MASSACHUSETTS INSTITUTE OF TECHNOLOGY
A. I. LAB

Artificial Intelligence
Memo No. 185 December 1970

PROPOSAL TO ARPA FOR RESEARCH ON ARTIFICIAL INTELLIGENCE AT M.I.T., 1970-1971

Marvin Minsky and Seymour Papert

Work reported herein was supported by the Artificial Intelligence Laboratory, an M.I.T. research program sponsored by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-70-A-0362-0002.

Reproduction of this document, in whole or in part, is permitted for any purpose of the United States Government.

PROPOSAL TO ARPA FOR RESEARCH ON ARTIFICIAL INTELLIGENCE AT M.I.T., 1970-1971

Marvin Minsky

Seymour Papert

1. INTRODUCTION

1.1 Objectives

The M.I.T. Artificial Intelligence Project has a variety of goals, all bound together by the search for principles of intelligent behavior. Among our immediate goals are to develop systems with practical applications for:

1. Visually-controlled automatic manipulation and physical world problem-solving.

2. Machine understanding of natural language text and narrative.

3. Advanced applied mathematics.

The long-range goals are concerned with simplifying, unifying and extending the techniques of heuristic programming. We expect the result of our work to:

- make it much easier to write and debug large heuristic programs;

- develop packaged collections of knowledge about many different kinds of things, leading to programs with more resourcefulness, understanding and "common sense";

- identify and sharpen certain principles for programming intelligence. These will be discussed further on, so we will only indicate some of them here:

  1. Heterarchical control -- better communication between processes that use different kinds of knowledge;

  2. Modification of hypotheses by "learning from mistakes";

  3. Control of processes by "demons" -- dormant processes that watch for certain kinds of events and intrude to modify the main processes and emphasize relevant data;

4. Implementation of these ideas by using the new kinds of primitives available in the new PLANNER language (see §2)

We hope these concepts will lead to programs that are less "robot-like" and open the way toward more capacity on real (instead of toy) problems.

This orientation -- already underway for about half a year -- is being directed toward hard problems in the following specific application areas:

Vision: building a practical real-world scene-analysis system.

Mathlab: making a system that really extends the effectiveness of Applied Mathematicians. See our 1969 Progress Report.

Narrative: handling "common-sense" understanding of descriptions of sequences of actions and conversations. See §4 and §6.

Robotics: more advanced handling and planning constructions; coordination with tracking of moving objects; connection with full power of the natural language system. See §3 and §4.

The project will also continue its various theoretical activities in computational schemata, computational geometry, automata, and complexity theory, often in close connection with the area of application. Here, the goal is to advance our fundamental ideas about, for example, the compromises between speed, cost and complexity of hardware with respect to alternatives in:

serial versus parallel machine organizations;

memory size versus computation time;

digital versus analog hardware elements;

special knowledge versus logical generality.

1.2 Strategy

We believe the most important problems in the field of Artificial Intelligence are centered, today, around problems of using many different kinds of knowledge in the same system. We think that working on such problems is the most direct path toward finding out how to build systems with greater "generality" -- systems that do not need to be rebuilt whenever the problems they have to solve are changed slightly. It is also the most direct path toward programming the kind of "common sense" needed not only for problems generally recognized as "hard" but also for the kind that people solve routinely -- even unconsciously -- in understanding the meanings of ordinary language.

In the past few years, the art of heuristic programming has developed so that writing programs to solve hard technical problems has become almost routine -- provided that the knowledge required can be put in reasonably restricted forms. Stanford's DENDRAL and M.I.T.'s MATHLAB programs show this sort of competence. But in all the subjects discussed below, in Learning, Vision, Planning, Natural Language, and Narrative Stories, things are not nearly so clear and the important difficulties all seem to cluster around a few common centers.

a. The problem of organization: hierarchy vs. heterarchy

We find that we can no longer separate programs cleanly into traditional parts and phases like: Pattern Recognition, Induction, Optimization, Tree-Search, Learning, Planning, etc. Instead, as we attack harder and more practical problems, we find that events on one level demand attention at others. For example, in the Hand-Eye system the gradual disappearance of a line being examined by a low-level vision program might be used as evidence that the region is smooth and shiny (and the apparent line is probably a reflection); this evidence, in turn, might be used to identify the object in view and change the whole strategy of the manipulation exercise being pursued. We discuss this in §3.

b. The problem of "error": accidental and deliberate

A second problem, when different kinds and sources of information interact, is that of inconsistency and error. This is not simply a question of reliability -- of deciding which is closer to the truth -- although that certainly is important. There are deeper reasons why assertions of different kinds cannot always be taken at "face value" at the same level. In human discourse, obviously, one has to take into account the motives of each speaker. But a similar situation obtains even in such a nonemotional environment as Mathematical Theorem-Proving, because simplifying "hypotheses" have to be treated specially -- and we do not expect "Theorem-Proving" ever to make much progress until

such facilities are made to operate smoothly. We discuss this in §5.

c. The problem of collecting and organizing general knowledge

As long as one deals with "toy" problems -- puzzles, games, and other situations (some of which may be quite practical) in which little or no interaction with other aspects of reality are required, Artificial Intelligence techniques are generally quite advanced by human performance standards. But when any understanding of things like space, and time, and people's desires, and economics, and design, etc., are required today's programs hardly approach the competence of a small child. We discuss this in §6.

2. THE PLANNER LANGUAGE

Until recently, most new heuristic programs were written from "scratch". Although based on principles from earlier work, one usually had to start over in the actual programming. This was not only because of incompatible information structures, but also because there was no standardization of even the most elementary processes of problem-solving itself -- handling of subgoals, descriptions, deductions, allocations and so on. The early work at Carnegie on "General Problem Solvers" made substantial progress on subgoal-management, but did not get very far on others of these problems.

Recently, we have seen changes in this picture. For example, Patrick Winston's new "description-learning" program (Thesis 1970) solved some of the kind of problems in Evans (Thesis 1964) without much modification. Terry Winograd's (Thesis forthcoming) new system developed for natural-language handling was made to work very well on mechanical manipulation problems (see §4). In the latter case, this was done partly by using the deductive capacity and other features of a new language, and the results were so interesting that we have decided to attempt to use it for rebuilding and unifying programs of several other of our projects.

The new language, PLANNER, offers new features that may turn out to be as basic and useful to heuristic programming as were the "DO"s and declarations of FORTRAN and ALGOL to numerical programming. In

older languages certain elements common to many programs, such as Iteration, Recursion, Boolean Conditions, Arrays, etc., were made publicly available so that one did not have to rewrite them for each new job. The IPL language, and LISP, and SNOBOL, brought further conveniences -- better data-structures, more automatic storage management, string manipulation and matching, attribute lists, etc. -- that were common ingredients of many problem-solving (as contrasted to ordinary "computational") projects. PLANNER makes available new mechanisms that promise to simplify the now-urgent problems of interlocking all our different kinds of programs. We now describe some of these mechanisms.

2.1 Demons

One of the mechanisms that PLANNER can make available is this: we can instruct it to behave so that

whenever __X__ happens, do __Y__ .

Now "X" can be a description -- in general terms with variables to be assigned -- of a kind of event, say the discovery of a new fact, or the proving of a theorem, etc. -- and "Y" can change a goal, or add new assertions to a data base, or erase information, or indeed do anything that can be expressed in the system. So in effect this statement sets up a sort of "Demon" (to use Selfridge's colorful name) that lurks, from then on, in the background until it sees what it is designed to look for. Quite a few problems of program development and interconnection can be viewed in terms of identifying the sorts of situations and

appropriate actions to build into such demons. Here are some examples of possible uses in our own application areas:

Vision: If a geometric region has a concave intrusion, look for:

1. missing lines
2. an occluding body
3. a shadowing body
4. an object in memory that has such a shape

etc.

Robotics: When an object is first seen, assign a real-time tracking process to its vertically highest feature. If it ever moves or disappears, enter an assertion about this into the data base.

Chess: Intercept any proposed King move and call a procedure to decide whether the Castling privilege is still important.

If King is ever actually moved, erase both the above Demon and this one itself!

Narrative: When a statement is made that appears to contradict normal knowledge, consider "flattery" or "bargaining" or "ignorance" (see §6). If these are plausible, inhibit deduction of consequences of the statement taken literally and deduce, instead, assertions about the speaker's motives for making the statement.

Now such things can be done in any ordinary programming language -- in principle -- but not in practice because one would either have the terrible job of inserting the traps at all appropriate places on all procedures, or incur the prohibitive expense of continual condition-checking. (In any case, the problems are much more serious with programs that add programs to themselves and these are what we will be dealing with in the learning processes of the future.) PLANNER could do the checking in its centralized assertion-handling facility, at a large but

not impossible cost. The obvious alternative -- inserting traps at all places that might create the condition -- is not practical in general, because the programmer may not even know the conditions required to cause the situation described by "X" (a child can be told to avoid fires without having to know all possible ways fires might be started). It remains to be seen how inexpensive such features could become, if PLANNER assertions are filtered through cleverly compiled decision-trees.

2.2 Convenience of Many Services

PLANNER offers smooth direct use of the entire LISP system, plus the conveniences of a string-manipulation language, and its control structure allows one to combine a variety of forms of subproblem interactions, error escapes, and accesses to common data bases.

2.3 Automatic Search and Backup Services

In PLANNER, when a subprocess comes to a conclusion on the basis of some experience or by deduction from other data it can add this conclusion as an "assertion" to the common data base. But it can be made also to store this assertion with "reservations", so that if, at a later date, one of the data involved is withdrawn or comes under suspicion then the assertion itself will be withdrawn or specially labelled.

Problem-solving often requires building up elaborate but possibly

temporary structures. PLANNER can be made to deal with statements that were once true in a model which may no longer be true after actions have been performed, and for drawing conclusions that may have to be deduced from such a change.

2.4 Deductive and Heuristic Assertions

To a large extent one can specify in PLANNER what one wants done rather than how to do it. Consider, for example, a statement of the form

"A implies B",

where "A" and "B" need not be specific terms but can be described by format-matching conditions; the system automatically searches for assertions that meet the specifications. As it stands this is a simple declarative statement. But in PLANNER we can ask that it instead be interpreted as the imperative:

> "if A is ever asserted, then assert B also and add it to the data base".

Or, it can be interpreted as advice for achieving goals:

> "if B is desired as a goal, assert A as a new subgoal to be deduced".

This is only the skeleton of the idea: PLANNER contains machinery for easy manipulation of several different kinds of declaratives, imperatives, goals and deductions. In attempting a particular deduction, for example, a goal-statement can specify suggestions for other information that might be useful (and even in what order) to make the deduction. All

these statements are expressed in a pattern-matching language in which control of procedures is specified in terms of the forms of assections rather than in terms of particular assertions.

2.5 Mathematical "Neatness" and Practical Compromise

The language is described by its designer, C. Hewitt, in a conference paper (IJCAI Proceedings, May 1969) and in more detail in A.I. Memo 168. In his dissertation he will present also a theory of the comparative strengths of a variety of Computational Schemata, including Iterative, Recursive, Parallel, and PLANNER-like systems, and show that these have increasing powers, in an appropriate sense. We feel that there is a potential advantage in using a language that has clean mathematical principles (as did "pure LISP") over languages that might be equally convenient for most practical purposes but lack theoretical clarity.

Because a smaller interpreter was needed for practical purposes, a simplified form of the language, called MICRO-PLANNER, was recently developed by T. Winograd, G. Sussman, E. Charniak and R. Greenblatt (A.I. Memo 203) which is able to handle efficiently the most essential aspects of Hewitt's language and is already adequate for the current projects. Eventually, an efficient realization of the full system may be developed; it does not seem to be a straightforward compiler-design problem because the control structure of PLANNER appears to be

fundamentally different from previous languages, as explained in a paper by Hewitt and Paterson (Comparative Schematology, June 1970). The "Demon" features, wishfully described above, can be realized with usable efficiency in the present, practical MICRO-PLANNER system only under the restriction that the recognition conditions can be expressed in terms of pattern-matching of atomic symbols at top list-structure level; it remains to be seen how expensive it would be to realize more elaborate conditions.

3*. THE VISION PROJECT: TOWARD A HETERARCHICAL PROGRAM ORGANIZATION

On the surface, work on Machine Vision has its most direct applications to activities like advanced automation and remote exploration. But in our laboratory we have found it an ideal testing ground for studying the heterarchical processes we believe necessary for attaining the kind of "common sense" adaptability that machines of the future will need.

The kind of visual performance we seek is exemplified by such goals as automatic analysis of a desk top littered with books, pens, telephones, etc., or the operation of an assembly-line under visual control so that the components do not have to be presented in precisely determined positions, or a vision system to guide a mobile automaton through unfamiliar terrain. The state of current operational systems is not quite ready for any of these: they work well only in scenes that are artificially and geometrically clean -- that is, composed of very simple objects that are illuminated so as to have few shadows, reflections, and the like.

To extend our capability in this area requires clarifying the successes and limitations of existing system components and then using this understanding to go further. But, these two "steps" cannot really be separated in practice, so our work is oriented not only to test

---

*Some of this section is taken from a proposal to NASA for a small contract, already awarded to us, to support the work of Adolfo Guzman in this area of machine vision.

particular theoretical ideas but also -- and this is at least as important -- to restructure the experimental system to maximize intelligibility and flexibility.

To develop this last remark we must give some more details about the system we have in mind. Previous programs to recognize geometric objects were usually organized in a hierarchical form so as to transform the data successively through such stages as these:

1. Raw data = a continuous numerical light function z = f(x,y).
2. Contour enhancement: replace f by a space derivative.
3. Identify edges and other features, and give them symbolic descriptions.
4. Group the features into objects.
5. Identify objects from their symbolic descriptions.

This is fine when it works. But each stage in the process has a certain fallibility. One way to improve the system is to optimize the components -- and because this is the easiest to do and to understand it is the direction generally followed. The results have never been satisfactory under real-world conditions. Fortunately, this is not the only way. Another route is to change the organizational structure of the system from a hierarchical form to what we propose to call a heterarchical* one. The key idea, as applied to this case, is to use a line-finder that is deliberately suboptimal in terms of its total probability of correct identification of lines but which is well matched

*"Hierarchy vs. Heterarchy" is a distinction coined and mainly used by Warren McCulloch in a slightly different context. We have decided to use it partly in honor of McCulloch but mainly because it is obviously better than our previous usage of "Horizontal vs. Vertical".

to the rest of the system in that its errors take on known forms. For example, in one of our presently working edge-finding systems (A.K. Griffith, Ph.D. Dissertation, 1970) the first steps often miss real lines but seldom propose false ones. The output of this line-finder then goes to a symbolic analysis whose purpose is not to completely interpret the scene but simply to decide whether the proposed set of lines seems "complete", e.g. do all the lines seem to be edges of "reasonably-shaped" regions? If it decides not, it generates a set of places where lines may have been missed. The system then goes back to an earlier stage of data-processing to apply more efficient tests for lines at these places. Although those tests are time-consuming in themselves they are not performed routinely, hence the overall computational efficiency of the heterarchical system is very much greater: for the same cost one gets much more reliability, or for the same reliability one has much lower costs.

Griffith's system demonstrates the power of even this small element of heterarchy. But it is clear that much more powerful results can be obtained by carrying the symbolic analysis further -- for example, by attempting, at certain stages, to decompose intermediate results into "bodies" as is done in A. Guzman's program SEE (Ph.D. Dissertation, 1968). This is also being studied by our colleagues at Stanford.

We now have an essentially hierarchical vision system. There is an interface between the output of an edge-finding program and the input to programs on the level of the SEE program, which in turn can communicate (but only in one direction) with the LEARNING program of Winston (Ph.D.

Dissertation, 1970). But in this system, small errors in the SEE program, due to deficiencies in the output of the lower-level program, still cause too much trouble at the highest level. Therefore, we are now starting to remove completely all boundaries between those programs -- or rather, between the heuristic and cognitive ideas embodied in those programs -- and embed them all in the form of "theorems" in a PLANNER-based language.

In the new system, the various capabilities of the programs will be dissected and packaged as small modules. These, in turn, are to be described by statements in the PLANNER language and can be activated by other statements. Their results, in general, will be the production of still further statements ("the dark area at ____ seems to be a shadow") as well as more conventional additions to a common data base. Our previous experience and the use of already working input-function programs and other modules should permit us to concentrate our effort on the organization and communication between the component modules of the system. Each module is supposed to have some ability to

1. criticize its input, asking for confirmation of it or even discarding it;

2. suggest actions "out of its domain", for instance a line-follower might suggest the presence of a book or the new SEE module might propose to look for an apparently missing line or the object-matching procedure might suggest that the body given to it to classify is really two bodies, not one;

3. construct an "excuse message", in any case of failure.

Thus, each module has a well-defined behavior (the line-finder locates the line it was asked to search for), but can also contribute

suggestions and requests for new actions.

We do not believe that we understand the situation well enough, now, to design a system in which modules are permitted to communicate freely and directly with each other, because there remain too many unsolved problems in such areas as resource allocation, "responsibility" for data, erasure, etc. Therefore, we plan first to put them all under control of an EXECUTIVE MODULE that:

1. gives orders to the modules, thus directing what should be done;

2. receives results from the modules and decides what to do next. The decisions are made by the deductive facilities of PLANNER, using the theorem Data Base, which is supposed, automatically, to

3. receive suggestions from the modules and implement, modify or discard them, according to "theorems" already in the data base, and

4. update the map of current knowledge about the scene, in accord with results and suggestions obtained.

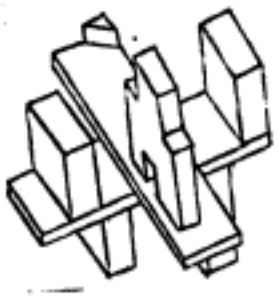
By keeping the executive procedures reasonably uniform, we hope the result will be an organization that is both simple and effective enough to attract the attention of our best theorists. The A.I. Group has a unique combination of first-rate experimenters and mathematical theorists, but they have rarely worked closely together in the past. We feel that the time is now ripe for this, partly as a result of our theoretical work on Perceptrons.

In developing the new system, several tasks are involved:

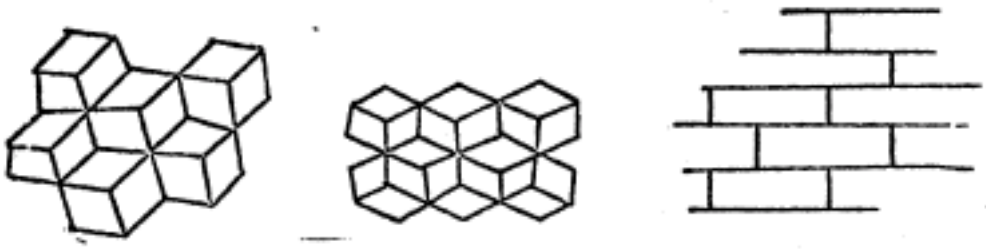
1. Understanding visual phenomena (lines, shadows, reflections, textures, etc.) and understanding the behavior of existing programs to

handle these phenomena. Classifying the failures of these programs, and modifying them to produce intermediate results of "excuses" when performance deteriorates, is necessary to produce "modules" that can be used effectively by the executive program. For example, we understood certain visual edge phenomena only after A. Herskovits (M.S. Dissertation, 1969) and A. Griffith (Ph.D. Dissertation, 1970) and L. Krakauer (Ph.D. Dissertation, 1970) carefully studied interactions between edges, highlights, cracks, and surface direction changes. We also now have a good idea, from the work of B.K.P. Horn (Ph.D. Dissertation, 1970) and of Krakauer, of how much can be found about the shape of a curved body from its shading. We know less about reflections and we still scarcely understand textures at all. On a higher level, we know relatively little about the potential power of Winston's new description-learning system. We would like a sharper theory of the apparently ad hoc parts of the SEE program; we think many of them can be derived from more general principles about convexity of bodies.

2. Developing the network of "suggestions", "excuses," "proposals", etc. Guzman's SEE program does a remarkable job at analysing a picture of arbitrary polyhedra into the objects present in it. It has no difficulty with this scene:



But it often fails on scenes like:



Now, one natural move would be to try to modify SEE to be better at dealing with scenes like these. Our alternative, in this case, is to become better at understanding SEE's mistakes (for instance the fact that they occurred in "repetitive" scenes), and to make SEE aware of the mistake, i.e. instead of reporting "failure" it will now report "I am failing because the scene is iterative or repetitive".* This "excuse" can now be analyzed by the executive module which could take an appropriate action, for instance calling a model-matcher program. P. Winston (Ph.D. Dissertation, 1970) has already developed a working system that can recognize and analyze many kinds of repetitive patterns, or other recursively structured interrelations, in such scenes. To put it another way, given that we have already a pretty good understanding of the visual phenomenon concerning a physical "body", this example suggests that to improve our ability to "parse" scenes it may be more profitable in this case to invest in understanding of "repetitive scenes" and their relation to "body", than to invest the same effort in deepening our knowledge about other topics than vision, e.g. mathematics or language or designing. There is always more than one way to do

*Of course, a more correct diagnosis would be that the bodies are so neatly, or closely, packed that there is excessive sharing of edges between bodies. In any case, another body of knowledge probably needed

something, and rather than struggle to force one scheme to solve many problems, it may be better to find out which idea is best in each situation.

3. Creating the executive program to organize the modules of a heterarchical system. Among the components of the system will be input modules that extract information and features directly from the scene; for instance, line-detectors, texture-comparators, focusing programs, color-detectors, etc. These input functions will not only provide some of the local information requested but will also be able to suggest the existence of objects, the grouping of certain features, the incompatibility of data, etc. But, they will not function as a traditional "preprocessor"; they will be called only when the executive has some reason to want their kind of information. Next, the system must contain the necessary ingredients for such procedures as the SEE program, to establish links between features, to form groupings of related items and portions of the scene, to suggest additional faces or features to be sought for, or to point out information that is "not credible", asking perhaps for its confirmation. Further, the system needs the ingredients for comparing the groupings proposed by the new SEE-like programs against models of the objects we are interested in, accounting for occlusions, shadows, and so forth. They should be able to suggest additional groupings of faces or features, searches for new data ("I suggest looking for an ellipse to the left of the handle"), and, again, requesting confirmation or rejection of conflicting information. But the main problem

after all this is to organize the primitive concepts of the executive program so that it will be easy to describe improved versions of the existing programs, and not too hard to make them interact as we wish.

Development of the new Heterarchical Vision System is under the direction of A. Griffith, B.K.P. Horn, G. Sussman, W. Williams, and P. Winston.

The reader who knows our previous work can see our current attack on building a complete Hand-Eye system is concentrated more on Vision than on Manipulation. This is because:

1. The present manipulator hardware and software, while primitive, is adequate for the actions our present systems demand.

2. We are starting to use visual feedback during manipulations, and this makes the present equipment more versatile than it has been in the past.

3. We are still not confident that we have the roots of a "decisive" theory for coordinated actions. We believe the most promising attack is in some new ideas about heuristics of learning coordinated skills, and we are planning to pursue these ideas (under another contract, being proposed to NSF). In any case, we feel that at the present time the most direct course toward improving our manipulation capacity*, at least for the next year or so, is through making the higher-level improvements in our methods discussed throughout this proposal.

---

*Although we do not plan to devote a large part of our laboratory work to developing better mechanical hands (we <u>are</u> continuing some work on better eyes) we feel that research-supporting agencies should be receptive to proposals for such work, because otherwise the manipulation programs will indeed become limited soon by the inadequate hands available today. Within two or three years, the ARPA projects will be needing better manipulators. They should have force-controlled, position-feedback actuators, very dextrous hands, and good tactile sensors. There are signs that AEC and NASA are coming to appreciate this, but the indications are that this field will continue to move slowly. Our group would like to be involved in such projects, in an advisory capacity, and could offer testing facilities.

4. NATURAL LANGUAGE AND SEMANTIC INTERPRETATION

In A.I. Memo 168 C. Hewitt described, among other things, some methods of solving deductive problems about the manipulation of objects by a robot, such as "pick up all pairs of cubes that are the same color with one cube on top of a third cube that is in front of the other". Terry A. Winograd has completed a system that translates such natural English statements into PLANNER assertions and theorems.

Winograd's system is designed to accept information in normal English sentences, to answer questions, and to execute commands, using semantic information to disambiguate grammatically complicated texts. To do this, Winograd uses a new linguistic scheme, based partly on the systemic grammar described in Halliday (1967) and in Winograd (1968), and partly on a special representation for both the grammar and the semantics. In previous attacks on such problems some workers have used heuristic tricks -- key words or matching of phrase fragments -- or they have used nonheuristic formal grammars to attempt a detailed analysis of the sentence to which the semantics are to be applied. These have never worked very well, though they were important first steps. Winograd's system is based on a heuristic grammar that uses contextual information in analyzing the sentence, carrying out the semantic analysis concurrently; this is made possible by representing the grammar as a program instead of as a set of static rules. Definitions of words, as well as the system's knowledge about things, may also be stored as programs and are available to the deductive part of the system. This

allows much more flexibility than one gets naturally either from "semantic networks" or from grammatical rule-lists. The program can make long, complex deductions in answering questions or in absorbing new information, using the deductive features of PLANNER. (See Winograd, "PROGRAMMAR: A Language for Writing Grammars", A.I. Memo 181.)

The system already contains a powerful collection of syntactic and semantic knowledge for connecting ordinary English with such information and heuristic rules as are needed to manipulate simple structures of simple objects. For example, it can handle the actual metric computations associated with the fact that one cannot set down an object until one has cleared away a space large enough to support it. The following sample dialogue is not merely verbal, but is associated with an actual simulation (displayed on a scope in reasonably "real" time) of an arm picking up and moving the objects mentioned in the discussion! The first line in each group is supplied by the human; the reply is generated by the program after the internal problem-solving process is completed. This demonstration example is from a draft of Winograd's forthcoming dissertation.

Warning: Although the performance is far more advanced, both in syntax and semantics, than its predecessors (for example, Raphael's SIR and QAS or Weizenbaum's ELIZA), this demonstration cleverly exhibits its virtues in a particular situation and it would not, today, work as well if the human operator decided to "change the subject"! The system's vocabulary, both in words and concepts, is still very small, and the sample below uses most of it.

Sample Dialog with English-Understanding Program -- July 4, 1970

Terry Winograd, M.I.T.

PICK UP A BIG RED BLOCK. (typed by operator)
OK. (machine's reply; "OK" here means that the action is also actually performed)

GRASP THE PYRAMID.
I DON'T UNDERSTAND WHAT YOU MEAN BY "THE PYRAMID". (there is more than one pyramid in the scene)

FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU ARE HOLDING AND PUT IT INTO THE BOX.
BY "IT" I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING.
OK. (it does)

WHAT DOES THE BOX CONTAIN?
THE BLUE PYRAMID AND THE BLUE BLOCK.

WHAT IS THE PYRAMID SUPPORTED BY?
THE BOX.

HOW MANY BLOCKS ARE NOT IN THE BOX?
FOUR OF THEM.

IS AT LEAST ONE OF THEM NARROWER THAN THE ONE WHICH I TOLD YOU TO PICK UP?
YES, THE RED CUBE.

IS IT SUPPORTED?
YES, BY THE TABLE.

CAN THE TABLE PICK UP BLOCKS?
NO. (by deduction using facts in the theorem-prover's data base)

CAN A PYRAMID BE SUPPORTED BY A BLOCK?
YES. (by deduction using facts in the theorem-prover's data base)

WILL YOU PLEASE STACK UP BOTH OF THE RED BLOCKS AND EITHER A GREEN CUBE OR A PYRAMID?
OK.

```
STACK UP TWO PYRAMIDS.
I CAN'T.

WHICH CUBE IS SITTING ON THE TABLE?
THE LARGE GREEN ONE WHICH SUPPORTS THE
  RED PYRAMID.

IS THERE A LARGE BLOCK BEHIND A PYRAMID?
YES, TWO OF THEM, A LARGE RED ONE AND
  THE BLUE ONE.

PUT A SMALL ONE ONTO THE GREEN CUBE
  WHICH SUPPORTS A PYRAMID.
OK.

DOES THE SHORTEST THING THE TALLEST
  PYRAMID'S SUPPORT SUPPORTS SUPPORT
  ANYTHING GREEN?
YES, THE GREEN PYRAMID.                 (the parser is pretty good, and
                                         so is the logic!)

WHAT COLOR IS IT?
BY "IT" I ASSUME YOU MEAN THE SHORT-
  EST THING THE TALLEST PYRAMID'S
  SUPPORT SUPPORTS.
RED.

HOW MANY THINGS ARE ON TOP OF GREEN
  CUBES?
I'M NOT SURE WHAT YOU MEAN BY "ON TOP
  OF" IN THE PHRASE "ON TOP OF GREEN
  CUBES".  DO YOU MEAN
  1.  DIRECTLY ON THE SURFACE
  2.  ANYWHERE ON TOP OF?

2                                       (operator's response)
THREE OF THEM.                          (machine reply)

THE BLUE PYRAMID IS NICE.
I UNDERSTAND.

I LIKE BLOCKS WHICH ARE NOT RED BUT
  I DON'T LIKE ANYTHING WHICH SUP-
  PORTS A PYRAMID.
I UNDERSTAND.                           (it knows very little about "liking"
                                         except that one usually "likes"
                                         things that are "nice")
```

DO I LIKE THE BOX?
NO.

DO I LIKE ANYTHING IN THE BOX?
YES, THE BLUE BLOCK AND THE BLUE
PYRAMID.

(boxes support things that are in them)

HAD YOU TOUCHED ANY PYRAMID BEFORE
YOU PUT THE GREEN ONE ON THE
LITTLE CUBE?
YES, THE GREEN ONE.

(the logic can deal with the history as well as with the present state, and this is nicely tied in with the syntax-analysis of tenses)

WHEN DID YOU PICK IT UP?
WHILE I WAS STACKING UP THE RED CUBE,
A LARGE RED BLOCK AND A LARGE GREEN
CUBE.

WHY?
TO GET RID OF IT.

WHY DID YOU DO THAT?
TO CLEAR OFF THE RED CUBE.

WHY DID YOU CLEAR OFF THAT CUBE?
TO PUT IT ON A LARGE GREEN CUBE.

WHY DID YOU DO THAT?
TO STACK UP THE RED CUBE, A LARGE RED
BLOCK AND A LARGE GREEN CUBE.

WHY DID YOU DO THAT?
BECAUSE YOU ASKED ME TO.

HOW DID YOU DO IT?
BY PUTTING A LARGE RED BLOCK ON THE
TABLE: THEN LETTING GO OF IT, THEN
PUTTING A LARGE GREEN CUBE ON IT:
THEN LETTING GO OF THAT CUBE: THEN
PUTTING THE RED CUBE ON THAT CUBE:
THEN LETTING GO OF THAT CUBE.

HOW MANY OBJECTS DID YOU TOUCH WHILE
YOU WERE DOING IT?
FOUR OF THEM.

```
WHAT DID THE RED CUBE SUPPORT BEFORE
  YOU STARTED TO CLEAN IT OFF?
THE GREEN PYRAMID.

THERE WERE FIVE BLOCKS TO THE LEFT OF
  THE BOX THEN.
NO, ONLY FOUR OF THEM:  THE RED CUBE,
  TWO LARGE GREEN CUBES AND A LARGE
  RED BLOCK.                              (rejecting an inconsistency)

PUT THE BLUE PYRAMID ON THE BLOCK IN
  THE BOX.
OK.                                       (it does)

IS THERE ANYTHING WHICH IS BIGGER
  THAN EVERY PYRAMID BUT IS NOT AS WIDE
  AS THE THING THAT SUPPORTS IT?
YES, THE BLUE BLOCK.

DOES A STEEPLE                            (it interrupts immediately)
SORRY, I DON'T KNOW THE WORD "STEEPLE"
PLEASE TYPE TWO SPACES.                   (so it can clear out the TYI buffer
                                           in case you typed while it was
                                           trying to interrupt you)

A "STEEPLE" IS A STACK WHICH CONTAINS
  TWO CUBES AND A PYRAMID.
I UNDERSTAND.

ARE THERE ANY STEEPLES NOW?
NO.

BUILD ONE.
OK.                                       (it does!)

CALL THE BIGGEST BLOCK "SUPERBLOCK".
OK.

HAVE YOU PICKED SUPERBLOCK UP SINCE
  WE BEGAN?
YES.

WHY DID YOU DROP IT?
BY "IT" I ASSUME YOU MEAN THE SUPER-
  BLOCK.
I CAN'T EXPLAIN A NONEXISTENT EVENT.      (an excuse for being unable to
                                           answer the question, taken at
                                           its face value)
```

```
IS THERE ANYTHING TO THE RIGHT OF THE
  RED PYRAMID?
YES, FOUR OF THEM, THE BOX, THE RED
  CUBE, THE BLUE PYRAMID AND THE BLOCK
  NAMED SUPERBLOCK.

PUT AWAY ALL OF YOUR TOYS.
OK.

THANK YOU.
YOU'RE WELCOME!
```

We repeat our warning; although the system appears, in this exercise, to have a good grip on aspects of logic, geometry, causality, etc., it could not (for example) discuss the relations between faces and edges of the objects. Only time will tell which kinds of extensions will involve the greatest difficulties.

5. WHY WE ARE STUDYING KNOWLEDGE AND LEARNING (RATHER THAN, SAY, THEOREM-PROVING OR TREE-SEARCHING)

Artificial Intelligence as a field of inquiry is faced with a serious crisis of identity. The form of the crisis is by no means unique: every discipline has had to deal with similar situations and this was often true of our own predecessor, the traditional study of Human Intelligence. The problem stems from the tendency for the pursuit of technical methods to become detached from their original goals so that they follow a developmental pattern of their own. This, of course, is not necessarily a bad thing viewed from a global perspective that assigns equal value to all areas of endeavor. Indeed, many productive areas of research were born of such splits. Nevertheless, if one is primarily interested in our particular goal, of building a science of Artificial Intelligence, one has to be concerned about the deployment of resources both on the local scale of conserving one's own time and energy and on the global scale of watching to see whether there is a "critical mass" of effort in the scientific community as a whole. We consider the large amount of work on Perceptrons, that started in the early 1960's, to have been a serious deflection, and we suspect that the same is happening now in connection with the study of formal procedures for Mechanical Theorem Proving.

As a first approximation to formulating the issues consider a hypothetical but typical research project claiming to be working on "theorem proving". Schematically, the project has the form of a large

computer program which can accept certain "bodies of knowledge" or "data bases", such as a set of axioms for group theory, or a set of statements about pencils being at desks, desks being in houses, and so on. Given the body of knowledge, the program is asked to prove or disprove suitably formulated assertions. What normally happens is that if the body of knowledge is sufficiently restricted in size, or in content or in formulation, the program does a presentable job. But, as the restrictions are relaxed it grinds to an exponential stop of one sort or another.

There are two kinds of theory about how to improve the program. Although no one actually holds either theory in its extreme form and although we encounter theoretical difficulties when we try to formalize them, it nevertheless is a good strategy for discussion to identify and name the extreme forms of the theories.

The POWER theory seeks more computational power. It may look toward new kinds of computers ("parallel" or "fuzzy" or whatever) or it may look toward extensions to deductive generality, or filing, or search algorithms -- things like better "resolution" methods, better strategies for exploring trees and nets, hash-coded triplets, etc., etc.

The KNOWLEDGE theory sees progress as coming from better ways to recognize, express and use the diverse forms that knowledge seems to take when used (for example) by human thought. This theory sees the problem as <u>epistemological</u> rather than as a matter of computational power. It supposes, for example, that when a scientist solves a new problem, he engages a highly organized structure of facts, models,

analogies, planning mechanisms, self-discipline procedures, etc., etc. To be sure, he also engages intricate and reliable "general" problem-solving schemata but it is by no means obvious that very smart people are that way primarily because of the superior power of their general methods -- as compared with average people. Their competence may be, in large part, due to the better organization of their knowledge.

But, the distinction between procedural power on the one hand and quality of knowledge on the other is surely a caricature of a more sophisticated kind of "trade-off" that we do not yet know how to talk about productively. A smart person is not that way, surely, either because he has luckily got a lot of his information well organized or because he has a deductive schema very good, say, at binding functional arguments or at keeping track of the ancestry of parts of a goal tree. His intelligence is surely more dynamic in that he has (somehow) acquired a body of knowledge and a body of procedures -- if the distinction is legitimate (which we doubt) -- that themselves are used to guide the organization of more knowledge and the formation of new procedures. The crucial requirement is that the system be organized in a way that permits the bootstrapping of its intelligence through the effective incorporation of new knowledge.

Heuristic Knowledge

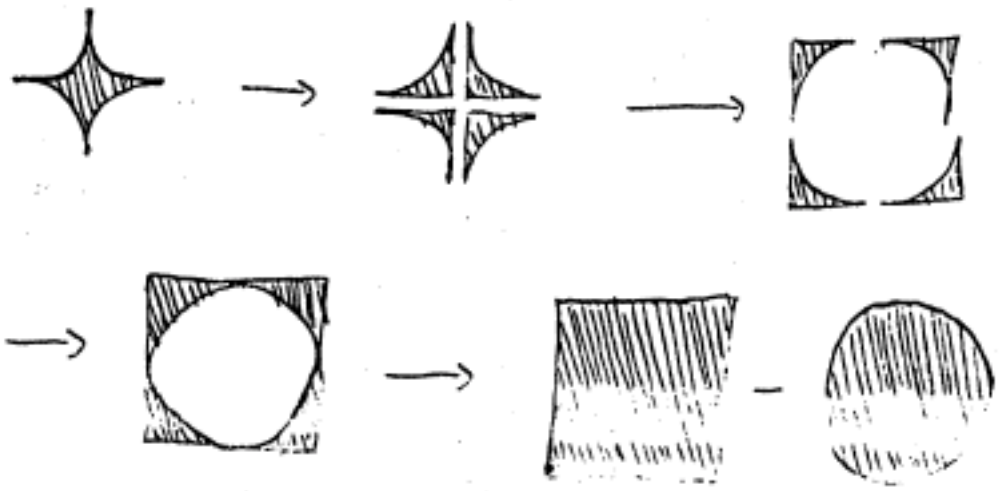
Now, returning to the specific question in mechanical theorem-proving, let us ask why those programs fail on hard problems unless

they are provided with carefully restricted diets of data. (They fail even more surely if given just the bare axioms of a mathematical system and asked to prove advanced theorems.) We might begin by examining the behavior of a human Mathematician. If he is asked to find the volume of some object he will probably use principles like:

To find the volume of an unusual shape, try:

1. cutting it into a sum of familiar shapes;
2. enclosing it "tightly" in a familiar shape and try to find the difference-volume;
3. transform, metrically, the space so that the shape becomes more familiar;
4. etc.

Thus, one would transform:



What is the form of such a principle?

If a situation is like "A" try to use Method "B".

Now, in the final "proof" this principle will not appear: But its use was crucial. We thus see that the three kinds of information in

1. the knowledge exhibited in the proof,
2. the knowledge used in finding a proof,
3. the knowledge required to "understand" the proof so that one can put it to other uses,

are not necessarily the same in extent or in content. The "Theorem-Prover systems" have not been oriented toward making it easy to employ the first and the third kinds of knowledge.

Successive Approximations and Plans

Another common and important kind of problem-solving is exemplified by an engineer or physicist analyzing a physical system. First, he will make up a fairy-tale: "The system has perfectly rigid parts, can be treated as purely geometric. There is no friction, and the forces obey Hooke's law." Then he solves his equations. He finds the system offers infinite resistance to disturbance at a certain frequency!

Ridiculous! But he does not reject this absurdity, completely. Instead, he says: "I know this phenomenon! It tells me that the "real" system predictably has an interesting resonance at or near this frequency." Next, he makes a new model, including damping and coupling terms, and he studies this system near the interesting frequency. (He knows that his new model is probably very bad at other, far-away, frequencies for there he will get false phenomena because of the unaltered assumptions about

rigidity; he believes these are harmless in the frequency band now being studied.) Then he solves the new second-order equations, perhaps by making some use of the first approximate (but divergent) solution. This time, he obtains a pair of finite, close-together, resonances of opposite phase. That "explains" the singularity in the simpler model, he feels. He has used the simpler, easier-to-solve equations to find where the interesting things might happen; then he abandoned one simple "micro-world" and adopted another, slightly more complicated and better adapted to the now-suspected phenomenon. This, too, may serve only a temporary role and then be replaced by another, more specialized set of assumptions for studying how nonlinearities will perturb the fine-structure of the resonances.

One cannot overemphasize the importance of this typical kind of scenario both in technical and in everyday thinking. One is absolutely dependent on having highly-developed models of many phenomena. Each model -- or "micro-world" as we shall call it -- is very schematic; it talks about a fairyland in which things are so simplified that almost every statement about them would be literally false if asserted about the real world. Nevertheless, we feel they are so important that we are assigning a large portion of our effort toward developing a collection of these micro-worlds and finding how to use the suggestive and predictive powers of the models without being overcome by their incompatibility with literal truth. We see this problem -- of using schematic heuristic knowledge -- as a central problem in Artificial

Intelligence today, and feel that the current batch of formal deductive procedures (such as trying to find formal syntactic methods to improve the efficiency of "Resolution-Based Theorem Provers") tend to obscure rather than clarify this issue. That is why we are not encouraging our students to pursue the latter approaches in their experimental work.

6. NARRATIVE, MICRO-WORLDS, AND "UNDERSTANDING"

In order to face squarely these problems of knowledge and micro-worlds, we have added to our top-level goal-list a new project area -- called "Understanding Narrative". The dialogue given in §4 is an example of a sort, but our goal extends to the hope of eventually understanding the kind of information found in books. We decided to begin by studying the problems one encounters in trying to understand the stories given to young children. Eugene Charniak, a graduate student, has been working in this area, and much of the analysis in this section is based on his work. We quickly found that we did not have adequate models for this, and have become convinced that obtaining them will be a vital step in developing machines with common-sense. Do not be misled by the frivolous implication. The point is that any six-year-old understands a great deal more about each of such absolutely crucial and various things as

time,

space,

case,

planning,

explaining,

giving,

breaking,

than any of our current programs.

Furthermore, before discussing some particular aspects of the Narrati Understanding problem, we want to point out that the image -- of

comprehending a story as read from a book -- does not quite do justice to the generality of our task. One has the same kinds of problems in:

- making sense of a sequence of events one has seen or otherwise experienced,
- watching something being built (why did they do that first?),
- understanding a mathematical proof (what was the real point, what was mere technical detail?)

and so forth.* But, and less obvious, many of the processes we do not normally think of as sequential -- such as seeing a scene (why is there a shadow here? what is that? oh, it must be the bracket for that shelf) -- are in fact processes, and concepts like "seeing at a glance" in our language and thought are not helpful in our search to find what are the real problems.

The "Face Value" of Assertions

People do not always mean what they say. One would certainly not go very far in understanding stories or in coping with people if every statement were taken as literal and infallible truth. To deal with stories in context one needs, to begin with, the concept of story (and this should be recursive, since there can be stories in stories). Then one must know about a number of special kinds of story involved with such activities as lying, pretending, bargaining, poetry, and so on. Some of these concepts might seem unnecessary for an elementary robot whose function will be to deal with a practical real world, in which no one will try to sell it the Brooklyn Bridge or mix in nonsense-verse with its

*See the discussion of "kinds of learning" in our 1969 Project Proposal.

daily information sheet.

But, in fact these frivolous modes of behavior are very close, in their requirements for intellectual structure, to some indispensible patterns of ordinary thinking. An intelligence agent might indeed operate in a limited world without knowledge of lying, but it is harder to believe that much intelligence is possible unless it has some form of the concept of mistake, or program bug.

We will return to this in a moment, but first consider the problem of making a program that can "understand" the kind of story a young child often reads. In a familiar fable, the wily Fox tricks the vain Crow into dropping the meat by asking it to sing. The usual test of understanding is the ability of the child to answer questions like,

"Did the Fox think the Crow had a lovely voice?"

The topic is sometimes classified as "natural language manipulation" or as "deductive logic", etc. These descriptions are badly chosen. For the real problem is not to understand English; it is to understand at all. To see this more clearly, observe that nothing is gained by presenting the story in simplified syntax: CROW ON TREE. CROW HAS MEAT. FOX SAYS "YOU HAVE A LOVELY VOICE. PLEASE SING." FOX GOBBLES MEAT. The difficulty in getting a machine to give the right answer does not at all depend on "disambiguating" the words (at least, not in the usual primitive sense of selecting one "meaning" out of a discrete set of "meanings"). And neither does the difficulty lie in the need for unusually powerful logical apparatus. The main problem is that no one has constructed the elements of a body of knowledge about such matters that is adequate for understanding the story. Let us see what is

involved.

To begin with, there is never a unique solution to such problems, so we do not ask what the Understander must know. But, he will surely gain by having the concept of FLATTERY. To provide this knowledge, we imagine a "micro-theory" of flattery -- an extendible collection of facts or procedures that describe conditions under which one might expect to find flattery, what forms it takes, what its consequences are, and so on. How complex this theory is depends on what is presupposed. Thus, it would be very difficult to describe flattery to our Understander if he (or it) does not already know that statements can be made for purposes other than to convey literally correct, factual information. It would be almost impossibly difficult if he does not even have some concept like PURPOSE or INTENTION.

In addition to a micro-theory of flattery, the Understander needs some facts. For example, he would surely miss the point if he thought that crows sing beautifully, that foxes in stories are symbolic of honesty rather than cunning, or that they hate eating meat but might do so as a friendly gesture to please the crow.

This is enough to indicate an area of research that might be called "epistemonomy" if that were not such a funny-sounding word. Is it part of Artificial Intellignece? Surely it is at least as important to have the models as it is to construct the program to use them as Understander is meant to do. Indeed, more so! For even the most trivial f the well-known deductive systems might be good enough if one had enough "epistemonomy"; while even the most powerful deductive system in the world

will be useless without a good body of knowledge. Yet there are now dozens, perhaps hundreds, of people working hard at deductive procedures, while the problem of knowledge is almost universally shunned.

How large would such a knowledge base need to be, and how could it assembled? It would be large indeed, but we are convinced that a powerful intelligence could be well oriented to the real world with the equivalent of less than a million statements. Assembling the collection of micro-theories would take many man-years (as does building a conventional encyclopedia, or even a dictionary). But, of course no such large project could be reasonably launched until we are in a position to choose from a variety of convincing prototypes of how such systems might be made to work. In effect, we are now scouting out the problems in making such prototypes.

Errors, Lies and Hypotheses

Erroneous statements call for much the same kind of treatment as lies: they are often recognized by inconsistency or implausibility, and demand similar precautions to prevent implications from spreading throughout the body of knowledge. And we have already discussed in §5 their close relative -- the deliberate kind we use in hypothetical thinking when we set up intentionally oversimplified models as a way to get started on a problem.

We recall that the primitive, "horizontal" kind of vision system uses "hypotheses" only in this kind of way: it might carry out a test

on the hypothesis that a line extends into a certain region. If this hypothesis produces inacceptable consequences it is simply abandoned and another one is generated. In contrast, the idea behind the heterarchical vision system is that one can always "learn from mistakes", that is to say come out of the situation created by the false hypothesis with additional knowledge, if only an excuse or explanation of why it failed. There is a deep difference of principle. The elementary system was easy to organize just because all consequences of the bad hypothesis are abandoned. In the heterarchical system some are retained and so a completely new problem of selection arises.

It might seem illogical to consider keeping consequences of wrong hypotheses. This is true, but its real consequence is that the dichotomy right/wrong is too rigid. In dealing with heuristics rather than logic the category true/false is less important than fruitful/sterile. Naturally the final goal must be to find a true conclusion. But, whether logicians and purists like it or not, the path to truth passes mainly through approximations, simplifications, and plausible hunches which are actually false when taken literally. And little good can come from a mind or a machine whose only ability with respect to these fruitfully false statements is to detect their falsity, abandon them and try some other route.

Structural Primitives

We mentioned lying, flattering, pretending, and hypothesizing as examples pointing to the need for micro-world construction. Though

requiring (presumably) similar structural machinery, each nevertheless requires some specific knowledge to differentiate it from the others. One could, for example, imagine a child who knows about story-telling and pretending, for play purposes, but who has not yet noticed that one might pretend in order to influence someone else's behavior in a bargaining situation. This child would presumably be confused by, or simply fail to understand what is going on, in:

> Janet: "That isn't a very good ball you have. Give it to me and I'll give you my lollipop."

(It is clear to us, in the original story, that Janet really is trying to get the ball and is not being sympathetic or altruistic.)

How can we build a system to understand Janet's behavior? Consider a sequence of suggestions:

1. Take all statements at face value.

   This won't work at all. It is so a nice ball.

2. Assign a truth probability to each.

   This sort of attempt to avoid building models for meanings has never led anywhere. Statistics unaided can do no good; they must attached, at least, to appropriate states.

3. We should not accept "that is not a nice ball". We should instead add "Janet thinks it is not a nice ball" to the data base.

   This is surely a step in the right direction, but is still dead wrong!

4. We recognize a bargaining situation.

   Janet is trying to reduce Bill's subjective value on the ball, so that he will be more likely to trade. Of course,

it is a nontrivial problem to know when such a model is plausible.

Now we have an appropriate concept and can begin to "understand". In fact, Janet's strategy doesn't work (in the actual story) and the sophisticated reader suspects that Bill discounts Janet's statement for what it is: a bargaining ploy. For the program to be able to have such a suspicion, not only must it have the bargaining model in its own analysis program but it must be able to embed a simplified version of this into its internal model of Bill, and in such a way that the program can realize that Bill can be fooled.

Now we could argue about the merits of "subjective value" or other schemes for modeling the bargaining concept. This is not the point; there are many other ways and, no doubt, that is part of the differences between people. Our point is that it is unreasonable to expect a program to interpret such behavior on the basis of ANY set of very general principles: the machine simply has to know, specifically, a lot about modes in which people -- especially children -- operate. On the other side we conjecture that, eventually, the required micro-theories can be made reasonably compact and easily stated (or, by the same token, learned) once we have found an adequate set of structural primitives for them. When one begins to catalogue what one needs for just a little of Janet's story, it seems at first to be endless:

Time	Things	Words
Space	People	Thoughts

Talking: Explaining. Asking. Ordering. Persuading. Pretending.

Social Relations: Giving. Buying. Bargaining. Begging. Asking. Presents. Stealing...

Playing: Real and Unreal, Pretending

Owning: Part of, Belong to, Master of, Captor of

Eating: How does one compare the values of foods with the values of toys?

Liking: Good, bad, useful, pretty, conformity.

Living: Girl. Awake. Eats. Plays.

Intention: Want. Plan. Plot. Goal. Cause. Result. Prevent.

Emotions: Moods. Dispositions. Conventional expressions.

States: asleep. angry. at home.

Properties: grown-up. red-haired. called "Janet".

Story: Narrator. Plot. Principal actors.

People: Children. Bystanders.

Places: Houses. Outside.

Angry: State

Caused by: insult

deprivation

assault

disobedience

frustration

spontaneous

Results not cooperative

lower threshold

aggression

loud voice

irrational

revenge

Etc.

But it is not endless. It is only large, and one needs a large set of concepts to organize it. After a while one will find it getting harder to add new concepts, and the new ones will begin to seem less indispensible.

Ultimately, there is no way to avoid the need for such knowledge; "common sense" and "general competence" simply do not exist as primitive, disembodied faculties either of mind or of "program organization" or of "integrated coordination of neurons acting in holographic concert". One must, in the final analysis, acquire knowledge about things, about facts, about procedures, about descriptions, and about heuristics for putting all such things together.

This does not mean the catalogue must be complete before we start! Once one knows a "certain amount" about the world, and a "certain amount" about how to understand things, one can begin to propose one's own models and debug them, one can learn by reading, arguing, experimenting, and so on. But we can discover this threshold of intellectual "bootstrapping" only by learning much more about using knowledge.

We believe that the identification is the harder part of the problem

and hope that the ideas in this proposal are advances in that direction. The ideas are not all new, of course, and there is much here that stems from our colleagues -- especially at Carnegie and Stanford -- and our students.

7. MATHEMATICAL THEORY OF COMPUTATION

7.1. Computational Geometry

A number of investigators in other places have taken up and greatly enriched themes from this area which originated as a serious field of study in our group. During the coming year we shall devote part of our effort to integrating some of these results into a general theory and examining applications to robot vision. Two examples follow.

J. Baker of Rockefeller University has developed a continuous theory of perceptrons. This theory shows how our basic perceptron theorems can be formulated and proved without reference to the discrete retina! An immediate practical consequence is that transformation groups involving rotations and contractions can be studied with the same ease and rigor as groups of translations. Consequences of this sort are extremely important to applications of perceptron theory, and we believe similar methods will emerge in other areas of computational geometry. Baker's work also has another kind of reverberation influencing prevailing ideas about the kind of mathematics most relevant to computation. An examination of the content of "Computer Science" courses will show a heavy emphasis on what is known as "finite" or "discrete" mathematics. However, we had already observed in developing the perceptron theory that the key mathematical ideas came from areas

of mathematics that are not usually so classified and indeed are not taught to students of computer science. Baker's theory leans even more heavily on "analytical-continuous" methods. Closer examination may show that these methods are logically inessential; simple ideas often dress up in complex mathematical forms. But, if Baker's methods are truly related to the computational content of the theory it will become urgent to explore some unsuspected areas of mathematics for computational relevance.

Another, very different area of computational geometry that has shown signs of active growth is the theory of interpreting line drawings in the plane of projections of three-dimensional objects. In our laboratory this is, of course, an old theme whose most elegant application was Guzman's program SEE. New ideas have come from several sources: D. Huffman at Santa Cruz has discovered some powerful general principles that apply to such curious (but important) cases as creased cloth as well as to the standard solid, rectilineal objects; the Stanford group has observed that a dual theory can be more noise-resistant in some situations than Guzman's representation; in our own laboratory C. Hewitt has shown that superficially quite different mathematical analyses can be used to interpret line drawings. Taken together these ideas point to a rich prospect of a unified and deeper mathematical theory of this topic.

7.2. Mathematical Foundations for Theorem-Proving and Problem-Solving Programs

We turn next to the mathematical underpinnings for our new direction of research in the construction of intelligent machines. We must state candidly that the area is as underdeveloped theoretically as "pattern recognition" was five years ago and that we are hardly in a position even to make plausible conjectures about many key problems. The most pressing theoretical need is for concepts to permit sharper formulation of problems, conjectures, and even different approaches to the problem area. The development of PLANNER and the study of problem-solving primitives is a step in this direction and one that can be given a clear mathematical form in the context of program schemata. In our laboratory C. Hewitt and M. Paterson have developed concepts of this kind to enable one to give formal versions of descriptions such as "LISP-like" languages and "PLANNER-like" languages and to prove some theorems about their relative power. During the coming year, work in this direction will continue on a theoretical level and stands a very good chance of finding applications in clarifying issues in designing problem-solving systems.

Computational Complexity studies in the area of deductive programs is another area in which work is needed. In recent times deductive procedures -- we are thinking particularly, but not only, of those based on the predicate calculus -- have

proliferated, but there is no nontrivial solid knowledge about their relative power or absolute limitations. We are sure that limiting theorems can quite easily be found in the form:

> any procedure P that resembles standard resolution methods in such-and-such ways will take more than such-and-such a quantity of computation to prove a typical member of such-and-such a class of theorems.

Investigations of this sort are quite urgent. The point is not to show that resolution methods cannot solve all the problems of intelligence. It is, rather, the hope that theorems about what cannot be done by programs without recourse to diversified knowledge will lead toward a mathematical theory of how knowledge is used.

A theme closely related to the use of common sense knowledge is the study of optimizing, or even just improving, computation by combining several mathematically well-defined computational methods. The following two examples illustrate a corner of this subject where we know a little and a corner where we have scarcely gone beyond formulating some problems and convincing ourselves of its importance.

The first example is the structure of a typical chess-playing program. The computation of the move to play is now always made in at least two stages involving different kinds of computation: a stage of plausible move generation based on essentially linear threshold computation and a stage of tree search.

Now, we ask the general question: under what conditions does it pay to so subdivide a computation? In particular, does the success of this "factoring" of the computation depend on very special features of the game of chess? To this we can give fragments of an answer. For example, we know that it is possible to define games formally like chess, for which a linear plausible move generator will contribute nothing!

The second example concerns the combination of symbolic and numerical methods. It is obvious, for example, that certain integrals can be evaluated symbolically much faster than they can be obtained by any known numerical techniques, but it is not at all obvious under what conditions it would pay to fit a symbolic description to numerical data in order to integrate it. The general problem of passing back and forth between symbolic and numerical -- or even conceivably between special analog devices and digital methods -- is deep and clearly not ripe for general solution. But, it does seem right to start collecting and examining special cases and phenomena in this area.