which they send messages,
>
> And, if when any actor's intentions are satisfied, so are those of all actors sent messages by it,
>
> Then the intentions of all actions caused by E are satisfied (i.e. the system behaves correctly).

Greif is investigating to see if this induction rule is related to the minimal behavioral fixed point in a natural way.

This work should help us formulate precisely what a program does as opposed to how it does it. It gives us a natural, intuitive way to establish that a program does what is intended. And it provides a more solid foundation on which to build an apprentice system.

A Simple Example

Consider the problem of writing a program to shift the gears of a truck with a manual transmission. We apologize for the necessity for

introducing new syntax but the following concepts are crucial to the discussion which follows:

[x <=(=> ybody)]

is actor syntax which at a rough intuitive level means: define an actor *x* which, when it is called with an argument (to which *y* is bound) executes *body*.

(rules x (=> y1 body1)(=> y2 body2)...)
roughly means: take *x*, and if it matches *y1*, execute *body1*; otherwise if it matches *y2*, execute *body2*, etc...

[x <=(intention [n] i1 definition i2)]
is an elaboration of 1, meaning that when *x* is called with *n*, then *i1* is the intention of the incoming call and *i2* is the intention when *x* calls out again.

Our first try at a shift procedure might be:

Primitive-shift-to: when called with a target gear checks to see if it is 1, 2, 3, or 4 and calls the appropriate select: upper-left, lower-left, upper-right, or lower-right respectively.

```
[primitive-shift-to <=
   (=> target-gear  (rules target-gear
 (=> 1 (select-upper-left))  (=> 2(select-lower-left))
 (=> 3(select-upper-right))  (=> 4(select-lower-right))))]
```

Now we consider the various select routines and their intentions. Each of the select functions has an incoming intention that the clutch be disengaged. Furthermore each of them has code (delimited by *) to do the selecting. When a selector calls out, we fully intend for the truck to be in the gear appropriate to that selection.

```
[select-upper-left <=
(intention []
(clutch disengaged)
*code-for-select-upper-left*
(in-gear 1))]

[select-lower-left <=
(intention []
(clutch disengaged)
*code-for-select-upper-right*
(in-gear 2))]

[select-upper-right <=
(intention []
(clutch disengaged)
*code-for-select-upper-right*
(in-gear 3))]

[select-lower-right <=
(intention []
(clutch disengaged)
*code-for-select-lower-right*
(in-gear 4))]
```

Our apprentice notices that for each one there is a physical constraint that the clutch must be disengaged before shifting. He queries us about this and so we decide to modify the function PRIMITIVE-SHIFT-TO to first disengage the clutch.

```
[primitive-shift-to <=
    (=> [target-gear]
        (disengage clutch)
        (rules target-gear
 (=> 1 (select-upper-left))     (=> 2 (select-lower-left))
 (=> 3 (select-upper-right))    (=> 4 (select-lower-right)))
 (engage clutch))]
```

Now the code for primitive-shift-to is to first disengage the clutch, then do the selecting as before, and finally engage the clutch.

We also write functions to disengage and engage the clutch.

```
[disengage <=
    (intention [clutch]
        (clutch engaged)
        *code-for-disengage*
        (clutch disengaged))]
```

```
[engage <=
    (intention [clutch]
        (clutch disengaged)
        *code-for-engage*
        (clutch engaged))]
```

Now our apprentice is mollified. However, the engineers dealing with the transmission come to us with some additional constraints. For example to select third gear the constraints are now that the clutch must be disengaged and the truck must be in either second or fourth gear. The other constraints are similar.

```
[select-upper-right <= (intention
 (and (clutch disengaged) (or(in-gear 2)(in-gear 4)))
 *code-for-select-upper-right* (in-gear 3))]

[select-upper-left <=  (intention
 (and  (clutch disengaged) (stopped))
 *code-for-select-upper-left* (in-gear 1))]

[select-lower-right <= (intention
 (and  (clutch disengaged) (in-gear 3))
 *code-for-select-lower-right* (in-gear 4))]

[select-lower-left <= (intention
 (and (clutch disengaged)(or (in-gear 1) (in-gear 3)))
 *code-for-select-upper-right* (in-gear 2))]
```

The new requirements say that (temporarily at least) the truck has to be stopped to shift into gear 1 and no gears can be skipped in shifting while running. (Note: you can shift directly from any gear to first if the truck is stopped.) So we have to write some new procedures to meet these new intentions.

SHIFT-to: when called with a target gear considers, in order, the following rules for the target gear:

> If it is first gear, then do a primitive-shift-to first gear.
>
> If it is either one greater than the current gear or one less than the current gear then do a primitive-shift-to the target gear.
>
> If it is greater than the current gear then shift-to one less than the target gear and then primitive-shift-to the target gear.
>
> If it is less than the current gear then shift-to one greater than the target gear and then primitive-shift-to the target gear.

```
[shift-to <=  (=> target-gear   (rules target-gear
            (=> 1  (primitive-shift-to 1))
            (=> (either (current-gear + 1) (current-gear - 1))
                (primitive-shift-to target-gear))
            (=> (greater (current-gear)) (shift-to (target-gear - 1))
                (primitive-shift-to target-gear))
            (=> (less (current-gear))  (shift-to (target-gear + 1))
                (primitive-shift-to target-gear)))]
```

We ask our apprentice to meta-evaluate our program. It thinks for a while and sees two problems:

> It can only shift to gear 1 if the truck is stopped.
>
> It should not be asked to shift to the gear that it already is in. [The procedure shift-to does not work if it is asked to shift to the current gear.]

We decide to give the following intention to SHIFT-TO: If the target-gear is first gear then the truck must be stopped; otherwise the target-gear must be 2, 3, or 4 and not be the current gear.

```
[shift-to <=  (intention target-gear  (rules target-gear
            (=> 1  (stopped))
            (=> (or 2 3 4)  (target-gear /=  current-gear))
            (else  (not-applicable)))
         *code-for-repeatedly-shift-to*
        (in-gear target-gear))]
```

To summarize we have used intentions in the following somewhat distinct ways:

To specify ***what*** the actor is supposed to do as opposed to ***how*** to do it.

As a ***contract*** that the actor has with its external environment. How it carries the contract is its own business.

As a formal statement of the ***conditions*** under which the actor will fullfill its contract.

The above example does not deal with all of the computational issues that our apprentice will be faced with. For example it does not have sophisticated data structures and has no concurrency or parallelism. Consider an on-line data base for an air traffic controller. We shall suppose that the data-base contains the position and amount of fuel left for each plane that is currently airborn. Various processes will want to read from and write into this data base. It is important that a process not get inconsistent information when it reads out the position and fuel of a plane. This might happen if another process is concurrently updating the position and fuel of a plane. Inconsistent information might result in a plane crash because the controller makes its decisions on the basis of the information it reads in the data base. So we need to put a scheduler in front of the data base to allow only one process to write in it at once. The actor formalism enables us to deal with problems like this in a straightforward way.

Peter Bishop is investigating the feasibility of an actor machine. There are a number of unsolved problems both at the level of architectural design and efficiency.

For example, ***protection*** is an important issue that must be faced by next generation systems. Actors provide a degree of intrinsic protection. There is no way to coerce an actor into doing anything that it doesn't want to do. Thus, at a certain level actors can be passed around quite freely since they will only work for authorized users. Futhermore an actor only knows about the actors that it has been sent as messages. By these means it appears that actors can implement all of the proposals for protection mechanisms that have thus far been published. More work is necessary before we will know how intrinsic protection is best utilized. There remain some important problems in protection involving intent and trust. We are currently considering ways in which hardware can be further developed to address the problems. Another important issue is ***retention*** of storage. Current garbage collection techniques are not very efficient if the amount of storage retained is very much larger that the amount of fast random access memory on the machine. Knowledge based systems will require a vast amount of on-line storage. We will be investigating techniques for making garbage collecton feasible or unnecessary under such circumstances.

D. PLANNER PROGRESS

This section gives a few more details about features of the proposed new ACTOR-based PLANNER system.

The PLANNER project is continuing research in natural and effective means for embedding knowledge in procedures. In the course of this work we have succeeded in unifying the formalism around ***one*** fundamental concept: the ACTOR. Intuitively, an ACTOR is an active agent which plays a role on cue according to a script. Data structures, functions, semaphores, monitors, ports, descriptions, Quillian nets, logical formulae, numbers, identifiers, demons, processes, contexts, and data-bases can all be shown to be special cases of actors. Our formalism shows how all of these modes of behavior can be defined in terms of ***one*** kind of behavior: ***sending messages to actors.*** An actor is always invoked uniformly in exactly the same way regardless of whether it behaves as a recursive function, data structure, or process.

The unification and simplification of the formalisms for the procedural embedding of knowledge has many benefits:

INTENTIONS: The confirmation of properties of procedures is made easier and more uniform. Every actor has an INTENTION which checks that the prerequisites and the context of the actor being sent the message are satisfied. By a ***SIMPLE BUG*** we mean an actor which does not satisfy its intention. We would like to ***eliminate*** simple debugging of actors by the META-EVALUATION of actors to show that they satisfy their intentions. To do this we have a proof-method called ACTOR-INDUCTION. Computational induction [Manna], structural induction [Burstall], and Peano induction are all special cases of ACTOR induction. Actor based intentions have the following advantages:

The intention is ***decoupled*** from the actors it describes.

Intentions of ***concurrent*** actions are more easily disentangled.

We can more elegantly write intentions for ***dialogues*** between actors.

The intentions are written in the ***same*** formalism as the procedures they describe. Thus for example intentions can have intentions.

Because protection is an ***intrinsic*** property of actors, we hope to be able to deal with protection issues in the same straightforward manner as more conventional intentions.

Intentions of data structures are handled by the ***same*** machinery as for all other actors.

COMPARATIVE SCHEMATOLOGY: The theory of comparative power of control structures is extended and unified. The following hierarchy of control structures can be explicated by incrementally increasing the power of the actor message sending primitive:

iterative-->recursive-->backtrack-->determinate-->universal

EDUCATION: The model is sufficiently natural and simple that it can be made the conceptual basis of the model of computation for students. In particular it can be used as the conceptual model for a generalization of Seymour Papert's "little man" model of LOGO. The model becomes a cooperating society of "little men" each of whom can address others with whom it is acquainted and politely request that some task be performed.

EXTENDABILITY: The model provides for only one extension mechanism: creating new actors. However this mechanism is sufficient to obtain any semantic extension that might be desired.

PRIVACY AND ***PROTECTION:*** Actors enable us to define effective and efficient protection schemes. Ordinary protection falls out as an efficient intrinsic property of the model. The protection is based on the concept of "use". Actors can be freely passed out since they will work only for actors which have the authority to use them. Mutually suspicious "memoryless" subsystems are easily and efficiently implemented. ACTORS are at least as powerful a protection mechanism as domains [Schroeder, Needham, etc.], access control lists [MULTICS], objects [Wulf 1972] and capabilities [Dennis, Plummer, Lampson]. Because actors are locally computationally universal and cannot be coerced, there is reason to believe that they are a ***universal protection mechanism*** in the sense that all other protection mechanisms can be efficiently defined using actors. The most important issues in privacy and protection that remain unsolved are those involving intent and trust. We are currently considering ways in which our model can be futher developed to address these problems.

SYNCHRONIZATION: It provides at least as powerful a synchronization mechanism as the multiple semaphore P operation with no busy waiting and guaranteed first in first out discipline on each resource. A synchronization actor is easier to use and substantiate than a multiple semaphor [Dijkstra 1971] since they are directly tied to the control-data flow.

SIMULTANEOUS GOALS: The synchronization problem is actually a special case of the simultaneous goal problem. Each resource which is seized is the achievement and maintenance of one of a number of simultaneous goals. Recently Suesman has extended the previous theory of goal protection by making the protection guardians into a list of predicates which must be evaluated every time anything changes. We have

generalized protection in our model by endowing each actor with a scheduler and an intention. We thus retain the advantages of local intentional semantics. A scheduler actor allows us to program ***EXCUSES*** for violation in case of need and to allow ***NEGOTIATION*** and re-negotiation between the actor which seeks to seize another and its scheduler. Richard Waldinger has pointed out that the task of sorting three numbers is a very elegant simple example illustrating the utility of incorporating these kinds of excuses for violating protection.

RESOURCE ALLOCATION: Each actor has a banker who can keep track of the resources used by the actors that are financed by the banker.

STRUCTURING: The actor point of view raises some interesting questions concerning the structure of programming.

> ***STRUCTURED PROGRAMS:*** We maintain that actor communication is well-structured. Having no "go-to," interrupt, semaphore, or other constructs, they do not violate "the letter of the law". Some readers will probably feel that some actors exhibit "undisciplined" control flow. These distinctions can be formalized through the mathematical discipline of comparative schematology [Paterson and Hewitt].
>
> ***STRUCTURED PROGRAMMING:*** Some authors have advocated top down programming. We find that our own programming style can be more accurately described as "middle out". We typically start with specifications for a large task which we would like to program. We refine these specifications, attempting to create a program as rapidly as possible. This initial attempt to meet the specifications has the effect of causing us to change the specificatons in two ways:
>
> > 1: More specifications [features which we originally did not realize were important] are added to the definition of the task.
> >
> > 2: The specifications are generalized and combined to produce a task that is easier to implement and more suited to our real needs.

IMPLEMENTATION: Actors provide a very flexible implementation language. In fact we are carrying out the implementation ***entirely*** in the formalism itself. By so doing we obtain an implementation that is efficient and has an effective model of itself. The efficiency is gained by not having to incur the interpretive overhead of embedding the implementation in some other formalism. The model enables the formalism to answer questions about itself and to draw conclusions as to the impact of proposed changes in the implementation.

ARCHITECTURE: Actors can be made the basis of the architecture of a computer which means that all the benefits listed above can be enforced

and made efficient. Programs written for the machine are guaranteed to be syntactically properly nested. The basic unit of execution on an actor machine is sending a message much in the same way that the basic unit of execution on present day machines is an instruction. On a current generation machine, in order to do an addition, an "add" instruction must be executed; so on an actor machine a hardware actor must be sent the operands to be added. There are no goto, semaphore, interrupt, etc. instructions on an ACTOR machine. An ACTOR machine can be built using the current hardware technology that is competitive with current generation machines.

Hierarchies

The model provides for the following orthogonal hierarchies:

SCHEDULING: Every actor has a scheduler which determines when the actor actually acts after it is sent a message. The scheduler handles problems of synchronization. Another job of the scheduler [Rulifson] is to try to cause actors to act in an order such that their intentions will be satisfied.

INTENTIONS: Every actor has an intention which makes certain that the prerequisites and context of the actor being sent the message are satisfied. Intentions provide a certain amount of redundancy in the specification of what is supposed to happen.

MONITORING: Every actor can have monitors which look over each message sent to the actor.

BINDING: Every actor can have a procedure for looking up the values of names that occur within it.

RESOURCE MANAGEMENT: Every actor can have a banker which monitors the use of space and time.

Each of the activities is ***locally*** defined and executed at the point of invocation. This allows the maximum possible degree of parallelism. Our model contrasts strongly with extrinsic quantificational calculus models which are forced into global noneffective statements in order to characterize the semantics.

PROJECT 4
NATURAL LANGUAGE SYSTEMS

DEFINITION: Our goal in this area is to learn how to make systems that understand language in deeper and more effective ways.

More specifically, we plan to develop interactive natural language systems for use with the performance systems described elsewhere in this proposal. An internationally known program developed in this laboratory by T. Winograd represented a direct and intense effort to combine both new and old theories into a single performance system. A next step is to "push ahead" to make this system more powerful, and several workers are doing just that.

However, many very serious problems remain. Winograd and others working in this area solved some of the most persistent problems of computational linguistics by using "procedural methods", in which one can write special computer programs to handle all sorts of contingencies. But this leads to a dilemma. Although it seems easier to handle any particular difficulty, the whole system grows hard to understand because of new and difficult-to-describe kinds of interactions. Furthermore, parts of the system become inaccessible to "self-conscious" operation -- they are hidden in the system code, away from the deductive mechanisms. The system becomes unable to explain its actions, either in direct answer to questions or -- worse -- even in response to debugging probes by systems programmers.

For this reason, we cannot depend entirely on the "holistic" approach -- that is, of making one single complete demonstration and experimental system -- we must also consolidate uniformities in representations. The traditional approach of grammarians, to put things into uniform, declarative structures, makes neater those concepts that can be so expressed, but their inflexibility has always led to inadequate power. We plan to push in a variety of directions to get better control of a great many ideas about procedures, and semantics, annotation and compilation, and several specific questions about the relations of linguistic structures to heuristic models of the world.

MILESTONES:

This is a complex, rapidly developing field. There is close interaction between many different laboratories, and systems are being shared over the ARPANET.

We expect to see some of the effects of this research on the Natural Language milestones mentioned in Projects 1, 2, and 3 above, within three years. There are different syntactic requirements and semantic representational and reasoning problems in each system. It is appropriate for natural language research to alternate between theory and experiment. This is the way we have proceeded in the past, in the

works of Bobrow, Raphael, and Winograd, and we expect that style to continue to be productive. Here are some of the steps we expect to see in the next three years. We cannot be more precise about dates, but most of these are involved with Ph. D. theses already in progress, and three years has been our mean time from start to finish of such projects.

W. Martin -- Natural language system based on new ideas about case-frames. This is a major enterprise in Project MAC. A number of workers in our laboratory are working closely with Martin and his group in this project.

V. Pratt -- a metalanguage for describing classical parsers. With facilities for incremental changes, and use of local variables in the parser. The prototype system will describe rules of conjunction and elision.

M. Marcus -- Concept of "wait-and-see" parser. Some parts of language are more rigid than others. The order of things within English Noun Groups are relatively inflexible, as compared with the ordering of constituents within a clause. The wait-and-see parser should be able to exploit these inhomogeneities to get more efficient, yet still orderly, parsing programs.

D. McDermott -- attempt to uniformize knowledge about "doubting" and plausibility. Attempt to uniformize "expert" knowledge for compilation into procedures, with declarative representation available for deduction.

A. Rubin -- Relation between conventions of time and tense in English to assumptions about qualifiers and quantifiers. Use of "usually", "probably". The sentence "I teach on Wednesday" is really a future, "I teach on Wednesdays" is a (chronic) future" even though on the surface these use present tense. The latter means "I usually ---- ".

R. Moore -- In a somewhat similar vein, R. Moore is studying, from a procedural point of view, questions about referentiality and "knowledge about knowledge". Some of these problems have been long-term bugaboos in classical declarative logic, and have never been carefully treated in connection with computational linguistics.

D. McDonald -- An "advice-taker" system for interaction in English between a chess program and a chess expert.

PROBLEMS:

How to build a metalanguage for describing modern "grammars".

The new heterarchical linguistic models do not submit to prestructured "syntax-directed" methods.

What should be the strategy with respect to ambiguities in parsing now that semantic methods are beginning to exist.

Can we find reasonably regular ways to describe the new heuristic parsing schemes that use larger structures: scenarios, frames, etc.?

Most important to us are questions related to the issue of "uniformity". There is a constant struggle, in both linguistic and deductive research, between schemes that appear to neatly formalize an area of knowledge in a set of relatively orderly principles and rules -- vs. "exceptions" that become critically important in real applications and clash with the broad uniformities. Can we make a system that can exploit the advantages of the regular systems (easy to debug, easy to augment by other loosely associated workers, etc.) within a framework that can handle exceptions also in an "orderly", higher level way?

PERSONNEL: V. Pratt, R. Moore, A. Rubin, D. McDonald, M. Marcus; others who will become involved (this is a popular and growing area), and collaboration with W. Martin's group in Project MAC.

COSTS: Major Computational Resources Required. Systems use languages LINGOL, MICRO-PLANNER, PROGRAMMAR, LISP, CONNIVER, PLANNER, MIDAS and possibly others. Natural language programs saturate our current computer system when running, and growth of this area will require expansion of primary computer memory.

PROJECT 4
NATURAL LANGUAGE RESEARCH

A. INTRODUCTION

Let us pretend that it is possible to decompose English conversation into processes of Listening, Thinking and Speaking. In large measure, it was the erasure of such distinctions that was responsible for the great progress of the past few years. But traditional frameworks are still useful in presenting an overall view of what is happening.

Bobrow's program, STUDENT, was an early product of this laboratory that exhibited all of these components cooperating on high school algebra problems. Winograd's BLOCKS program, SHRDLU, is a considerably more sophisticated approach to the same issues, with a much deeper connection between details of the structure of natural English and the meanings of words, clauses, and whole discourses. Our experience with SHRDLU has had two important consequences:

> It has shown us that quite non-trivial natural language programs can be written with today's hardware and software. This demonstration has encouraged a fresh burst of natural language research, particularly in this laboratory.
>
> It has pin-pointed the problems we must deal with in producing a successor to SHRDLU.

It is appropriate for natural language research to alternate between theory and experiment. SHRDLU represented a direct and intense effort to combine the accumulated theory into an experimental program. We intend now to focus on the theoretical implications of SHRDLU's good and bad features, with the hope that what we learn will be applicable in the near future to another large program that will cope with many issues not addressed by SHRDLU.

We propose to divide our attention between listening, thinking and speaking, deferring for the time being the issue of getting the results of our efforts to interact gracefully. The reason for this temporary de-emphasis of the "holistic" approach to natural language is the need for considerable attention to theoretical detail before we will be ready for the next SHRDLU.

B. LISTENING

Under the rubric of "listening" we include everything necessary to convert typed input into a representation convenient for the "thinking" experts to work with. The immediate difficulty here is that it is unclear what that representation ought to be.

Vaughan Pratt and Mitchell Marcus are studying some aspects of parsing. Pratt is concerned with the representation of grammatical knowledge - what is an appropriate language for describing English to people and/or parsing programs? Marcus is designing a parser whose overall strategy takes advantage of certain properties of English.

Representation of Grammatical Knowledge - Vaughan Pratt

A programmer is free to impose as much or as little structure on his programs as he pleases. The advantage of having little structure is that he can attend to the problem of encoding his algorithm with a minimum of constraints on how he may express himself. Unfortunately this style of programming frequently leads to obscure programs, with some undesirable consequences:

The program is hard to debug.

It is hard to tell what information has been represented in the program.

The first consequence may be the programmer's own private problem. When the program proves successful, however, as SHRDLU did, computational linguists, in order to to build on the original program, may have to start from scratch. This is a major difficulty with SHRDLU at present. The second consequence is a problem that arises when the programmer wants to claim that his program in some sense describes the properties of its problem domain. To date this has not been a serious problem for natural language programmers because no program yet exists that contains enough information to constitute a better manual of English than those that already exist on bookshelves. This situation may change with the descendants of SHRDLU, and so is worth anticipating.

Historically, the first parsers were relatively unstructured. Later, context-free grammars became popular, and imposed their own peculiar brand of structure on parsers. Kuno's Predictive Analyzer represents the zenith of that era. In 1967, Thorne experimented with a more procedural approach to representing grammatical knowledge, and was quickly followed by Bobrow and Frazer, Woods, and Winograd. In the process, the flexibility conferred by the procedural approach allowed programmers to write code for each problem as it arose, to the detriment of any structure they or their programming language intended to impose. We seem to have witnessed the rise and fall of structured programming in parsers.

Pratt is exploring one approach to this apparent malaise which may combine the advantages of structured and unstructured approaches to parser writing. One wants to retain the advantage of being able to solve any problem just by generating the appropriate code, without losing the clarity conferred by a more structured style. The approach suggested is to start out with a metalanguage for describing parsers

which is designed to deal elegantly with the known issues in parsing natural language, and then not freeze the design of this metalanguage but rather let it grow in response to unanticipated needs. For this to be successful, growth will have to be slow; if it turns out that the process of adding descriptive mechanisms to the metalanguage does not converge rapidly, then we are no better off than with a completely unstructured approach.

There are at least four possible pay-offs from such an approach:

It will be easier to write parsers from scratch, and to understand and extend other peoples' parsers.

It will be clearer what our parsers have to say about English; that is, the parser itself may constitute a viable grammar of English.

The metalanguage that evolves may turn out to be a valuable descriptive tool for linguists independently of its application to computers.

The structure of the evolved metalanguage will in itself provide insights into the nature of English, in that it will be a source of generalities about different kinds of linguistic phenomena.

One fault with previous attempts to provide a complete descriptive language, such as Chomsky's transformational grammar, is that they did not explicitly provide for growth, and so stagnated. The situation is a little like that of a carpenter who initially sees a need for a hammer, saw and screwdriver, and from then on tries to do everything using only those tools. By allowing room for growth, we may avoid the situation in linguistics where one school of thought corrects for inadequacies in another's metalanguage by abandoning it completely and replacing it with a radically different one, which makes it difficult for the differing schools to build on each other's work.

A similar situation is obtained in programming language design philosophy. One school of thought offers PL/I as a complete panacea, while another prefers the notion of an extensible language on the grounds that we are unlikely to anticipate every programming need. It is possible that some of the ideas that have arisen in the development of extensible languages may be transferable to the problem in hand.

A serious difficulty with "growing" a metalanguage is that nothing at all is known about how to do this gracefully. Pratt has been using LINGOL, a programming language designed for linguists, as a medium in which to carry out some experiments with this approach. The results so far have indicated that smooth growth can be achieved only by playing it by ear. At present the process seems to consist of discovering

inductively the most general form of what one wants to say, and then designing the appropriate metalanguage feature. An example of such a feature is the use of the notion of local variable in the surface structure of English sentences. This feature was readily implemented by drawing on the corresponding feature in LISP. It seems to be a generally useful idea, being applicable to a large variety of problems in negation, as well as to difficulties in subject-verb and adjective-noun agreement when translating into, say, French or German.

A project that may test whether this idea of growth can be carried further is that of finding an appropriate way to describe the rules of conjunction and elision, as in "The Chinese have short names and the Japanese long". Further experience with such problems hopefully will lead to a better understanding of the value of this approach, and may lead to more systematic methods of growing metalanguages.

Wait-and-See Parsing - Mitchell Marcus

When a parser is faced with a choice of possibilities, it may choose one, proceed, and if difficulties are later encountered, back up to the point where the decision was made and choose another possibility. Alternatively, it may choose all possibilities simultaneously, and carry along all of their consequences in parallel, hoping that sooner or later most of the possibilities will fall by the wayside. In the early days of parsing, only the first option occurred to programmers. Then Cocke suggested an algorithm which was adopted by Kay, which implemented the second option. At the time it was felt that the parallel method was less efficient than the back-up method, which Kuno proceeded to use in the Harvard Predictive Analyzer. In 1967 Younger and Earley independently pointed out that in principle the parallel approach was really considerably more efficient than "destructive" backup, in which the result of parsing a substring is abandoned when backing up over that substring. Kuno changed his parser to incorporate non-destructive backup and reported order-of-magnitude improvements. Thorne's (1967) procedurally oriented parser used the parallel-parse approach also, but from then on Thorne's successors reverted to the backup method in the hope that by guessing cleverly, backup could be held to a minimum. So far no one has claimed that his parser guesses right most of the time.

Marcus proposes to combine the general idea of parallel parsing with techniques that take advantage of certain features of English. It appears to be the case that some rules of grammar are more robust than others. In particular, the rules that dictate word order within Noun Groups are quite inflexible. For example, one may say "my five handsome hunting dogs" , but not "my handsome hunting five dogs" or "my five hunting handsome dogs". Moreover, it is not even a question of style; the meaning of the phrase can be distorted or lost by such a permutation. The situation with Verb Groups such as "should have been seen" is even more rigid with respect to word order, although adverbs are permitted to appear within the group, preferably after the first

word of the group.

In contrast, the order of clause constituents (Noun Groups, Adverbs, Verbs) is relatively unimportant. Adverbs may appear anywhere, although they may not break up already formed Noun Groups. Subject and Object tend to go before and after the verb respectively, but this rule is frequently broken, e.g. "Some flavors almost everybody likes", and poetic license readily allows "Heard I the lark at eventide" while tending to frown on interference with Noun Group rearrangement, other than to allow adjectives to follow nouns (a variant also permitted occasionally in prose).

These observations suggest that a fairly conventional syntax-based approach may be adopted for parsing Noun Groups, and that a more semantically oriented technique is appropriate for elucidating the relations holding between the clause constituents.

This motivates the notion of the Wait-and-See parser. When a Noun Group is being scanned, the parser commits itself immediately to the internal structure of this Noun Group, and also attempts to determine its semantic referent in order to facilitate assembling clause constituents. Ambiguities within a Noun Group are recorded as part of the structural information about the Noun Group and are thus an internal problem and only marginally relevant to the clause constituent assembler. Ambiguities about where the Noun Group begins and ends are recorded as separate Noun Groups, to be sorted out by the clause constituent assembler. Verb Groups are handled similarly; ambiguity tends to be less of a problem with Verb Groups than Noun Groups, and so the situation is even simpler. Other types of clause constituents are for the most part relatively simple and require only cursory inspection in order to identify them as clause constituents.

The part of the parser described so far may be characterized as a "middle-down" parser, as distinct from the usual top-down and bottom-up parsers. It is middle-down because initially it is on the look-out for clause constituents, and can generally manage to predict the type of constituent from looking at its first word. The commonest exception to this is in the case of Noun Groups that don't begin with a determiner and follow another Noun Group, as in "The city people like is Boston". In this and similar cases, the parser preserves its middle-down flavor by guessing all possibilities as early as possible.

As clause constituents are discovered by the first component of the parser, a second component, the one that is going to draw on semantic information, attempts to fit these constituents together to form clauses. In the absence of rigid syntactic rules at this level, this component will frequently have to allow several constituents to accumulate while it waits to see (hence the term Wait-and-See parser) what is the most plausible way of putting the constituents together. It is at this level that considerable emphasis will be placed on clever

methods to use all available syntactic, semantic and contextual information to complete the parsing of the sentence. Many problems related to conjunction and elision will also be handled at this level, and Marcus hopes that his approach will considerably simplify these problems.

The wait-and-see approach can be seen to differ in important respects from both backup-oriented and parallel parsing. Certainly no backup is involved. On the other hand, leaving clause constituents lying around unattached until their role can be determined differs from the parallel approach in that the latter attempts to commit constituents to all of their (possibly multiple) roles at the same time. The wait-and-see approach commits nobody until a definite role emerges.

Marcus proposes to implement a parser based on these ideas starting at the end of the current summer.

C. THINKING

"Thinking" embraces a wide range of activities and problems. A listener may try to figure out what the speaker really meant by his utterance; he may search for inconsistencies in it; he may deduce its consequences; he may attempt to reply to it; he may take physical action based on it; he may draw conclusions about the speaker; or he may simply choose not to believe a word the speaker is saying. Obviously this is not intended to be an exhaustive list - it just illustrates the complexity of the "thinking" problem domain.

Consistent with this complexity, we have most of our people working on problems in this area. Drew McDermott is following up the ideas developed in his Master's thesis, and is concerned with the problems of representing and updating knowledge; the degree to which one can successfully carry out the latter depends heavily on the nature of the former, and so work is needed on suitable representations. Andee Rubin is addressing the interplay between time and tense; nominally English has provision for dealing with past, present and future events, but in practice relying on tense alone may give quite erroneous information about time. Robert Moore is developing a language suitable for representing a number of concepts that are dealt with poorly or not at all by other formalisms such as the predicate calculus.

Representing and updating knowledge - Drew McDermott

The notion of procedural embedding of knowledge has been popular for some time in this Laboratory. Drew McDermott has just completed a Master's thesis in which he explored this concept in considerable depth, by representing as procedures all the knowledge in TOPLE (for TOPLEvel), a natural language understanding system. McDermott is dissatisfied with some aspects of procedural embedding.

One problem that arose in TOPLE was that different types of knowledge varied in their accessibility by TOPLE. Some knowledge was stored as CONNIVER items. These are pattern-accessible pieces of knowledge. Things like (SLIGHTLY-BIGGER BOX1 BOX2) were stored this way. However, knowledge traditionally requiring quantifiers, like, "if x is slightly bigger than y, and y is slightly bigger than z, then x is bigger than z," were hidden away in procedures and ad hoc data tables. Such assertions were not in the same format as the items, and could not be meditated upon at all by TOPLE. Still other forms of knowledge, such as, "if s is a step in the proof of p, and you want to doubt p, try doubting s," were even more remote, being distributed throughout the system.

If we assume that a program that "knows itself" is going to be able to make more effective use of what it knows, it would seem reasonable to elevate all knowledge to the same level. This seems to lead back to the idea that knowledge should perhaps after all be represented as declaratives. Unfortunately, one always needs some procedure lurking somewhere in order to make things work, and so we appear to be in a bit of a dilemma. McDermott plans to sort out these issues in the not too distant future.

For the present, McDermott is considering alternative representations for belief systems similar to TOPLE. The types of knowledge that need to be represented include:

Knowledge about particular domains. This is the knowledge about space, what monkeys can do, etc. In TOPLE, it is scattered about various CONNIVER methods.

How to use such information. For example, the different partial orderings (such as size lattices for physical objects, space relationships) are searched by expert routines rather than by a general deductive system. A different example is the monkey simulator by which TOPLE understands the monkey in its world. The knowledge about what a monkey can do is scattered about and interleaved with knowledge about what he is likely to do.

How to doubt things. Each routine that adds a piece of knowledge to the world model is responsible for saving with it the information needed to remove it later should it conflict. There is no reason why this information shouldn't be expressible.

What changes are better than others. This is done with entirely ad hoc numbers and whistles in the current version, and should definitely be systematized.

How to debug statements with quantifiers. In the current TOPLE, all such statements are buried in routines which the system is not

smart enough to examine. There is no reason why a formal language cannot be discovered which can express how to debug statements in the language.

The more knowledge is encoded in declaratives, the more the belief system will have to be reading instructions and following them or figuring them out, rather than just doing them, and the slower things will go. This is especially true if, say, knowledge about how to go about deducing is stated in a deductive formalism. A most promising approach seems to be to express everything in declaratives and compile the world model periodically using the methods studied by Sussman here at M.I.T. He has a BLOCKS-WORLD system with knowledge about programming, debugging, and physics, which compiles statements about blocks and actions into programs to carry them out. McDermott intends to study his work very carefully to see how it might be converted to thinking about doubting beliefs instead of moving blocks, and to see to what degree the kinds of knowledge his program has can be expressed in a uniform language. Then perhaps his program can be used to compile general declaratives into expert procedures.

Time and Tense - Andee Rubin

This year Rubin proposes to continue her investigation of natural language, concentrating on the semantics of time expressions. A preliminary goal will be to understand the syntax of the English tense system; this structure has already been investigated, as evidenced by the work of Halliday and Bruce. However, knowing the tense of a clause is often not enough; in many cases it is completely misleading. Even the simplest of tenses, present tense, has a myriad of uses, spanning past, present and future. Consider these examples:

Cars use gas. (universal)
Alphonse loves his doctor. (present)
Alphonse sees his doctor Wednesday. (future)
Alphonse sees his doctor Wednesdays. (future)
If he goes, I'll go. (future conditional)

Even sentences which appear at first glance to contain present tense both syntactically and semantically, may have this meaning changed by context. A secretary entering an office with official notices who asks "How many people are in this office?" doesn't expect a reply which simply requires looking around and counting. Her question might be rephrased as "How many people are usually in this office?" Time considerations are not immune from the whole "usually" problem: how to do logic with qualifiers such as usually, probably and typically.

On the representational side, time will not be organized as a uniform set of points linearly ordered by the relation "before." Rather, time will be in chunks, such as YESTERDAY, THIS-YEAR, THIS-MORNING etc., which will be in general hierarchically organized. Events will be

partially ordered within contexts, but certainly no total ordering can be expected. In addition, certain subcontexts may have a predetermined internal structure; we tend to organize workdays into MORNING, LUNCH, AFTERNOON, DINNER and EVENING and the time of events is thus often specified as "after dinner" or "before lunch."

This research will be carried out in the mini-world of everyday activities, such as going to school, shopping, going to work, traveling etc. The program will be a virtual roommate which accepts information about the past, present and future activities of its roommates and answers questions about their whereabouts, plans and schedules. It is hoped that this context will provide plenty of opportunities for using and deciphering more complex time and tense references.

A Computational Theory of Descriptions - Robert Moore

Methods advocated for representing knowledge in Atificial Intelligence programs have included logical statements (McCarthy, Sandewall), semantic networks (Quillian, Schank), and procedures (Hewitt, Sussman and McDermott). All these approaches share one fundamental concept, the notion of predication. That is, the basic data structure in each system is some representation of a predicate applied to objects. In this respect, the various systems are more or less equivalent. But this basic idea must be extended to handle problems of quantification and knowledge about knowledge. Here the systems do differ although these differences result from the descriptive apparatus used in the particular systems being compared, rather than from an inherent advantage of, say, procedures over declaratives or vice versa.

Advocates of PLANNER (e.g. Winograd, p. 215) have argued that the predicate calculus cannot represent <u>how</u> a piece of knowledge should be used. But this is true only of the <u>first-order</u> predicate calculus. In a higher-order or non-ordered declarative language, statements could be made which would tell a theorem prover how other statements are to be used. PLANNER, on the other hand, has no way of directly stating an existential quantification, but this does not mean that procedural languages are necessarily incapable of handling that problem. Moore has worked out the preliminary details of a language D-SCRIPT with powerful formalisms for descriptions, which enables it to represent statements that are problematical in other systems. Since it is intended to answer questions by making deductions from a data base, it can be thought of as a theorem prover. Since it operates by comparing expressions like the data-base languages of PLANNER and CONNIVER, it can be thought of as a pattern-matching language. And since it includes the lambda calculus, it can be thought of as a programming language.

The following example suggests the sorts of problems addressed by D-SCRIPT. A classic problem is that of representing opaque contexts. An opaque context is one which does not allow substitution of referentially

equivalent expressions or does not allow existential quantification. For example the verb "want" creates an opaque context:

(1.1) John wants to marry the prettiest girl.

This sentence is ambiguous. It can mean either:

(1.2) John wants to marry a specific girl who also happens to be the prettiest.

or:

(1.3) John wants to marry whoever is the prettiest girl, although he may not know who that is.

Under the first interpretation we can substitute any phrase which refers to the same person for "the prettiest girl". That is, if the prettiest girl is named "Sally Sunshine", from (1.2) we can infer:

(1.4) John wants to marry a specific girl who also happens to be named Sally Sunshine.

We cannot make the corresponding inference from (1.3). It will not be true that:

(1.5) John wants to marry whoever is named Sally Sunshine, although he may not know who that is.

Because of this property, (1.2) is called the transparent reading of (1.1) and (1.3) is called the opaque reading. It is almost always the case that sentences having an opaque reading are ambiguous with the other reading being transparent.

Other problems deal with time reference (compare "The President has been married since 1945" with "The President has lived in the White House since 1800"); and representing knowledge about knowledge (e.g., if John knows that Bill's phone number is 987-6543, may we represent the English sentence "John knows Bill's phone number" as "(KNOWS JOHN (PHONE-NUMBER BILL 987-6543))?); one really wants in one's formalism to be able to name a piece of knowledge without having to say what it is.

For all these types of sentences, D-SCRIPT provides representations which allow the correct deductions to be made. Further, it provides separate representations for each meaning of the ambiguous sentences, and these representations are related in a way that explains the ambiguity.

A detailed account of D-SCRIPT may be found in [Moore 1973].

D. SPEAKING

The question of "how do we speak?" has for the most part been ignored by natural language researchers in favor of asking "how do we understand?".

But there are important questions here about how we organize our thought. Understanding how people decide what is appropriate to say -- applying linguistic knowledge -- and knowledge about the state of understanding of the recipient -- is an intellectual task of no small order.

Work done to date on such systems as LUNAR, SHRDLU, or other question-answering systems involve rather weak generation schemes or speaking components, relying to a large extent on inflexible combinations of canned or fill-in-the-blank responses. Computer programs have not been conscious enough of what they said, making it impossible to carry on a proper conversation. In the future, programs in more complex domains, such as medical diagnosis, will be required to fluently explain their reasoning and to be aware of the knowledge of their audience so as to speak at an appropriate level.

If we want to expose our conversation machine to Turing's test; that is, to have a lifelike quality, the system needs a coherent representation of the scenario in which it is involved. The discourse system must be designed to exploit this structure. Thus the speaking component of a conversation machine, while not critical in some applications, provides the glamour that makes conversations seem purposeful, fluent and natural. This assumes, of course, the existence of high quality listening and thinking components.

A Talkative Chess Program -- David McDonald

David McDonald is planning to develop a prototype program cabable of discussing its own reasoning processes in natural language. It should be able to change its ideas as a result of the conversation.

Chess has been chosen to be the domain because of the complex but straightforward nature of the reasoning involved and because, since it is so well contained, there would be no problems with ambiguous vocabulary. McDonald is interested both in linguistic analysis of what kind of language knowledge is involved in speaking and with the question of how that knowledge is structured in a computational environment and of what the points of connection are with the other parts of the program's cognitive structure.

Once we have an adequate semantic domain, we can consider for the first time (at least in a computational context) what are the differences between the choices every language speaker makes continuously. Consider these sentences, with the same simple propositional content:

John's bitching in the office indicates unhappiness with his job.

John's bitching in the office indicates that he is unhappy with his job.

John bitches in the office which indicates that he is unhappy with his job.

John bitches in the office indicating that he is unhappy with his job.

What are the differences between these sentences that motivate us to pick one over the others? Can we pin down and work with the notions of overtone or viewpoint that are involved here?

PROJECT 5
IMPORTANT THEORETICAL TOPICS

DEFINITION: This "project" brings together a variety of vital questions that underlie the problems faced by all the other tasks.

Inference from Incomplete Data
Modal Logic and Formalization of Common Sense
Qualitative Physics
Scenarios, Frames, and Reasoning by Analogy
Memory Structures and Difference Networks
Theories of Semantic Information Retrieval

MILESTONES: Time scales for recognizable achievements in these theoretical areas are hard to set, but in most cases we expect significant results within the two years, because of the involvement of Ph.D. students.

Learning Machines: As always, the limitations on learning are set by what we know about representing the kinds of things to be learned. We expect substantial extensions of Sussman's debugging-learning system to appear within two years.

Structural Description: M. Dunlavey's Ph.D. topic concerns extending Winston's "grouping" descriptions so as to deal with interactions, e.g., to understand the structures at the corners of periodic brick-like structures -- given descriptions of the non-singular parts and general knowledge about three-dimensional objects.

Qualitative Physics: Several members of the Laboratory -- Papert, Abelson, Minsky, Goldstein, DiSessa, and others -- are working on attempts to formulate commonsense knowledge about static and dynamic mechanics. These will result in several publications within the next year or so. Many problems arise in experiments on machine intelligence because things obvious to any person are not represented in any programs. One can pull with a string, but one cannot push with one. One cannot push with a thin wire, either. A taut, inextensible cord will break under a very small lateral force. Pushing something affects first its speed; only indirectly its position! Simple facts like these caused serious problems when Charniak attempted to extend Bobrow's "Student" program to more realistic applications, and they have not been faced up to until now.

Inductive Inference: R.J. Solomonoff will be a member of the laboratory in 1974. We hope to see this result in some applications of his well-known general theory of induction. It is not known whether this kind of theory can in fact be applied usefully, even though many mathematicians and philosophers are impressed with its clarity

as compared to previous formulations.

Modal Logic: J.R. Geiser will continue to explore relations between the AI formulations and those of modern work in modal logic.

Frame Theory and Scenarios: Minsky and several students are working on a new strategy for representation of common sense knowledge. An extensive publication should appear within a year.

COSTS: Chiefly in personnel, with console and computational costs for heuristic programs in the first three and last areas mentioned above.

PROJECT 6
EXPERT PROBLEM-SOLVING PROGRAMS

DEFINITION: In this area we place projects in which ideas have reached the point of concreteness at which enough theoretical problems are understood to justify an experimental heuristic program. The most advanced of these is Chess: R. Greenblatt expects to make important advances in incorporating surprise-analysis and some other tactical and strategic elements into his chess program. See the detailed discussion in Chess Subproject Section, below.

Also in progress are

Geometry: A. Brown is working on a Ph.D. thesis in this area. His program will advance our understanding of how to use logical methods under the control of knowledge-based methods for representing proof and application strategies. The scientific directors of the AI Laboratory have taken a strong position against the world-wide general optimism about the potential in "mechanical theorem proving" within traditional formal systems of logic or modest variants of these. To put it in an almost quixotic form, we consider Logic to be little more than a last-resort heuristic to be tried when common sense has failed. Nevertheless, we need to find forms of logic that can be used by intelligent systems without the restrictions that logicians have traditionally assumed necessary and valuable.

Series Acceleration: R.W. Gosper has discovered a variety of methods for decomposing and reassembling convergent mathematical series so as to greatly accelerate the rates of convergence over certain domains of range and desired precision. He is trying to systematize these results. It is hoped, also, that the theory of these methods may illuminate long-standing questions about the relative computability of classical transcendental numbers.

Semantic Information Systems: J. Meldman is beginning a doctoral thesis in the area of finding legal case citations; i.e., to present a case and have the system find one in the literature that agrees in enough legally important features.

Theorem-proving: A. Nevins is extending his logical deduction plus heuristic case-analysis system.

MILESTONES: Projects such as these usually make significant steps in two years or so, as they are on the scale of individual large heuristic programming projects. We expect a definitive publication in each of the above in 2 years or less.

APPLICATIONS: We do not think of these as applications systems so much as "checkout" and field-testing of ideas in "tough" environments. Generally, all of the central ideas in these experiments are concerned with how to deal with:

Decisions under uncertainty
Decisions under limited resource constraints
Decisions under imperfectly specified goal conditions

or where one cannot actually check out all possibilities but has to choose some action to perform.

COSTS: Personnel, console, and processor costs. None of these require special hardware. In some cases, the principal researcher will require systems programmer assistance in expediting interactive services, special displays, or extensions of the high-level languages.

PROJECT 6.1
A COMPUTER CHESS "SUBPROJECT"

A. INTRODUCTION

Although chess research has proceeded at the MIT-AI lab and its predecessors for almost 15 years, it has not previously been made a part of project proposals or received explicit funding support. The following brief history is offered to explain the current state of the subject.

How a chess program works

Both chess programs and humans playing chess (from all evidence) rely heavily on the method of heuristically limited search. This simply means that given a position, one proceeds to look at the possibilities for one side, the replies for the other, etc. Rapidly, however, one runs into the exponential growth of the tree phenomena and it is necessary to stop the analysis well short of final solution (checkmate) and form a judgment as to the merits of the position. The most common way to conceptualize the structure formed in this way is called the game tree. The given position is visualized as the top node in the tree, with alternatives for the side which is to move forming the level 1 branches. The level 2 branches are possible replies for the other side, and so forth. For computers, the module that forms this judgement at the terminal nodes is called the static position evaluator. The "rational" behavior of the players can then be modeled by a method called mini-max. This method simply reflects the fact that given a number of choices, a player will choose the most favorable to himself, and that what is good for the white player is necessarily bad for the black player. Using the mini-max method, the values assigned by the static position evaluator are successively propogated back to the n-1 level nodes, n-2 level nodes, etc., until a value is assigned to the topmost node. Then the move which led to the the top node getting its value is played and the procedure is completed.

A Few Basic Improvements

The alpha-beta algorithm is a method whereby the tree searching procedure described above can be made more efficient. Its basis is that once a move has been proved bad (worse than some other move) it is not necessary to continue looking to see exactly how bad it is. A plausible move generator may be incorporated for the purpose (among others) of discarding worthless moves more efficently. A large class of methods may be used to attempt to insure that the positions represented by the terminal nodes are stable, that is, neither side can greatly improve his position in a few moves. However, this remains a very difficult problem

(equivalent in the limit to solving chess).

B. "LEVELS" OF CHESS PROGRAMS

The chess programs developed in the last 15 years can be roughly classified as follows:

Level -1

Newell, Simon, and Shaw program and the Kotok et al program, both circa 1961. These early programs were extremely weak, playing at the level of a novice who has played about ten games in his life. One reason for this weakness was that the programs were overly concerned about "high level" heuristics at the expense of the basic maxim of chess to "take the other guy's pieces". They failed to play out exchanges, resulting in gross blunders. The search parameters were far too narrow for the sophistication of the program. For example the Kotok program investigated 4 plies deep with widths of 4, 3, 2, and 1 (meaning it looked at the "best" 4 moves in the origional position, the "best" 3 replies to each of them and so on). The program would have no doubt played better if it had been set to look only two plies deep, looking at 8 top moves at each ply. Another major factor was extreme lack of debugging and tuning since each program played only a few games in its entire existence.

Level 0

The Russian program of 1961, the Greenblatt program of 1966, and the Carnegie program of 1972 search plausible capture chains which are indefinitely deep, thus avoiding gross blunders that leave pieces totally loose. They are also characterized by much wider searches than the level-1 programs, investigating from 15 to all the legal moves at the top two levels at tournament speed settings (2.4 minutes per move).

There is little or no attempt at sophisticated chess knowledge, concentrating instead on brute force searching. The programs are relatively simple and easily debugged. The early Greenblatt program achieved a USCF tournament rating of 1450, and a Carnegie rating of 1280. This means that they could beat most amateur chess players, but were in the lowest class of tournament players.

Level 1

The Greenblatt program of 1968 incorporated various improvements. A "simple" (by later standards anyway) plausible move generator was used. It has some idea of positionality, mechanisms for crediting attacking

and defending moves, and consideration of discoveries and blockages, among other things. The static position evaluator included pawn structure as well as material. This program achieved a rating of 1720 in one tournament and once drew an 1880 player. These are the best achievements of any chess program to date. An 1880 player is right at the median of tournament players, and since 200 points corresponds approximately to one standard deviation in the distribution, 1720 is slightly less than one standard deviation below median. The program could beat almost all amateur (non-tournament) players it played.

Level 2

The Greenblatt program of 1971 had conceptualization of the idea of threat, answering the threat, and other "medium level" concepts. Forward cutoff is heavily relied on to avoid unnecessary searching. Assertion lists called move description tables were formed and processed, leading to realization of how the various components of the move affect each other. For example, if piece X is attacked and piece X is also pinned, then the attack assumes special significance.

In one tournament, the program drew 2 games in 6 starts, all against strong players rated 1680 or higher. Although this resulted in a performance rating lower than that of the level 1 program, when allowance is made for the fact the opponents were uniformly stronger, the overall performace was only slightly weaker (In achieving the 1720 performance rating, the level 1 program had the opportunity to beat two 1400 players). Program complexity is an order of magnitude greater than the level one program, and there was a high incidence of bugs.

Level 2.5

The current Greenblatt program has further improvements and better debugging facilities added to the features of the level 2 program. It also incorporates into the static position e aluator new piece mobility and board control components. It has not been tried in tournament competition. In considering level 2 and higher programs, one must keep in mind the tremendous accuracy demanded of the plausible move generator. These programs basically don t look at moves unless they can see they are good. There are many, many reasons a move might be good, and it only takes one omission to lose the game. The plausible move generator is faced with the task of examining hundreds of moves, each of which might be good for between one and two hundred different reasons. The actual game tree is vastly smaller than the level 0 and 1 programs, with a tournament move based on making perhaps 250 or less moves. The programming effort involved has greatly exceeded origional estimates, but it appears the job is very nearly done at the present time.

Level 3

The next step is the passing of symbolic information between levels of the search. Thus if certain moves are found to be good at the lower levels, information is passed back describing them so that the higher level can try to stop them (surprise analysis). A feature called "threat pass down" propagates information from the higher levels down. Basically, if a move is played with a known threat, it assures that analysis is not stopped before that threat is disposed of. These and other related features should be implemented in the near future, but then a long process of tuning and development will probably occur. What rating a level 3 program can achieve is an interesting question. I believe an 1800 rating is not unreasonable, but upward progress may soon be made impossible without improvements to the end game. In any event, this question should be resolved.

DISCUSSION: At level 3, at last, we would be developing realistic problem-solving skills in a situation where a hostile opponent is trying to find and exploit our weaknesses. It remains to be seen just how this differs from the "games against nature" that AI programs ordinarily consider. To be sure, many heuristic programs have been written to play games. But we feel that in most cases the approach could be characterized as depending mostly on "tree-pruning" concepts, and do (or did) not analyse the special features of the competition. (The analyses of Newell, Shaw, and Simon are not subject to this criticism, we feel.)

C. CHESS AS A MICROWORLD FOR AI

Chess has been of interest in the past mainly because of its classical nature and certain well-publicized bets, predictions and international matches. Recently, however, it is beginning to appear that chess may well turn out to be an important micro-world for AI in its own right. Comparing Chess to other micro-worlds (such as the BLOCKS WORLD), we note the following points:

1) Chess is a "real" problem in the sense that is not defined by the AI researcher, but by the outside world. Parts of the problem can not be defined away without paying a high price.

2) Chess is a "world" as distinguished from a "micro-world" in that we are competing with serious adult persons rather than trying to do what every six year old can, as in natural language.

3) Chess offers unique advantages for learning programs. Note that by learning here I dont mean adjustment of pre-existing polynomial coefficents or spot assimilation of some fact like an assertion. I refer to an extended analysis of a hard problem not specifically foreseeable by the programmer and conceptualization and incorporation of the results. At a later stage, chess may prove

interesting because of the opportunity it may offer to study "meta-compiling" processes. In other words, once the data to be learned is obtained, it must be processed to more fully incorporate it with the existing structure and to increase efficiency. Compare the first time you ride a bicycle with subsequent attempts.

4) Chess is already one of the most deeply studied areas of human problem solving, in view of the work of DeGroot, Newell and Simon, and some others.

D. PHASES OF THE GAME

It has been recognized by human writers for many years that chess can profitably be considered to consist of three subheadings, the opening, the middle game and the endgame. It is found that level 0 and above programs play reasonably well in the opening using the middle game methods, although it is a simple matter to incorporate an opening book of pre-determined responses. The main effort of computer chess programmers has been on the middle game, which is dealt with above. No serious effort has been made to date to deal with non-trival endgames, although it is frequently said that the difference between a master and a expert is endgame play. If the program has won material in the middle game, it can usually arrange to win by queening passed pawns.

PROJECT 7
IDENTIFYING AREAS OF APPLICATION

A. INTRODUCTION

Over the past decade, there has been (in our firm opinion) great progress in the field of Artificial Intelligence. A large portion of the important work in this area was supported by ARPA's IPT sponsorship. The results are not widely understood, partly because this is a genuinely technical domain and partly because both the press and the scientific public have strong, non-technical, prejudices about what is possible and what is not. Thus, in order that the investment be effectively exploited, it is important both to produce better expositions and documentation of technical progress and to produce informed technical evaluations of the practicality of near-term application projects.

We are not quite ready to make a commitment to do this. We do not think there would be any value in commissioning such a study without pre-empting the services of one of the very few men who we feel have comprehensive overviews. What we do propose is to work over the next year to produce a general exposition, based on expanding the plan of our 1972 Progress Report, which has been demonstrably successful in explaining much of the subject to other audiences.

In particular, we are proposing two small-scale study projects, outlined below. The first will be to assess the implications of AI research on the general Information Retrieval problem; we feel that progress in semantic representations may have brought this area close to a major breakthrough point. The second project will be to develop a work-plan for a large-scale assessment of AI potentials; some preliminary sketches are presented below.

B. STUDY PROJECT FOR A PERSONAL ASSISTANT SYSTEM

DEFINITION: Study project for a major application: the "manager's assistant". We feel that the field of Semantic Information retrieval is ready for major advances. There already exist a wide variety of specilized Management Information Systems. They illustrate very well a familiar sort of "AI Paradox":

> It is easier to make a program behave as a highly specialized expert than to make one behave in a straightforward, common sense way.

This applies to Information Management just as it does to many other fields. Thus, over the years we have seen special systems developed

for

Inventory control
Classroom scheduling
Assembly-line and job-shop allocation
Investment trust reports
Library retrieval
Accounting data

and so forth. One uses such systems (if one uses them at all) on constrained occasions, for constrained purposes. There are no systems, however, that one can use for "ordinary" purposes, to do simple jobs.

What limits the amount a person can do? Ability, opportunity, information, time. Every person is a manager, deciding what to do himself, what he can get others to do, what information he needs.

We would like to consider the possibility that the time is ripe for developing a "general-purpose" computer based Assistant system. Its purpose would be to save one's time, help get and keep information, and perform chores that either one has to do or one does not do because of all those limitations. To present the image, here is a scenario involving a Professor and a Personal Assistant program.

This example is chosen only because we know this situation best. Any reader more familiar with such functions as Administration, management, Engineering Design, Military Command, etc., can easily construct a similar scenario for that kind of function.

> A professor happens to read a paper on plasma stability by a certain author. He does not consider the main result very original or important, but a certain example in the paper might be relevant to one of his students' work.
>
> He tells the Assistant Program the citation of the paper, and enters his remarks. In 1975, he would have to type it in; in 1980 there is a good chance he could speak directly to the machine.
>
> The Semantic Analyser sends a note to the student (or to the student's Assistant Program). It makes a note about the citation and the Professor's remarks in a data bank for information retrieval about Plasma theory and related subjects; it assigns a description that will not attract attention unless there is some reason to suppose that the situation might have changed regarding the paper's originality.
>
> Some months later, the Professor needs the example! He cannot remember the citation or the author's name. He does not even remember that the paper was about plasmas -- the example

concerned classical Hamiltonian machanics. Fortunately, he does remember that he had thought the student might be interested, and that the incident occured last Winter. He tells the Assistant Program this, and it instantly produces the citation, and displays the relevant part of the text on the display.

In the meantime, it so happens that the author is announced to present a Physics Colloquium at Harvard. The Assistant Program, which reads the daily newsletters every morning, notices the name and leaves a "MAIL" signal for the student who, it thinks, might possibly be interested in meeting the visitor.

The message can just as easily be sent, over a computer network, to any known individual who might want such notification and has agreed to receive such services.

In our study so far, we have come to the conclusion that such a system could be brought to a practical level -- for use by a computationally sophisticated person -- in two years. It would take longer, perhaps another two years, to extend its capabilities and smooth out the human engineering to make it equally useful to non-specialists. There are all sorts of hard problems that make precise time estimates difficult, since so much will depend on concurrent progress in the computer manipulation of common-sense ideas and meanings. In any case, we believe that the possible payoff of such systems could be enormous, and have all kinds of applications.

MILESTONE: We are NOT proposing to embark immediately on such a project. Instead we are planning a serious feasibility and exploratory study during the coming year. The result should be a report surveying the state of knowledge, comparing strategies of attack, determining which applications (Administration? Management? Personnel Supervision?) would be best for the first attempt at building a working system.

PERSONNEL: We do not feel that our present staff has all the expertise necessary to make such a study, and we propose to use a panel of consultants, involving such people as Engelbart and Teitelman, who have made studies into different aspects of this area.

C. STUDY PROJECT FOR ASSESSING AI APPLICATIONS IN GENERAL

DEFINITION: A thorough, substantial assessment of the current state of knowledge about machine intelligence; the important solved and unsolved problems, and assessment of those areas in which applications seem most likely to pay off in the near and middle future.

MILESTONE: A book-length discussion of the issues, of clarity and composition suitable for non-specialists, to be completed by the end of

1975.

PERSONNEL: Marvin Minsky, Seymour Papert, possibly others.

Some general remarks on the state of AI will be found in Section 3 of this proposal. Immediately below are some areas we feel competent to organize more detailed assessments for.

D. HAND-EYE MANIPULATORS

It is hard to think of any field that could not be transformed by the availibility of machines that perform physical actions quickly and carefully under visual and tactile feedback control. The projects proposed here are especially relevant to MAINTENANCE, INSPECTION and REPAIR of complex systems, by assisting relatively unskilled personnel.

In particular, the attention of this Laboratory will be concentrated, whenever the extra complication is not excessive, on the development of methods and systems to deal with "Micro-Automation" -- that is, physical manipulation of very small components such as will be found in advanced electronic systems. This is an area in which there has been particularly little advanced development, limiting the practical applications.

This area is of particular interest because human workers do not compete in it very well. AI techniques in this area will also be useful on the Macro-Scale, e.g., in CONSTRUCTION situations where there are problems about human skill, manpower, safety, or speed.

Eventually we expect to see these devices employed in areas of critical civilian concern, e.g, MINING, UNDERSEA DEVELOPMENT, and SPACE.

For example, unless fusion power becomes available soon, we will need more MINING and/or SOLAR POWER. Advanced Automation will be required for low-grade shale mining or for the construction and (especially) maintenance of enormous arrays of solar cells on earth or in space. In mining, AI techniques might help solve problems of moving and restoring.

Finally, we expect to see major applications in Agriculture, perhaps in two decades.

Some of these are discussed further in our MICRO-AUTOMATION proposal, MIT-AI memo 251.

NOTE: In earlier proposals and publications we observed that U.S. "knowhow" in advanced automation has been permitted to coast along. We believe that the skills involved in what we call "advanced automation" will become vital to many of the other near-crisis problems of the near future, as noted in other items below. We recognize that ARPA is not

the agency to carry the research load for this area, but we want to put on record that we expect the work described in this proposal to have very substantial contributions to those areas.

The problem of low U.S. productivity is now recognized, both with respect to foreign competition and in absolute internal matters. Certain kinds of advanced automation can forestall or at least space out the adjustments that might otherwise arrive in destructive waves. We cannot take up here the question of social consequences of advanced automation, but will state our position briefly. Certainly, if advanced automation were thoughtlessly applied to existing industries without compensatory planning, serious dislocations could ensue; this responsibility of economic planners must be recognized. On the other hand, ordinary marketplace adjustments do not seem able to handle rapid large-scale changes in such areas as agriculture, energy, housing, transportation, education and health services. Only advanced automation can make possible large-scale physical adjustments without similarly large-scale social transients.

E. ADVANCED INFORMATION SYSTEMS

We believe that AI research can show the way to computer-based information systems far more capable than have ever been available. We plan to attack the area of Information Retrieval, both for traditional data-base and library problems and for more personal MANAGEMENT INFORMATION SYSTEMS problems.

The new Information Systems should help to increase the effectiveness of individuals who are responsible for complicated administrative structures, as well as for complex information problems of technical kinds. In particular, the services will be available and designed to be useable over the ARPANET, and will be designed to interact with the personal systems of other individuals to recognize conflicts, and arrange communication about common concerns.

F. AUTOMATIC PROGRAMMING, AUTOMATIC DEBUGGING

In all the areas noted above, and already in many conventional areas, computer-programming plays a large, expensive and frustrating role. Not only expense, but delay and unreliability of programs threaten to grow rapidly in the future. By pushing ahead on the related fronts of Automatic Programming, Automatic Debugging, and "SELF-DOCUMENTING" programs we believe that this trend can be held in control.

G. MEDICINE

There is now more real physiological, pathological, and therapeutic knowledge than any small group of physicians can encompass. But there are not enough "specialists" to serve as consultants and this problem will become steadily worse, to the point that present inadequacies and inequities will become dangerous. In another project, we plan to apply our knowledge-based programming technology to the construction of a "computer consultant" -- a large heuristic program that will be able to provide to general medicine the equivalent of a highly skilled consultant-specialist -- with a level of competence well above that available in the average non-research hospital service area. (The first such system will, probably, be concerned with renal and cardiac problems.)

The resulting services will be available to unspecialized personnel in rural or otherwise remote and inaccessible areas, over computer network connections.

We are asking for NIH support for joint work between AI workers and an outstanding teaching hospital in this area.

Similar health information services will have to be developed for other medical areas, and for diagnosis, treatment planning, special therapies, and health services administration.

There are direct applications of AI results, both from the robotics area and from the general information managing areas, for devices for PROSTHETICS for the handicapped. Generally, all such work has equal value as potential "augmentation" of normal persons.

H. EDUCATION

There are already significant applications of AI ideas to training and education, far beyond the "first-order" services provided (for better or for worse) by the earliest "computer-aided instruction" systems. The value of the ideas in our (NSF-supported) LOGO project are widely recognized by educators. In attempting to understand how intelligence develops, it is of course especially fruitful to concentrate research on younger children for scientific reasons. (This work is supported by the NSF.)

However, we have evidence that the methods developed in our educational research are equally effective for use with adults, and we expect they will help solve problems of equipping people with skills adequate for useful occupations even if they are from backgrounds considered handicaps today.

In any case, there is a close relation between the development of AI theories and the work on human development in our group.

I. OTHER AREAS OF APPLICATION

LANGUAGE and SPEECH research will make the new systems available to non-specialists, as well as to people with busy hands!

LARGE SYMBOLIC COMPUTATIONS such as are done with MATHLAB, will make practical new areas of physics and engineering.

PICTURE PROCESSING, PATTERN RECOGNITION are of continuous concern in AI, and should continue to be productive areas.

Several of these were further explained in our Micro-Automation proposal.

SECTION 3
RESEARCH ON ARTIFICIAL INTELLIGENCE

A. INTRODUCTION

This section discusses some aspects of the state of the art in AI, that relate to the projects of this proposal. It is not a general discussion. In fact, as noted in [2.7], we are proposing to create just such a general survey and assessment; there is nothing in the literature today adequate to give a non-specialist a competent overview of what has happened in the past five years.

B. TIME-SCALE OF APPLICATIONS TO CRITICAL AREAS OF NATIONAL CONCERN

In each of a number of important fields, some of which are discussed in Project 7, conventional technology is today pushing against one or another bottleneck involving effective use of information, knowledge, or real-world interactive capability. We believe that the work proposed herein will play a vital role in each of these areas.

Our position is that AI can help with many kinds of complex programs -- because of its new ways to manage -- or at least, describe technically and formally -- kinds of interactions that in the past could be treated only empirically and informally.

We believe that many of these areas can be open to our techniques within five years -- some sooner and some later. In some cases it is clear what must be done and how long it should take. In others we have to speculate, where the problems are not entirely understood in case there may be serious unforeseen difficulties. A lot depends on how urgent these areas are seen as by the relevant agencies. And a lot depends on the availability of talented young workers in this field.

In any case, if we start on the important problems now, implementation of important, large-scale applications could begin in less than five years, given appropriate priorities. In section 2 we proposed "milestones" that suggest how certain of these areas can be developed.

IMPORTANT! We are not saying that the large-scale aplications will be in operation five years from now! We mean that by that time it will become a matter of investment and priorities. Right now, there simply would be no way to make 500 people work on implementation of an intelligent management system; the semantic structure has to be drafted first. There is no way to get a lot of people to work on an Automatic Programming and Debugging system; the representation of program

intentions and schema has to be designed first. There is no way to get a lot of people to work on a big natural language system; we first have to specify representation of scenarios, time and tense, etc., etc., etc. (We could right now, ask 500 people to help design the mechanical side of industrial robot systems, because we have a decade of complaints about inadequate mechanical devices, without good force or touch sensors, and marginal visual hardware.)

In five years, we would expect that enough testing of different representational schemes will have been accumulated to make such decisions comfortable.

Why do we say "five years"? The development of useful MACHINE VISION, along the lines we proposed around 1965, is just maturing; its first useful applications in assembly of unpositioned components are just now in progress in several laboratories. The MATHLAB project, also dating from that period in our Lboratory, is just now serving its first outside users. The current applications of NATURAL LANGUAGE processors that are just in the past year beginning to use semantic structures are in a large degree results of work of ours and others that first took form, again, in the middle '60s. It might be noted that the whole area called Artificial Intelligence originated, roughly, around 1955, in the sense that it could be distinguished from "cybernetics" and "control theory" and "operations research" by its emphasis on symbolic description and related processes. If the past is any guide, we see that the first of two decades explored many theoretical avenues. The second decade developed some of the better ideas to the point where applications could be considered. There are still many major difficulties in all areas; besides theoretical problems, it is still very hard to write and debug the large systems involved, and they run slowly and expensively on present hardware in the current implementations of the languages.

In this perspective, 10 years might seem more plausible for development of large-scale practical applications. But the members of this project, which has made rapid advances recently, expect the field to move even faster than in 1963-73. This is partly because of recent theoretical progress, but also because:

In 1963 there were only perhaps 20 full time workers in AI -- that is, heuristic programming and symbolic representation. There are now at least 200. Most major universities have AI courses now; in 1963 there were only 4 or 5.

The problems seem much more urgent in this decade, although the relevance of AI is not yet widely recognized. See the specific items below in this section.

The needed large-memory, interactive, computational facilities are now generally obtainable. In the coming decade the reductions in cost and the availability of Network Services will make

them widely available.

Technical tools are improving fast enough to have an effect soon. Automatic Programming aids of the sort being developed (see [2.3]) could help in other AI projects in three or four years.

C. IMPORTANT PROBLEMS IN AI

Artificial Intelligence, as a new technology, is in an intermediate stage of development. In the first stages of a new field, things have to be simplified so that one can isolate and study the elementary phenomena. In most successful applications, we use a strategy we call "working within a Micro-World".

In the rest of this Section, we discuss a variety of problems that arise when one develops a microworld to prove out some idea and then wishes to expand the system toward real-life applications.

Micro-Worlds

Over a long period, we have learned how ideas about intelligence could be tested and developed, and a style of research strategy has emerged. The selection of test environments is, we think, very critical, and many blind ends and traps have swallowed up workers who had not given the question sufficient thought. For example, the use of two-dimensional patterns for "vision research" led many groups away from discovering important principles about scene-analysis -- because the basically reversible transformations of a two-dimensional visual world do not require one to use symbolic descriptions. Unfortunately, this leads one -- several steps later -- away from adequate theories for learning!

A good example of a suitably designed Micro-world is shown in the well-known project of Winograd, which made many practical and theoretical contributions to Understanding Natural Language. Much of that system's power comes from the way in which the semantic system is able to represent things that happen in the so-called Blocks World. That test environment contained just the right order of complexity for the level of semantic competence that seemed achievable at the time. Winograd's Micro-World contained essentially only:

---a few kinds of (physical) OBJECTS. Each object had
---only a few PROPERTIES (color, size, location). Also, there were
---only a few ACTIONS (grasp, lift, move) available.

The interactions between these were relatively manageable -- the "problem-solver" contained in the system was "state-of-the-art" -- so that things seemed more or less within the comprehension of the

experimental language-semantic system.

Problems in Expanding a Microworld

Since the Winograd demonstration and thesis, several workers have been adding new elements, relations, and features to that system. That work has not gone very far, however, because the details of implementation of the original system were quite complex and, accordingly, it took quite a long time for subsequent workers to analyse, document, and systematize the system for use by others. (It is only now taking acceptable form.) The work of Woods (at BBN) is currently in better form for other applications; this is partly because its structure was from the start more uniform and systematic, demonstrating (we think) just the kinds of trade-offs being discussed. In the work of Winston's group on the "visual" Blocks World (the extension of the original MicroWorld, which was the "Polybrick" system of Guzman's thesis) progress toward including more and more has been steady.

The Need for a Knowledge-Structure Theory

Obviously intelligence is not merely having knowledge. One needs good ways to decide what is relevant to a situation -- Intelligent Information Retrieval. Within the Microworlds that have been studied in AI research, many problems and proposals have been examined, and these have led to some powerful techniques. At first, many workers hoped it would be possible to separate the problems of reasoning and deduction into:

A basis of factual knowledge
A (possibly complicated) procedure for using the knowledge

But the separation leads to serious problems. Too much of one's knowledge is concerned precisely with which (other) knowledge should be applied to various kinds of situations. Unfortunately, this turned out to be harder to "represent" than did "ordinary" factual knowledge.

We note that at the MIT AI laboratory, this problem is taken much more seriously than at most other research centers; this is why we do relatively more work on designing advanced knowledge-based systems and less work on trying to extend the "logical theorem-proving programs". While we agree it is of some importance to find out how far such systems can be made to go, we don't think they will ever go far enough to solve most important practical problems.

In particular, we think that some extremely important kinds of knowledge (vital to solving any difficult problem) were overlooked almost entirely, in the early days. These are the "facts of life" about interactions, bugs, bottlenecks, etc., which every child comes to know.

We believe that now that the area is recognized as a manageable technical subject (see [5.3]) there will be a large change in the capability to apply larger bases of information to hard problems. Most important, we think, in the next year or two is to get more work done on understanding "types of bugs" -- as it is called in Sussman's thesis -- and apply these ideas to automatic programming and debugging applications.

Representations

It is generally agreed, also, that success in problem solving depends on finding a good way to represent one's knowledge so that it can be made to fit the problems one is trying to solve. That this is true is shown rather clearly in the history of mathematics, in which "mere notational" issues made very large differences in practical effects, viz., the power of the differential calculus notations, or the modern vector representations.

Nevertheless, although the importance of adequate representations is now generally recognized as central to AI, we think it is important to remember that the classification and isolation of knowledge is not yet adequate. The recognition that there are many "types of bugs" -- that is, that there are different kinds of bad interactions between simple processes -- which require different kinds of interventions, was not so much a matter of representation as of recognition of the problem.

Combining MicroWorlds

People ask: now that you have programs that do "this", and ones that do "that" -- why don't you combine them all to get one program that is much more intelligent?

We cannot usually do this. Sometimes the difficulty is simply that the programs use different representations -- they can't communicate in any one language. The cure for this is usually to try to rewrite both. Another approach, in principle, is "modularity"! Make all your programs out of compatible modules. In electronics, this is pretty easy; make sure all inputs and outputs are "TTL-level compatible; make all power supplies run at 5 or 15 volts.

The trouble is that the idea of modularity, itself, is not very appropriate to intelligent systems, in which the problems of interaction are the important and difficult ones. When two "knowledge-modules" are connected, one has to put -- somewhere -- a key to the most likely interactive types of bugs that can be expected to emerge.

It remains to be seen whether such uniform approaches as Hewitt's ACTOR module (see [5.3]) will yield substantial gains here, or the Predicate

Calculus formulations of McCarthy et al. at Stanford. We are quite confident that there are important immediate advances to be obtained from the self-annotating self-debugging approaches proposed in [5.3], over the next few years, and that these have immediate as well as long-term applications.

The Blocks World -- Success in Combining Two Micro-worlds

By something of a coincidence -- certainly for quite different reasons -- our research on Vision over the past few years had also converged on a Micro-world of the very same sorts of simple physical geometrical objects and actions, and we use the same "Blocks World" caption for the experimental vision environment that was used for that work. We note in passing that many visitors and readers missed the point rather badly, and were skeptical that anything very realistic could be learned by working with perfect simple shapes, no textures, no noise, no complex curves or "soft" surfaces. But we claim that through this careful, sometimes tedious analysis we laid the foundation for the main lines of the successful systems now being developed and demonstrated; certainly at the higher levels of scene-analysis (where all contemporary groups are now following our general outline) and (though to a lesser extent) even on systems that deal with poor, noisy, textured situations. Basically, we took the position that

> "One should not directly attack the problem of recognizing a pattern immersed in noise, until one is able effectively to recognize it in the clear".

This may seem to be common sense but experience shows that most beginners feel it too simpleminded to take seriously.

We note in passing that our own recognized progress in non-trivial machine-learning is in large part due to a related principle; as McCarthy has put it,

> ...one should not attempt to make a machine learn something unless one is sure there is some way to tell it that thing ...

i.e., that the machine has access to an internal representation that can in some natural way encode the desired information. Without adequate representation one can do little, with it, other difficult problems often melt away. Thus, in Winston's thesis, once we had an adequate representation for DIFFERENCES between other descriptions, several interesting kinds of learning became easier to program, and some reasoning-by-analogy was easy to incorporate.

On the other side, the same principle has been used to show that one very popular approach to building learning machines was inherently defective. The core of our results in the Theory of Perceptrons was

based on separating the questions about learning from those about the machines' representational capabilities; then, when the latter were shown to be inadequate, we were able to prove (PERCEPTRONS, MIT Press, 1969) that no series of "training sessions", however prolonged, could make the poor machine learn its lessons. And, as a by-product of that analysis, we did discover a number of limited, but interesting and useful tasks that perceptron machines could in fact be made to do.

Uncertainty

There are a number of areas in which present foundations are not quite strong enough to support the proposed applied projects. These must be studied further, in smaller micro-worlds.We have to get better control of the areas briefly discussed below in order to bring the applications from the "controlled-environment demonstration" phase to the real application requirements.

Types of Uncertainty

As AI programs get more ambitious in the size and scope of their knowledge bases, we will encounter more and more situations in which conclusions are unsure. This is not a new problem; uncertainty is familiar even in the most highly defined situations with no chance element. For example, in Chess, one must "take chances".

Taking chances does not, however, mean using probabilities! The best game-playing programs have not used probability models in their performance. (Whether they use such theories in their conception and construction is another matter entirely, and one that is hard to settle or evaluate. Thus, in Samuel's Checkers programs, one can argue both sides of whether probability is involved. Certainly, a close relative of mathematical correlation is used.) In any case, the new areas will be necessarily engaged in more "plausible inference" and evidence-gathering activities. In the course of that research, we can expect to face and develop ideas about decision under uncertainty; we predict that the results will be quite different from classical decision theory and classical utility theories. On the other hand the results must also be practically effective. It will be interesting and, we hope, valuable, to see how these theories will relate to the traditional decision-theory, game-theory, and economic-theory models.

Similar problems appear in more immediately important areas. In Automatic Debugging, one has to face the problem of plausible inference of intentions and plans in a program -- see [Appendix 5.3.1 -- unless the programmer has already spelled this all out.

In any case, there are other areas of uncertainty. The quality of evidence depends on its source. One needs policies and methods of

accounting for authority, reliability, and strength of conviction.

In Chess, one uncertainty comes not so much from lack of information but from too much; the impossibly large full search tree is too large to examine and one must adopt some strategy that uses less information. Note that the uncertainty, in Chess, is in part that one does not know the opponent's strategy. To cope with that kind of situation one does best to build some sort of MODEL of the opponent (or, more generally, of the environment one is in). But different problems, in such a fix, depend on different ways in which one set up that model, and one should make provisions for noticing at least some such dependencies. We believe that the new program-accessible comment schemes like those in Sussman and Goldstein's theses, will show ways to do such things.

Qualitative-quantative issues

In controlled demonstrations we have found that we can usually sidestep problems of estimation and uncertainty by a variety of heuristic devices. There is some reason to believe that this can be done quite generally and, frankly, most of the project scientists do not believe that humans ordinarily use (in their heads) the kinds of probabilistic decision and utility models that have occupied the primary attention of most "quantitative social scientists". Certainly, we still feel this way about the use of "analog" models for visual imagery -- another area in which the majority of psychologists think "quantitative" machinery must be required. We believe this is a mistake based on unfamiliarity with the power of "symbolic-descriptive" mechanisms that have evolved in work on AI. The same may be true in decision theory, and the problem is beginning to face us as we begin to deal with much larger varieties of kinds of information within the same system. We expect, for example, that the work on Chess, which features the program's necessary ignorance of the opponent's plan, will clarify this issue, and our planned work on the Management Assistant project, which also will range over a highly inhomogeneous data base, will also have to face such issues.

There are many other ways in which qualitative issues enter common sense situations. Everyone knows ways to use qualitative quantities; if A is very near B and B is very near C then A is (at least) near C. But one can also say that A is very near C under usual conditions. Obviously one cannot iterate this very many times.

Such issues are implicit in many of the current projects. They are also important in the Medical Assistant project area (which we expect NIH to support) that also concern some of the same Laboratory staff members. We expect a substantial interchange of efforts in this qualitative-quantitative representation area between the workers of this proposal and with our colleagues W. Martin, A. Gorry, W. Schwartz, and others in the proposed NIH project.

Extended Events, Frames, Scenarios

We believe that the problem of representing real world knowledge will have to be faced by dealing with much larger chunks than our colleagues have believed necessary. We feel that the problems that beset the "theorem-proving" projects and, generally, all those using methods related to the formalisms of traditional "Mathematical Logic" come from the limited resources such systems have for representing the interactions within highly structured real systems. Our knowledge about the real world, or about those subjects in which we are particularly competent, is not a bland, uniform structure of simply-interconnected "facts". We envision it as more like real geography; houses are not equally near one another, but are arranged -- for good but intricate reasons -- in towns, cities, metropoli; these have centers and capitols of many different sorts. The road system reflects this structure more or less perfectly. Similarly (though the simile soon becomes fatuous) our knowledge is made up not of similar knowledge about many different things, but of elaborate constructions about a few things. Plus -- and this is vital -- higher level data about how other less intimately familiar systems are similar and how they are different. Thus, the main tool of reasoning -- in the opinion of the principals of our project -- is not regular deductive logic but, rather, procedures for drawing analogies and for making plans to carry out their suggestions.

The elements of this kind of knowledge might then resemble stories rather than axioms; scenarios rather than snapshots, typical-examples-plus-advice about applications rather than "general principles". Our first attempts to use these ideas will be in the areas of Natural Language and Scene Analysis, but the general idea is already influencing plans for the other applications projects.

Heterarchical Programming

Programming and debugging are getting too hard. This is because the systems are getting larger and more complex. They use, in a single system, many different kinds of information, and the interfaces which translate between one representation and another add to the complexity of the system. Fortunately (1) better understanding of heterarchical systems is rapidly growing. (2) We frankly expect that our recent progress on Automatic Debugging will help -- perhaps not directly on the programs themselves -- but certainly on the new style of internal documentation suggested by the phrase "program-accessible representation of programs' INTENTIONS".

Goal: Programs with "common sense". One should be able to ask a program why it did something, how it works, why it said something.

To answer such questions, the program has to contain a representation of its own activity -- in terms of 'intentions' or higher-level goal-oriented comments, rather than or in addition to the usual "declarative" or "procedural" program-language description of the process to be carried out.

Indeed, we believe that such an "explanation", or "excuse" language is necessary not only for making sense of a program's activity and operation --- for intelligent application, debugging, and extension -- but also for developing any large program in the first place. But usually, at least today, the goal oriented description exists only informally in the programmer's mind.

In Sussman's thesis we see some steps toward this. When a test program shows a "bug", a higher level program tries to explain it, perhaps in terms of side-effects of conflicting actions, or side effects of competing goals. If, for example, the requirements for concurrent goals interact, so that fulfilling the conditions for acheving Goal A make it impossible to achieve Goal B, it may suffice simply to achieve goal B first. But the program will only think of doing this if it can explain the failure in terms that suggest that those goals are in fact closely associated and therefore candidates for swapping.

SECTION 4
THE ARTIFICIAL INTELLIGENCE LABORATORY

A. INTRODUCTION

The MIT AI Laboratory has evolved from a small group of students working with John McCarthy and Marvin Minsky in 1958. It was originally part of MIT's Research Laboratory of Electronics and the MIT Computation Center. It joined in the creation of Project MAC as the Artificial Intelligence Group, and became a separate MIT Laboratory some years later. During the period of ARPA support, it has been co-directed by Marvin Minsky and Seymour Papert.

In this section we describe some of its activities accomplishments and problems.

B. SYMBOLIC APPLIED MATHEMATICS

The first serious project on getting a computer to work with literal formulae and real mathematical principles rather than numerical calculations was done here by Slagle, whose 1961 Ph.D. Thesis showed how a heuristic program could compare favorably in performance with a better-than-average MIT first-year student on symbolic Integration. The program could not deal at all with the context of such problems, so that it was useful only once the problem had been put into the right form.

This demonstrated that heuristic techniques could deal with symbolic mathematics in a rather natural, lifelike manner. The behavior of Slagle's program resembles rather closely that of mathematics students who are very intelligent but not particularly expert in that area.

Some years later the same problem -- symbolic indefinite integration -- was attacked by Joel Moses. His thesis exhibits highly expert performance over a much wider range than did Slagle's. Moses' program demonstrated publicly that we were at the stage where AI methods could produce a mathematical "assistant" that could really extend the capability of a serious user.

At the same time, W. Martin addressed himself to other problems in the area of symbolic applied mathematics; to making theories of representation, interaction, simplification, and so forth. The success of the two theses of Martin and Moses suggested joining forces, which led to the MACSYMA system -- or the MIT MATHLAB project -- that is now an independent part of Project MAC.

Since then, many other centers have attacked parts of the problem, and this field is just now becoming a significant addition to the tools of the scientific world in attacking large, semi-quantitative problems in

applied mathematics, engineering, and physics.

Another, parallel development in this period has been the work of Bobrow and Charniak. Bobrow developed a heuristic program, STUDENT, that worked on less symbolic, more realistic "word problems" at the high-school algebra level. In this work, the emphasis was not so much on formal mathematics as on semantics and natural language. Bobrow showed that by using a semantic model of what sentences (probably) mean, one can sidestep issues in linguistic theories that seemed very serious if attacked without reference to meanings. The success of this prototype had a noticeable effect over the next few years in redirecting the attention of computational linguistics to the practicability (and, we think, the indispensibility) of involving meaning with syntactic analysis.

Years later, E. Charniak produced a Master's Thesis in which some word problems in Calculus could be handled. This thesis raised new issues; it became clear that as Charniak attempted to include in his scope such issues as time-analysis, simple mechanical statistics and dynamics, and properties of common materials (you can pull with a rope, but not push!), the earlier Micro-world of simultaneous linear equations and phenomena that had sufficed for Bobrow's problems was not adequate. Indeed, the extension of symbolic applied mathematics to non-symbolic real-world problems still awaits refinement of common sense knowledge-structures. We hope to fill this gap as a result of work on "qualitative physics" [see 5.6].

C. VISION

There is too little space here for an adequate summary of the AI Laboratory's work on real-world vision. There will appear a survey in P Winston's paper in the August 1973 Proceedings of the International Conference on AI, Stanford, 1973.

Briefly, we claim that it was primarily our line of emphasis that has created the contemporary atmosphere of optimism and accomplishment in Machine Vision. The main points were:

Emphasis on Symbolic description rather than Picture-Transformation
Emphasis on Heterarchical use of real-world knowledge
Emphasis on problems of 3-D occlusion and figure-ground separation

This work resulted in several internationally known papers, notably the Ph.D theses of A. Guzman, P. Winston, and D. Waltz.

The "low-level" problem of finding real physical features in picture scenes has been a very difficult one; important contributions were made in work by J. Holloway, R. Greenblatt, A. Griffith, T. Binford, B. Horn, and A. Herskovits in our Laboratory. Three Ph.D theses on other Vision

topics were written by A. Griffith, B. Horn and L. Krakauer, exploring different mathematical and structural models of such topics as shading, contrast, illumination gradients, highlights, focus maps, etc.

The outcome of all this was the modern approach to vision generally called "Scene-Analysis" -- a term that serves mainly to distinguish the approach from its predecessor, usually called "visual pattern-recognition".

D. NATURAL LANGUAGE

In our Laboratory, the work on Natural Language developed in the context of understanding common sense semantics rather than from the milieu of classical linguistics. The work of Bobrow, noted above, and the work of B. Raphael on a question-answering program that used contextual clues to solve problems of semantic ambiguity were the first attempts to take this approach. The work of Charniak, as noted above, was a next step, but showed the need for a more carefully developed Micro-world of context. In Winograd's thesis we saw these and many other ideas from linguistic theory brought together and applied to a more adequate symbolic Blocks World that resembles (superficially) the one used in our early vision research. Most important, perhaps, were Winograd's innovations in making the syntactic-semantic interactions actually work in a heterarchical programming system; a dream that had never been realized effectively because of inadequate technical understanding (and courage).

Semantic Information Processing is now one of the most active fields of AI, and promises to yield systems useful in many practical areas.

E. ROBOTICS

In the history of efforts to understand how to make autonomous physical assistants, the AI Laboratory claims a very special role. The first such system, so far as we know, was the tactile-sense heuristic program of H. Ernst which, in 1961 was able to search for different objects on a table and assemble them into a tower or put them into a box. Some years elapsed, in which we made plans to move in the direction of visually-controlled manipulators, and finally, in the mid-60's R. Gosper, using programs based on earlier work by J. Holloway, R. Greenblatt and S. Nellson, was able to demonstrate the first completely autonomous hand-eye system that could analyze a scene well enough to locate non-overlapping blocks and assemble them into a simple tower. In 1970 P. Winston, B. Horn and E. Freuder demonstrated their "copy" system, which looks at one scene of blocks and then assembles a copy of it from spare parts in another part of the table.

Next, there was a period of activity, during which we were publicly rather quiet, to rebuild the system. Our first system (and those of our colleagues at other centers) only "appeared" to see well; in fact they were impossibly sensitive to mistakes because of occlusion of part of one object by another. Also, they were horribly sensitive to picture-processing errors due to not-quite-perfect illumination, or flaws on the surfaces of the objects. It was only in the past year or so, with the completion of Winston's heterarchical Vision System, and using ingredients like the Guzman-Winston-Waltz object-finding theories and the Shirai heterarchical edge-analyser, that we could get performance good enough to suggest moving on outside the closely controlled Blocks World visual environment. The current system can cope with very complicated occlusions of objects in three dimensions, using a single (non-stereoscopic) viewpoint. As noted in [5.1], we plan to use range-finding and other more powerful "vision" methods in the more complicated real-world applications.

F. LEARNING

In the early years of our work, we tended to avoid attempting to make machines that were supposed to "learn to become smarter", following this principle: don't try to make a machine learn a class of behaviors until you are sure that the machine or procedural structures available are capable of supporting that behavior. This turned out to be a good idea, since little progress was seen, in the first decade, by those attempting to get significant learned behavior by using Neural Network, Perceptron, Evolutionary, Adaptive-Adjustment, or Inductive-Inference theories.

In fact, one of our most substantial theoretical accomplishments was to develop the definitive theory presented in the book, PERCEPTRONS, showing why certain low-level linear optimization machine structures cannot learn to account for interactions of first level cues, features, or context-dependencies.

By 1970 our general understanding of structural representations had progressed to the point of Winston's thesis, in which we see a relatively high-level form of learning, in which what is learned depends on comparing descriptions of current events with summary descriptions of what has come before. This work has evoked widespread interest, and we see many projects, here and around the world, moving in that direction.

Many important things we learn are not mere facts, but knowledge-handling capabilities, in particular, procedures that make for better learning strategies. The recent work of Goldstein and Sussman, which might appear to be concerned chiefly with Automatic Programming, is really concerned with modifying procedural representations of knowledge in accord with the extent to which the procedures in fact behave as they are "supposed" to. We regard this, then, as the most promising path presently available toward really "adaptive" machines.

G. COMPUTER LANGUAGES

The AI Laboratory has played an important role in the evolution of the programming languages and computer systems that are used today by most successful workers in the AI field. The LISP system, originally developed by J. McCarthy, has become, in effect, the "standard" international language of AI. The AI Lab's LISP 1.6 system is probably the most powerful version of LISP, and is certainly in more extensive worldwide use than other variants. In its current form, one does not have to pay the "traditional" overhead of interpretative operation for procedures that do not require it; it is believed that its numerical efficiency is comparable to that of most known FORTRAN or PL-1 compilers! This LISP 1.6 has a powerful compiler, and an extremely versatile READ (input-output) system, both due largely to Jon White, extensive interrupt system, and many other features. It meets practical compatibility requirements. The present version can run on any DEC 10/50 monitor. Less sophisticated versions run on IBM systems.

It was discovered, however, that the data-structures originally provided in LISP were not really adequate for some of the "representations of knowledge" that AI was beginning to need. For example, Bobrow needed a form of "pattern-matching" for his linguistic system, and invented the language METEOR, embedded in LISP, so that he could have the features developed earlier by Yngve in his self-standing language, COMIT. At about the same time A. Guzman, working with H. McIntosh developed a language CONVERT, also embedded in LISP, initially for work in symbolic applied mathematics, a field that McIntosh had conceived about the same time we did. Independently, the Newell-Simon group at Carnegie also come to the conclusion that pattern-directed program control might make important advances in behavioral theories.

None of those languages seemed to "stick". Perhaps they were inadequately engineered, perhaps they were before their time, certainly none of them had clearly formulated theoretical foundations of the quality that the original LISP of McCarthy had. But recently, the PLANNER proposal of Hewitt, as implemented in the MICRO-PLANNER language used by Winograd (and many others, now) seems to have met the real-world conditions for acceptance. Its descendant, CONNIVER, has recently had the most favorable acceptance in this family, but only the next few years will tell what is going to serve best.

In other centers, new languages like QA-5, STRIPS, and SAIL are also moving in this direction -- to make available pattern-directed control and to allow reference to multiple processes, multiple (bound) environments, "IF-NEEDED" and "Antecedent-Theorem" demon-like processes and the like. These are all, we believe, reponses to the need for heterarchical control of knowledge-based problem-solving systems.

H. TIME-SHARED COMPUTER SYSTEM

The MIT AI Laboratory developed the first time-sharing system for the PDP-10 (then PDP-6) system. This system, called ITS, was essentially completed about four years ago. It has recently become overloaded by the increasing complexity of the jobs it has to do.

Many people agree that for AI purposes the ITS system is probably the most effective time-sharing system available. However, because we do not want to be in the service business, we are interested in accepting a larger system, even if not quite so effective, if the maintenance can be provided elsewhere. However, this is not quite possible at present, as will be noted in the section below about our computation requirements.

I. CONTRIBUTIONS TO FUNDAMENTAL THEORIES

The project has made many fundamental contributions both to Artificial Intelligence and to modern computer science in general. We will not review these in detail, but just list their names.

Mathematical Theory of Computation: J. McCarthy, et al.
Theories of Schemata and Program Control: Hewitt, Paterson, et al.
Abstract Complexity theory: Blum
Perceptron Theory
Concrete Complexity Theories: Minsky, Papert
Contributions to theory of parallel computers.
Computational Geometry
Contributions to theory of Productions, etc.

J. COGNITIVE PSYCHOLOGY

In the very last few years, it has become recognized that AI research is much closer to psychological research than was at first believed. The theories in the Vision System are widely considered as candidates for theories of human vision. (See Sutherland's essay in the British SRC volume.)

The more general view, that what is learned is in large part determined by knowledge-based processes (as in Winston's thesis) seems relatively new in psychology -- although there is rarely anything really new in the sense that one can find such proposals in earlier literature -- and is attracting a great deal of current attention.

The AI-based theories of education, as demonstrated in our LOGO project, have had a big impact, recently, on the community of educational theorists.

K. AUTOMATIC PROGRAMMING

This area is historically very close to AI. Indeed, the first really thoughtfully human-engineered debugging systems (like DDT) arose in the community loosely associated with AI work. J.C.R. Licklider was an early promotor of such ideas. We believe that our recent work -- notably of Sussman and his colleagues -- has been the primary catalyst in the new wave of interest and optimism (e.g., as shown by the Balser report).

L. SPEECH RECOGNITION

The revival of interest in this area for probably feasible applications stems perhaps as much from the advances in AI-types of heterarchical (and semantic) programming as from any major advance in speech-science proper.

M. OTHER SUBJECTS

The complete summary of so large a Laboratory over so many years would take too much space here. We mention a few other topics of importance in their own right:

Work by Daniel Edwards on Cryptanalysis
Work by Bledsoe, Abrahams, Levin, Silver, Minsky, Norton, Slagle on theories of Mathematical Theorem-Proving
Work by Papert on theories of development of Intelligence
Work by A. L. Samuel of the well-known Checkers Program
Work by J. McCarthy on Chess
Work in many areas of applied mathematics and numerical analysis by R. Gosper, R. Schroeppel

N. HARDWARE: OUR MEMORY EMERGENCY

The Laboratory needs an increase in computational power. Because of the knowledge-based character of the new generation of programs, their size is large compared to programs of five years ago.

This is not a transient. All reasonably intelligent programs will be pretty large from now on. The practical cost of this largeness is not serious, except for the very immediate future, because all industry predictions agree that the cost of, say, a 500,000 word primary storage will approach a few thousand dollars in the next decade. But right now we are in a crushing bind, because our time-sharing system can support only one such program at a time.

History: We contracted and supervised construction of the first fast mass core-store -- through a development contract with Fabri-tek. Although it was at first a powerful research tool, we have suffered since, by being "stuck" with it -- the reward of pioneering. ARPA has put aside our request for modernization from year to year. It has become a critical bottleneck in achieving our goals.

There are several problems with it.

(1) It is dangerously marginal. If there is a major failure, no one will be able to fix it; it has been out of production for a long time and we have maintained it ourselves.
(2) It is slow. Modern PDP-10's have 1 microsecond memories. The Fabri-tek memory is 2.9 microseconds.
(3) It is too small. Large programs saturate it and cause the system to go into "thrashing" mode of paging onto disc. MATHLAB, an offshoot of our Laboratory, has already found it necessary to add another 250K of memory. This has made it possible for them to run several knowledge-based programs at a time. Our system is swamped by one program of the size of Winograd's Natural Language Program. We will find it very difficult to debug the application programs unless the memory is expanded.

All of the system development has already been done, since the AI and MATHLAB machines use identical systems. The additional memory will also allow Network users to try out our application programs. At present, this cannot be done.

The system is extremely slow when any CONNIVER or similar program of the order of 100K is run. We use shared pure procedures as much as possible, and common compiled code, but the CONNIVER control stack must be represented as list-structure.

SHRDLU, McDermott's reasoning program, Lavin's Vision program, and several other important research programs are too large to run on our current hardware when other users are competing for time. To make reasonable progress, we would expect each of the project areas to have 1 or 2 programs running at all times!

Doctoral Students from the A.I. Laboratory

The Laboratory has an interesting academic record, as evidenced by this listing of all of its doctoral degree recipients.

Slagle	NIH. Head, A.I. Research group
Jones	NIH.
Norton	NIH.
Hodes	NIH.
Blum	Prof, Berkeley
Raphael	Head, robotics research, S.R.I.
Bobrow	Head, A.I. and language, Xerox (formerly BBN)
Teitleman	Xerox
Abrahams	Prof, N.Y.U.
Winston	Prof, MIT, Head of Robotics
Winograd	Prof, MIT (1973, Stanford)
Hewitt	Prof, MIT
Horn	Prof, MIT (1973)
Krakauer	Industry
Griffith	Industry (Information International)
Evans	Industry (private software company)
Guzman	Prof, Polytechnico, Mexico
Waltz	Prof, Univ of Illinois (1973)
Charniak	Staff, Istituto Per Gli Stui Semantici E Cognitivi, Switzerland
Moses	Prof, MIT Head of MATHLAB, MAC
Martin	Prof, MIT Head of Automatic Programming, MAC
Luckham	Prof, Stanford
Smoliar	Prof, Technion (Israel), Univ. of Penn (1973)
Sussman	Prof, MIT (1973)
Goldstein	Prof, MIT (1973)
Abelson	Prof, MIT (1973)
Perkins	Lincoln Lab.
Henneman	Prof, B.U. (Formerly,Univ of Texas)
Beyer	Prof, Univ of Oregon
Fell	Prof, Northeastern Univ.

Other Ph.D. students who did their work at AI Lab.
Baylor (from Carnegie)
Beller (from Brandeis)
Roberts (from Carnegie)

Closely associated Ph.D. Theses:
Ernst (IBM); Roberts (ARPA); Sutherland (Utah); Fischer (Waterloo); Knowlton (BTL)

Extended residencies:

Bledsoe (Chairman, Mathematics, Univ of Texas)
Cocke (IBM research)
Voyat (Genevá) Prof. CUNY Graduate Center
McConkie (Cornell)
Marr (Cambridge)
Paterson (Cambridge)
Nevins (current)
Rabin (Hebrew Univ.)
Forte (Chairman, Yale Univ. Music Dept.)
Slawson (Chairman, Music, Univ. Pit.)
Binford (vision research, Stanford)
Samuel (IBM Laboratories)

Our Laboratory has very close ties to the AI groups at
Bolt, Beranek and Newman
Stanford
Carnegie-Mellon
Stanford Research Institute
Edinburgh

SECTION 5
SUMMARY PROGRESS REPORT

MACHINE VISION

Work on machine vision has progressed rapidly in the last few years. Many basic issues are now more sharply defined, permitting us to focus outside the restricted world of carefully prepared simple polyhedra.

We here summarize some of the progress with emphasis of the last year. At the "performance" level, we can take a collection of flat-sided objects of assorted shapes, pile them up, and ask the program to analyse, disassemble, and rearrange the objects into another structure. The latter can be specified by a symbolic description or by presenting a physical example to be analysed by the system. Many "low-level" vision problems had to be solved to reach this level of performance. Many of them are summarized in our January, 1972 Progress Report, and much more detail is available in technical notes and reports. We have made no compromises in our original long-term goal to set a firm foundation for Monocular machine vision! This vision system works as well on pictures of a scene as it does on the physical scene itself. It is not based on the use of physical range-finding methods, tactile-probe exploration, or other "active" sensors.

The following particularly noteworthy results have appeared in the last year.

1. David Waltz has worked out a semantic theory of polyhedral line drawings that is a major breakthrough in several respects. The theory gives deep insights into the success of earlier work and provides a powerful analysis capability for separating regions into bodies; in identifying edges as convex, concave, obscuring, shadow or crack; in using shadows to determine contact; and in reasoning out the orientation of object faces.

2. Previous vision systems suffered from an artificial division into line-finder/scene-analysis partnerships, communicating only by way of a handed-over line drawing. The new systems of Jerry Lerman and Yoshiaki Shirai show how the barrier can be eliminated and how high level knowledge of physical constraints and partial analysis can guide the filters and trackers that most intimately deal with low-level intensity information.

3. Tim Finin has given the evolving vision system considerable deductive depth through several goal-oriented programs. One of these specializes in using a theory of "perceived groups". Often, some of an object's individual dimensions, position, or orientation parameters are indeterminate because of an obstruction in the line of sight. In these situations the vision system hypothesizes the missing information, using other objects considered similar by virtue of alignment in a stack, a common purpose, or simple proximity.

4. Finin, Lerman, and Slesinger have completed a visual feedback module that checks the position of a block after positioning by the hand. Then it jiggles it into place if its positional error exceeds a small threshold. This feedback link exploits the random-access capability of a programmable image acquisition system by looking only at points lying on a small circle around expected vertex locations.

5. Bob Woodham has done initial work on visual motion tracking. As a first step in effecting a coffee-pouring demonstration, he has worked out and compared several mechanisms for monitoring the rising level of coffee in a stylized cup.

6. Scott Fahlman has devised a construction planning system which solves problems in two distinct directions. First, three dimensional modelling skill has been developed in the form of sophisticated touch and stability tests. Second, in cooperation with the specialists in CONNIVER language, he has demonstrated the need for and use of advanced control and data base mechanisms. The system can plan fairly complicated constructions requiring temporary scaffolding supports or counterweights.

7. Rich Boberg has explored the problem of reversing the analysis process, that is, reconstructing a scene from an abstract description. We believe this is the first step toward an automatic design system where the machine contains and uses considerable common sense knowledge about the constraints inherent in a physical world.

8. John Hollerbach has probed the problem of describing complex shapes through work on complicated, higher order polyhedra. His heuristic theory of projection shows how many objects can be sensibly decomposed into basic shapes, modified by protrusions and indentations.

9. In another domain, Mark Adler has shown how to make progress toward solving the problem of line drawings with curves. In a style reminiscent of initial work on polyhedra, he has outlined an approach to the analysis of some highly constrained kinds of drawings. This should contribute conceptually to work on more general real vision, to diagram reading and manipulating services, and eventually to personal assistant systems in which sketches must supplement commands in natural language that do not lend themselves to verbal explanations.

PROGRESS IN MACHINE LEARNING

The long sought goal of machine learning has seen a major conceptual breakthrough. The concept is simple: we say that someone has learned something if he is able to perform appropriate processes. One rarely builds a new process from nothing; presumably one extends and adapts processes developed for other goals. Thus, learning is like debugging a computer program, and a smart person is one who knows good ways to characterize defects and requirements, and has good methods for making the appropriate procedure changes. This idea, in different ways, has led to the following large steps.

1. G. Sussman has completed a computer program that contains some sophisticated knowledge about diagnosis and repair of computer programs. Given a sequence of block-building tasks, similar to those performed by Winograd's natural language program, Sussman's program performs a sequence of modifications on an initially trivial building program. Eventually, the "learned" procedure is able to perform all the operations that were initially built-into Winograd's system. Sussman's methods have attracted wide attention, even before publication of his thesis, and are the basis of a number of other automatic programming proposals.

2. I. Goldstein has developed a theory of automatic analysis and systematization of programs which propose to achieve certain kinds of goals but do not actually work. By setting up an "annotation" structure based on matching parts of the defective procedure to parts of the goal description, Goldstein's procedure can propose and make corrections, using methods similar to those of Sussman. Again, part of the "secret" of learning lies in having the knowledge and ability to focus clearly on what parts of procedures are not working properly, and Goldstein's thesis is, we believe, a major move in this direction.

3. M. Minsky has formulated a new theory, called Frame-Systems, which, it is hoped, will show quickly how to do complicated common-

sense reasoning in systems that are not confused by containing large amounts of non-relevant information. This theory also suggests a number of ways in which new knowledge can be added in an orderly way, again without leading to information overloads. It is hoped that this theory will combine with Winston's earlier learning program ideas to allow operation over a wider range of tasks.

MICRO-AUTOMATION

We have now selected and acquired a computer configuration, a vidicon system, and a digitally driven x-y table. A new arm designed for us is under construction for November 1973. A new computer eye design is being completed. It is expected to have much better optical properties than previous computer eyes.

AUTOMATIC PROGRAMMING

We mentioned above the projects of Goldstein and Sussman: Program Analysis, and Automatic Debugging. Another project which is well under way, concerned with developing a programming formalism and technology, is the ACTOR system of Hewitt and his associates, in which implicit and undesirable interactions are unlikely to arise accidentally.

NATURAL LANGUAGE RESEARCH

Winograd's BLOCKS program demonstrated new dimensions to the connections between details of the structure of natural English and the meanings of words, clauses, and whole discourses. Our experience with SHRDLU has had two important consequences: it has shown us that quite non-trivial natural language programs can be written with today's hardware and software, and this in turn has encouraged a fresh burst of natural language research, not only here but in other laboratories.

The current proposal summarizes recent work on consolidating what was learned from this work, and what has been done with it recently. The system has been revised and documented so that it can be used by others who want to go further in this area.