MASSACHUSETTS INSTITUTE OF TECHNOLOGY
A. I. LABORATORY

Artificial Intelligence
Memo No. 245

October 1971

PROPOSAL TO ARPA FOR RESEARCH ON ARTIFICIAL INTELLIGENCE AT M.I.T., 1971-1972

Marvin Minsky and Seymour Papert
(and staff)

Work reported herein was supported by 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 Contract Number N00014-70-A-0362-0002.

PROPOSAL TO ARPA FOR RESEARCH ON ARTIFICIAL INTELLIGENCE AT M.I.T., 1971-1972

Marvin Minsky

Seymour Papert

(and staff)

INTRODUCTION

0.1 Scientific Structure of the Artificial Intelligence Laboratory

The activities of the Artificial Intelligence Laboratory can be viewed under three main aspects:

1. Artificial Intelligence: Understanding the principles of making intelligent machines along the lines discussed in previous proposals, and elaborated below.

2. Natural Intelligence: As we understand intelligence better we see fewer differences between the problems of understanding human and machine intelligence.

We have been increasingly able to translate our ideas about programming machines into ideas about educating children, and are currently developing systematic methods in elementary education. And conversely, we attribute to our observations and experience in the latter activities much of what we believe are important new conceptions of how to organize knowledge for programs that really understand.

3. Mathematical Theories: This aspect is relevant not only because we often need to solve specific mathematical problems but especially because we are firmly committed to maintaining a mathematical style in the laboratory. In many centers we have seen decline and deterioration following an apparently successful "experiment" in artificial intelligence because the principles behind the performance were not understood, hence the limitations unseen.

0.2 Organization of the Laboratory

Organizationally our laboratory is divided into a number of groups with a certain degree of overlap (both in people and in interest).

VISION: programming machines to see

ROBOTICS: programming mechanical manipulations

LANGUAGE: programs that understand English

PLANNER: implementation of a new programming language

PROGRAM-UNDERSTANDING: programs that understand processes

MATHEMATICS: especially schemata, complexity, theory of computation

MATHLAB: (in cooperation with the group in Project MAC)

"I.T.S.": The A.I. Lab's time-shared computer system

DEVELOPMENTAL PSYCHOLOGY: the educational research group (NSF)

HARDWARE: experimental shop facility

ADMINISTRATION.

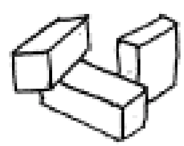
For the past year, the Artificial Intelligence Laboratory has been an independent M.I.T. Laboratory, separate from Project MAC. (The group was started in 1958 as part of the Research Laboratory of Electronics and the M.I.T. Computation Center and later became a group of Project MAC. It has grown to a size and complexity that now needs its own administration.) The funding is mainly from ARPA; the studies in Education and Child Development are supported by the NSF, and there are some smaller support contributions from NASA, NIH, and from some sources of fellowships and assistantships.

In the area of ROBOTICS, we have come to the conclusion that the research directions that have been followed, while extremely productive for Artificial Intelligence research, have not been effective enough in stimulating practical developments in the area of advanced automation. Accordingly, we have proposed to ARPA, in a separate supplementary letter, to consolidate what we now know in the design of a complete and easily copied mini-robotics laboratory system, as a packaged system, so that other centers can get started on applied research in that area.

1. Progress in the Hand-Eye Project

Progress in our laboratory can be seen on different levels. Very specific projects, such as the hand-eye work supported by ARPA, achieve very specific and increasingly ambitious goals. Several years ago we were struggling with the problem of making a computer see anything at all, and were proud of the first demonstration in which the computer could see a visually isolated, clear, well-illuminated shadow-free cube well enough to add it to a tower. Last year our seeing machine was able to look at a complex of mutually obscuring objects, analyze the scene visually, dismantle it and put it together again in mirror image.

> The first scene copied was as shown below. Note that objects obscure one another by way of support and in-front-of relationships. The objects are not of any particular size; both size and position are determined in the copying effort as required.



Since then we have increased its ability to tolerate less than perfect lighting conditions, see shadows, deal with even more obscured objects and so on.

But, our progress appears as quantitatively continuous only when viewed very superficially. A deeper examination shows qualitative changes in the methodology of the work, in the kind of question we ask and answer, and above all in the kind of theoretical conclusions we draw from it. To define the trend of our work and, particularly, to place next year's work in its proper context, we need to review our progress on these more fundamental levels.

So, let us review briefly the progress in vision, this time examining the style of operation of the programs rather than their outer behavior.

Seeing and picking up an isolated object under good conditions does not actually need any of the techniques of programming or theoretical concepts we see as characteristic of "artificial intelligence". It can be handled as an interesting, difficult, but typical problem in classical programming and engineering.

The main initial direction forward from the simple case in our laboratory was the development of what we now call "mini-theories" of important visual phenomena. Some of these concerned problems of an optical character, such as the light distribution at the edge of an object, the deduction of surface curvature from shading, the local structure of shadow lines and so on. Others were of a geometric character, such as the classification of vertices and edges into regions, the dissection of scenes into separate objects, defining relations between objects in three dimensional space.

We consider our programs as belonging to Artificial Intelligence from the time they begin to use such mini-theories. But merely using them is only a beginning: the truly deep questions are connected with <u>how</u> they are used. And the major focus of our work has turned toward defining and understanding phenomena of interaction of diverse kinds of knowledge.

> An early example of such interaction in the vision system occurred in the process of locating objects in space. One straightforward method using focus was used with a more complicated module that depends on heuristically derived conjectures about how objects in the scene support each other. The focus method is less accurate, but is immune to the gross blunders occasionally made by the support-dependent module. When used together one can get both great reliability and great precision.

The simplest way to use mini-theories such as those listed above is separately, for example in a pass-oriented or hierarchical program structure where "optic" mini-theories might be used for line detection, geometric mini-theories for parsing scenes and so on. There is no doubt that useful results can be obtained this way and we have demonstrated some; but it is now clear that the real pay-off is in a different, ultimately more powerful direction, illustrated by the idea of heterarchical program structure.

A fair image of what we mean by this can be obtained by thinking of ways in which human specialists might be integrated into a large organization. The pass-oriented model uses the following pattern for solving a problem: the problem is divided in advance into stages or sub-problems, which are passed out to the specialists; each specialist sees his part of the problem, does what he can with it, passes on the result and is through. Heterarchical organization implies the possibility of communication between specialists' consultations, sending the problem back for further study and so on.

The advantages and disadvantages of each mode of organization are plain enough: the first structure lends itself to orderly work but suffers from rigidity and if there is any flaw in the original plan it will fail at that point. The second structure is much more powerful on condition that the process does not fall into chaotic disorder. Experience of human organizations shows that a free interaction can work only if somewhere in the background is an infrastructure of conventions and understandings. This is equally true of program organization: one of our central technical problems is how to set up such organizational infra-structures.

During the past year it has become clear that our conceptual and technical tools for this task have reached a critical mass. A new style of programming and thinking about programs for Artificial Intelligence has almost explosively spread through our own laboratory and has even already begun to affect the work at other centers of research. The first major example of programming in the new style was T. Winograd's program discussed in last year's proposal to ARPA. Since then, as we anticipated in that proposal, our Vision System has been recast into the new form and several new projects have adopted it from the outset. The following section contains a summary statement of the main features of these new programs. A more detailed account can be found in our forthcoming Progress Report (available 1971), which has taken the form of a monograph on the fundamental problems of Artificial Intelligence (and is therefore a piece of progress in its own right as well as a report on this and other progress). We do want to emphasize our belief that what we are doing really is quite different from the approach of Artificial Intelligence generally followed up to now in other centers and expressed in the recent round of books on the subject. We see the difference as fundamentally this: faced with the apparent diversity of kinds of knowledge, the common approach at other centers is to seek ways to render it more uniform so that very general, "logically clear" methods can be used; our approach is to accept the diversity of knowledge as real and inevitable, and find ways to manage diversity rather than eliminate it.

2. Knowledge and Common Sense

An exciting aspect of the new programming style is that it promises at last to break up the stereotype of the computer's slavish dullness, or "superspeed moron" character. A few years ago, the remarks of the previous section would have been appropriate to describing the differences between men and computer programs; now they contrast the new with the old programs! A good example of a degree of "common-sense" is supplied by the interactions of the programs in the "Blocks World" system that is responsible for the behavior in the attached dialog -- (Appendix). The syntactic and semantic systems generate construction goals, like "Build a steeple." The definition of the goal ("a steeple is a stack which contains two cubes and a pyramid") is converted into a plan -- a step-by-step specification of subgoals -- that is interpreted as a program for building a steeple: find a cube; put another cube on it, put a pyramid on that. Now when a first cube is found, its top might be cluttered with other objects. We do not want the robot -- like the assembly-machine in Chaplin's Modern Times -- to smash the second cube down willy-nilly. But we don't want, either, to have to write into our plan: "find a first cube and remove the things on it" because (1) it is so obvious to say it and (2) it only sweeps the problem under the rug: the program will have to find a clear surface somewhere to put down the junk it takes off the first cube.

Winograd's system faces the problem once and for all. An almost autonomous network of statements and procedures "know" that to put one thing on another there must be a place it will fit. If there is no such place one must make one (or fail). To make a place one must move something else, and (recursively) put it somewhere else where it will fit, etc. But one cannot put something on itself, even if there is room, because it won't be where it was at the future moment of setting it down after moving it, etc. etc. Now, instead of writing this sort of thing into each particular application program, we create once and for all a "micro-world" of knowledge about how things are supported, how supports change when things are moved, and so forth. This knowledge is invoked by the occurrence of patterns either in the outer world, or in goal-statements, whenever they occur, and the common-sense processes intervene and take over the actions until their invoking patterns disappear. For example: there is in the micro-world a statement whose effect is

> "If A is supported by B, and A is moved, then erase from the current description of the situation any statement of the form 'B supports A.'"

Notice the indirect character of this. It is a statement not about the physics of support but about when to forget statements concerning support! (When you paint an object, and then move it, analogous statements should not disappear.) One could conceivably do without this advice, at the cost of recomputing after each change in the world all the relations between all subsets of objects. This is impractical and is a common cause of examples in which a system works on "toy problems" and collapses on real problems. Or one could do without this advice, at the cost of recomputing after each change in the world all the logical consequences of that change; this leads in a different way to collapse. One can make heuristic compromises: motions change geometrical relations but not (usually) other attributes of objects. The art -- and science, eventually -- here is in finding what are the points that are so immediate that one should know them directly, and which can be left to more general but more laborious deductive systems.

Common sense is not magic. If we want our computer to act as though it knows the elementary strategies about physics and geometry, we must give it that knowledge somehow. But we need not do this anew for each program! So our goal is to learn how to refine the ideas in the Blocks World, <u>and</u> the ideas in pattern-matching invocations that make this knowledge engage relevant situations, so that we can keep this "data" permanently in the system. Then <u>any</u> program written therein will automatically behave sensibly in that sphere of activity.

3. Some Features of the New Programming Style

(a) Use of the language PLANNER

Programming languages such as ALGOL are designed to make it easy to express the kind of statement certain classes of users are likely to require: for example, algebraic expressions and repetitive loop structures. In programming for Artificial Intelligence we need to express very easily such instructions:

> To achieve Goal A, set up
> sub-goal B and if this
> fails, try C or D.

PLANNER is a language designed to be highly expressive in talking about just this kind of advice.

Example: An example that illustrates PLANNER's expressive power is that a graduate student, Ira Goldstein, was able to write a PLANNER program rather like the old Gelernter Geometrical-Theorem-Prover -- but not only did Goldstein need only a few days of work; once made, his program could easily be modified to handle classes of proof involving constructions which Gelernter was unable to do in a long period of work. This ability to perform experiments in Artificial Intelligence rapidly and flexibly is a most decisive change in the style and fertility of work.

(b) Automatic Mechanisms for Fallible and Contingent Instructions

The image of a classical numerical program is as a sequence of actions performed one after the other and expected to succeed. In Artificial Intelligence, programming an action might be an attempt that fails. In this case it is necessary to ensure that when it gives up, the micro-world in which it works is not left untidely cluttered. The PLANNER system contains powerful mechanisms to take care of such situations.

Example: The examples already cited of how the Winograd program hangles its Blocks Words is a typical case. If the proposed action includes a statement-erasure the failure back-up can restore the description to its former state with, perhaps, some editing relevant to the failure.

(c) Procedural Description of Knowledge

It is slightly ironic that the most popular approaches to Artificial Intelligence programming force knowledge to be stated in a form of "logic" as assertive propositions such as:

The Box is Red.

Now, there is no problem here, in simple statements about attributes. But other kinds of knowledge are much better stated as procedures (or "programs") rather than as facts. Even such a simple statement as:

"If there are no cars coming, cross the road."

is misleading if translated into a logical implication such as:

For every x, [(x is car) & (x is not coming)] $\Longrightarrow$ [crossing is permissible].

It is more naturally transcribed as:

"Look left, look right, if you haven't seen a car cross."

The distinction between these modes of expression is not merely verbal. A deeper aspect is seen by picturing a "logical theorem prover" trying to <u>prove</u> by resolution or other logical principles that no car is coming! Deeper yet are the consequences in more complex situations: we believe that even in the size of programs we are now using they make a difference between easy programming and very difficult programming. With more complexity we believe that the difference can become one of possible versus impossible.

(d) <u>An Example of Procedural Necessity</u>

The concept of "nearness" is a good example showing <u>procedural definitions</u> to represent knowledge. Everyone knows what "near" means. If we are told that

The car is near the garage.

and

The garage is near the house

then we can be sure that the car is near the house. But we <u>cannot</u> put this "transitivity" into a formal logical system by a rule like

<u>Rule 1</u>: (A near B) and (B near C) $\Rightarrow$ (A near C)

for unrestricted application of this rule would yield absurdities like "1 is near 100" because 1 is near

1.001 and 1.001 is near 1.002, etc., etc.!

It is clear what is needed. In any particular context, "NEAR" is used to represent a certain size range, and if one uses chains of longer than a very few steps, one may get out of that range. The obvious thing to do is to add

Rule 2: Don't use Rule 1 more than (say) four times, unless there is some basis for believing that you are still in the same size range.

This cannot be said in any ordinary "logical system"! No system of "mathematical logic" allows statements inside the logic to talk about the deductive process that uses the logic. This, we claim, is disastrous for intelligent systems because in solving a hard problem one must devote much attention to monitoring and planning the problem-solving activity! We can state Rule 2, or the equivalent, in the PLANNER language. (The only earlier system we know of in which one could do this was Teitelman's PILOT.)

Meta-technical Remark

This attitude is not shared by most other groups working on artificial intelligence, and we feel that the widespread committment to try to represent ordinary reasoning in terms of a "consistent" mathematical logic system is having almost as bad an effect as was the earlier preoccupation with perceptrons and linear-separation clustering algorithms -- another kind of attempt to find a uniform way to represent all different kinds of things.

An interesting sidelight on this is the phenomenon in which many people interpret Godel's theorem as showing a difference between men and machines. It does not. What it says is that any system which is able to discuss its own procedures, and apply these to itself, has the potentiality of deducing some falsehoods. It is perfectly possible to program computers to be able to discuss their own procedures -- Winograd's program comes close to this -- and we believe that this is the best path toward intelligent programs; "Rule 2" is a simple instance. We do not believe that enough is known, today, to make worthwhile the search for an adequate and "consistent" logical intelligent system (that is, one that is <u>inherently</u> unable to have <u>any</u> self-contradictions). This has never been done even for ordinary mathematical arguments, to say nothing of everyday common sense.

What men do is much more PLANNER-like. Suppose that you deduce a contradiction in some argument or find that some plan you expected to work out did not. One then looks backwards and tries to "localize" the trouble. Then one makes a pattern-invoked heuristic rule, to prevent that kind of deduction being made again. "Rule 2" is just such a device. A superb non-trivial example is provided by the way real mathematicians deal with "naive set theory". They do not reject it, as did Bertrand Russell in his attempt to rebuild mathematics without it. Instead, they now have rules like: if your statement resembles Russell's Paradox (because it talks about itself) then be careful; find another way to do it. It is astonishing how few such caveats have been adequate to keep contradictions out of ordinary mathematics. And we believe that among the most important forms of human knowledge are just such rules that indicate which lines of thinking are unsound. Another example: whenever one finds a theoretical outlook which can explain things "too easily", as do mystical concepts of "unity in everything" or the dialectical elements in Freud's Theory (in which many causes can produce opposite effects) one says to himself (unless he is still adolescent): "This method is too good. That means the inconsistency is too close to the purposes I want it for." Then one tries to build up protective knowledge structures for preserving what one can (Freud proposes many valuable new ideas about how knowledge is represented -- and misrepresented!) of the new theory.

In taking this path (and we believe it is the only promising approach, today, to understanding intelligence), we must be very clear about the risks. We must understand that as our programs get better at analyzing their own processes, and incorporate better heuristics for preventing unsound kinds of reasoning from emerging, the possibilities of contradictions become buried more deeply but are not eliminated. We must not be entirely diverted by good empirical results. We must not give up on trying to get a complete theory. It is certainly not inconceivable that we can construct a systematic theory of consistency in a logical area wide enough to cover all important areas of intelligence; one must be on guard against other unwarranted (but common-sense!) pessimistic misinterpretations of Godel-like theories.

(e) Mini-theory Construction as a Technical Goal

We do not see the problem of Artificial Intelligence as one of programming existing knowledge; it also involves the acquisition and classification of new substantive knowledge about such areas as: intentions; excuses; goal structures and so on. A curious feature is that such enquiry insofar as it has been conducted at all in the past, has been the domain of analytical philosophers and literary critics. Their analyses were, however, limited by their lack of computational models, so we are able to go further. Nevertheless, doing so depends on the acquisition, by people working on Artificial Intelligence, of sophistication in areas of thinking far removed from the content of "computer science" courses.

Example: To illustrate the point we give an example from E. Charniak's work on making programs understand narrative. This work was mentioned in last year's proposal, and has recently reached the level of an operational program capable of answering questions such as in the following example. In the left column is a narrative taken from a children's reader. (In its present form the program interacts in a special format; we have translated this into English for simplicity of reading.)

STORY

Jack and Janet are in the house. Jack is holding a box of pencils and a box of paints.

"Janet, see the paints and pencils that Daddy got for us," Jack said.

Janet went to look at them.

"Are the paints for me?" she asked.

"No, the paints are mine," said Jack. "The pencils are for you, Janet."

Janet said to herself," I want the paints."

Jack began to paint a picture of a read airplane. Janet went to look at it.

"Those paints make your airplane look funny," she said. "You could make a good picture of a red airplane with these pencils."

Question: Why did Janet say that the paints were bad?

Answer: She wants the paints.

Comments: The program must interpret "funny" as "bad" in this context (This is not done for it in the input format). Even then Janet really said that the picture was bad, and it is necessary to transfer this to the paints, and then it must know that if you want something another person has, you might make nasty comments about it in order to get it. If we had asked, is the picture funny, the response would have been (in essence) "No, she said so but she had an ulterior motive." To do this the program needs a lot of information about wanting, trading, giving, owning. Statements can not be taken at face value -- translated into simple logical statements. They must be treated as evidence for the program to use to build a model of what really might be happening.

Specific Directions for 1972

(a) Vision

In last year's proposal the main goal for vision was creating a new heterarchical system in PLANNER. This was done by a team consisting of P. Winston, B. Horn and E. Freuder. The new system quickly proved itself by performing a "copy demonstration" in which assemblies of blocks were analyzed, taken apart and re-assembled in mirror image! This represents a very definite advance on the state of this art.

> This demonstration is significantly different from any we have made before in its generality. The earlier demonstrations illustrated techniques that we could (and often later did) use in other contexts. But the demonstration itself was usually rather rigidly unchangeable. The new vision system can be interfaced as it stands to almost any task . . . building towers, putting blocks into boxes etc. etc. Of course it has limitations and needs improvement. But it does not need to be reprogrammed to be used for a very general class of other tasks.

The weakness of the new vision system is its failure to make effective use of all the heterarchical capabilities built into it. Indeed at the moment it works as if it were a series of separate heterarchical programs. But since the facilities for greater interaction are there, it will make continuous progress.

> The most immediate weakness of this sort is a bottleneck of communication between finding lines and interpreting scenes. Work is currently focussed on removing this.

During the next year the system will be extended in the following directions:

(a) A series of new "mini-theories" to enable the system to use (rather than be bothered by!) shadow lines as a source of information.

(b) More interactive line-finding.

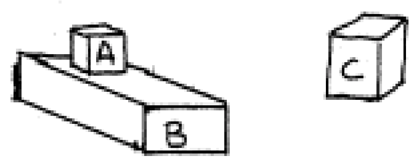
(c) Other representations of objects than as sets of lines.

> The possibility of multiple forms of representation for the basic elements of the system is the consequence of heterarchy and one of the more exciting sources of theoretically interesting problems.

(d) Range-finding.

(e) The ability to confine its analysis to aspects of the scene with a high degree of relevance to the immediate question. At the moment the program still collects and analyzes much more information than it needs.

The following situation illustrates what we have in mind here.



If the machine wants to pick up object A, it soon realizes that its position-locating module first requies some information about the size and position of B. Consequently B is examined as a result of a goal to move A, but C is properly ignored as irrelevant to the goal.

(b) Robotics

We shall concentrate on problems described in our separate letter on mini-robotics.

(c) Language

There are a number of new areas that demand investigation:

> What are the problems in handling other aspects of ordinary grammar?
>
> What new primitives will be needed as new words are added, etc.

These problems are inseparable from those that arise from adding a wider range of "meanings" to the entire semantic system. What happens when we try to extend the system beyond those problems that it now can handle with respect to the blocks world? There are alternative concepts of how to proceed, and let us consider two extremes:

One approach is to add to the blocks world incrementally. It is easy to add new kinds of objects, and properties for them, to the syntactic-semantic-problem solving complex.

It is somewhat harder to add new predicates about spatial relations; for example, "NEXT-TO' might be important in some problems. But such an increment means that one must also add new procedures for taking account of such new elements, be they mentioned explicitly in a main goal derived from a natural language command, or arise internally in a description invoked by an already present theorem. The new procedures represent ways to solve problems and understand situations, but they cannot efficiently be used unless "recommendations" are added to the theorems of the older knowledge base so that the new knowledge can be invoked when (and only when) relevant. Recommendations pointed the other way are needed, too, and modifications must be made to the Meanings" of appropriate words so that the syntactic and semantic systems can handle the nuances associated with the wider spectrum of meanings that the system is now required to deal with.

Presumably, as this kind of incremental extension is made, some changes will be easy -- whenever the system does something that "it should know better than to do" the programmer can intervene and attempt to adjoin new "advice" as a new theorem, recommendation, or entry in the dictionary, or fragment of PROGRAMMAR or PLANNER program. But sometimes this will be found very difficult, because instead of a small addition or change the system will want a new kind of data-structure, or a new heuristic strategy for achieving a new kind of goal.

For example, Winograd's Blocks World does not have a variety of "uses" for the mechanical structures it builds. In particular, it does not have any concept, at present, of multiple-support-upwards. It can deal with situations in which one object supports many others, as several stacks on a cube on a table, but it cannot handle such structures as bridges and arches in which one object is supported by several. When a child builds a high tower that turns out to be unstable, he has ideas about "reinforcing" it by providing multiple support to lower elements.

How hard is it to add such concepts? We do not really know, yet. But it seems clear that it would be hard to do "incrementally" because the data-structure in the original blocks world assumed single support, and all the "theorems" about construction and interference between goals are written in terms of this single support.

Another approach opposed to the incremental would be based on a concept of "microworlds" (referring to suborganizations of knowledge, not to physically separable parts). The original "blocks World" is an elegant mini-theory which, by itself, is a highly satisfactory model of certain kinds of interactions between purposes and physical relations. Is there no way to preserve its effectiveness, intact, in a larger system?

If the new area of meaning were very different it would be much more clear what to do. If, for example, we wanted to talk about what the objects of the blocks world were used for, (boxes are used to store things away that one does not expect to need soon, towers are used to make high structures, pyramids aren't used for anything, etc.) we would have little trouble. We could build a different micro-world about block-structures and their uses, and procedures for designing structures we needed for different purposes, and then turn to the blocks world to find how to make the structures. Presumably, it would not be excessively hard to add to the system a collection of theorems and recommendations that would serve to tie the two separate micro-worlds into a system that could solve problems that need both kinds of knowledge.

Even this has not yet been done, however. So one of our goals is to accumulate experience in finding ways to interconnect two comparatively separate micro-worlds, another is to get experience in the kinds of problems involved in extending an existing micro-world. The first experience is, we predict, somewhat more valuable, for if it can properly be done, the interaction advice may be able to survive the extension of the sub-microworlds involved.

In any case, it is our conviction that there is no plausible alternative to this idea of structuring knowledge into reasonably coherent packages.

(d) PLANNER

Work is in progress on further and better implementation of the language. The limited version known as "micro-planner" is being used in several other centers. We like this, of course, but feel concerned lest a restricted form of the language become too well established, and will take what steps we can to make the best possible form generally available. In particular, we are concerned to make the language available over the ARPA net.

(e) Program-Understanding Programs

This is an area in which rapid growth is very likely. It is closely related to the Natural Language project and to the structure of PLANNER. The explicit goal is to write programs capable of understanding programs.

In addition to the central issues described before, we hope over the next year to go further into systems that understand, in various senses, more about processes. There are a number of different aspects of this general goal that arise over and over again in work on artificial intelligence and, indeed, in many branches of computer science. For example:

> Compilers do not have much idea of what they are doing. Programs are compiled without any sense of the intention of the source program, and the resulting code is produced without any awareness of implications of the specified process.
>
> Operating systems do not know what they are doing, either. They are dealing directly with the execution of processes but have no semantic model of what is involved.

Learning programs like that of Winston produce descriptions of structures from a sequence of examples that have been presented. Much of learning involves the acquisition of new processes. We do not believe that there is necessarily a large difference between acquiring descriptions that represent the structures of objects and descriptions that represent the specification of procedures. But if we are to be able to do the latter, we experience with problem-solving systems that deal directly with the semantics of programs, and this experience is generally lacking.

More generally, in order for a learning program to be versatile, it has to be able to analyze procedures that it learns, to adapt them to new situations, to debug them, etc. We believe that human children who are not able to cope with complicated situations are that way because their process-understanding capability is inadequate.

We are considering several approaches to developing competence in this area of procedure understanding programs.

1. Carl Hewitt has been developing a formalism, called INTENDER, in which one can associate with definitions of procedures semantic statements that make assertions about the effects that the procedural statements are supposed to have. In a sense, this is a sort of formalization of the semantic clues programmers frequently leave in the form of mnemonics and comments. There are a variety of ways to use these statements, ranging from proving that the procedures will have the intended effects to making the procedures adaptable to use by larger systems that can understand and change the code, or translate it into other source languages.

2. Hewitt has also developed a system for "Procedural Abstraction," described in his thesis which observes partial protocols of the behavior of a program. The protocols provide evidence about the different directions that program branches can take, with different data, and the system constructs proposed programs that are minimal models in the sense that they contain just enough structure to account for the behavior. In effect, the system is an abstract learning program that can duplicate behavior that it observes, creating as small a program as it can find (in the sense of doing as much as possible by means of loops). This is the kind of process that would be appropriate, for example, in a system to learn the grammar embodied in some program, by discovering that certain groups of words all have the same effect on the program's branching and goes on to discover what certain kinds of phrases all have similar roles in causing that branching.

3. Ira Goldstein is developing a program capable of understanding very simple programs such as might be written by a beginning student.

A typical kind of task that this program will perform in its elementary stages is to recognize the partial equivalence of the two LOGO programs P1 and P2, the verbal descriptions P3 and P4 and a rough drawing of a square:

P1
```
TO XX
1 FORWARD 100
2 RIGHT 90
3 FORWARD 100
4 RIGHT 90
5 FORWARD 100
6 RIGHT 90
7 FORWARD 100
```

P2
```
TO 77
1 FORWARD 100
2 RIGHT 90
3 77
```

P3 Draw a line, then a right angle, then another line the same length, a right angle, another line, another right angle, another line.

P4 Draw a line, turn a right angle and keep repeating.

4. We have an extensive research program, funded by the NSF, to investigate how children describe, develop and think about their own processes.

(f) <u>Mathematics and Mathlab</u>

In cooperation with the groups in Project MAC, we plan to continue developing relevant background theories in schemata, complexity, and applied mathematics.

(g) <u>ITS and Programming</u>

We are very interested in decentralizing our use of computers. Some ideas for doing this even for control of robotics are mentioned in the mini-robotics proposal. The situation is more straightforward in decentralizing programs of a purely symbolic kind and we will attempt to develop some of our new projects through network access to other computers provided that arranagements can be made to guarantee appropriate service.

(h) <u>Visitors</u>

Our faculty will be reinforced by long visits from M. Blum (Berkeley), Y. Shirai (Electrotechnical Laboratory, Japan) and possibly M. Rabin (Israel).

APPENDIX

Sample Dialog with Winograd's Language Understanding System

The Blocks World was designed by Terry Winograd to serve as a subject domain for the development of a language understanding system. This system is radically more ambitious than earlier attempts to deal with natural language in several respects. It is as much, or more, concerned with the problem of representing the meanings in a discourse than with the grammatical structure; Winograd's thesis asserts that the interactions between meanings and sysntactic structures are so intimate that it is a bad idea to try to draw a definite boundary between the procedures that "parse" a sentence and the procedures that are to be used to draw logical inferences about the meanings carried in the discourse.

Winograd's system is a large, intricate system. It is described in his thesis at several levels of detail and it is impossible to do justice to it in any short summary. Our strategy will be first to present an example of its performance in the blocks world context, and then to discuss some of the mechanisms that are involved in attaining this performance. In the following dialog the system carries out actions and answers questions about a simple scene containing a table, hand, and box, and several blocks and pyramids, as shown on page 8. The lines in lower case were typed in by a person, and the lines in upper case are the computer's response. Indented lines are comments describing some of the features being exhibited.

The dialog was carried out in real time with the response of the "robot" displayed on a CRT screen. Figures are included to show the state of the scene at various points.

1. pick up a big red block.

======= OK.

> The system answers "OK" when it carries out a command. In order to pick up the red block, it had to clear it off by finding a space for the green one and moving the green one away.
>
> Insert Figure on page 9 about here.

2. grasp the pyramid.

======= I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.

> There are three pyramids in the scene, so the program does nothing.

3. find a block which is taller than the one you are holding and put it into the box.

> The system has to figure out what is meant by "one" and "it", as well as using deduction to decide which block it is holding.

====== BY "IT", I ASSUME YOU MEAN THE BLOCK WHICH IS TALLER THAN THE ONE I AM HOLDING.

Printed out to keep us informed.

======= OK.

The hand has to set down the red block to pick up the taller blue one. Figure 4 shows the result of the system's manipulations so far.

Insert Figure 4 about here.

4. what does the box contain?

======= THE BLUE PYRAMID AND THE BLUE BLOCK.

5. what is the pyramid supported by?

Earlier the system responded to "the pyramid" by saying "I don't understand which pyramid you mean." now, it assumes that we mean the one just mentioned.

======= THE BOX.

6. how many blocks are not in the box?

Negatives such as "not" and "no" are handled whether they appear in clauses or noun groups (such as "no boxes").

======= FOUR OF THEM.

7. is at least one of them narrower than the one which I told you to pick up?

The phrase "at least one of them" must be analyzed to refer back to the previous question, and "the one which.. ." must also refer to blocks. Noun groups can have complicated numbers such as "at least one", "more than three", "exactly seven", etc. The memory of past events is used to deduce which block is "the one which I told you to pick up". Notice the complicated grammatical structure in which "the one" is actually the object of a clause embedded two levels deeper.

======= YES, THE RED CUBE.

8. is it supported?

Passives can be used without a "by ..." phrase.

======= YES, BY THE TABLE.

9. can the table pick up blocks?

======= NO.

The semantic rules make this a nonsensical question which therefore must be false.

10. can a pyramid be supported by a block?

======= YES.

The deductive system finds an actual example, so it knows this is possible.

11. can a pyramid support a pyramid?
======= I DON'T KNOW.

The question makes sense, but there is no example in the scene, so there is no easy way to tell without being able to examine the robot's programs.

12. stack up two pyramids.
======= I CAN'T.

The robot tries and fails.

13. the blue pyramid is mine.
======= I UNDERSTAND.

This data is added to the data base.

14. I own blocks which are not red, but I don't own anything which supports a pyramid.
======= I UNDERSTAND.

Two new theorems are created for proving things about "owning".

15. do I own the box?
======= NO.

The box supports whatever it contains, including a pyramid.

16. do I own anything in the box?
======= YES, TWO THINGS: THE BLUE BLOCK AND THE BLUE PYRAMID.

17. will you please stack up both of the red blocks and either a green cube or a pyramid?
======= OK.

Logical connectives such as "and" "or" "either" etc. are handled in both the grammar and semantics. A command is executed even if it looks like a question. The command is carried out by the following series of steps: It puts a green cube on the large red block (note that it chooses the green cube with nothing on it), then removes the small pyramid from the little red cube, so that it can use that cube to complete the stack as specified, resulting in the situation of Figure 5.

Insert Figure 5 about here.

18. which cube is sitting on the table?
======= THE LARGE GREEN ONE WHICH SUPPORTS THE RED PYRAMID.

Ordinarily the system would be satisfied with calling this "a big green cube", but since we asked "which cube" it has to be more specific in naming it. The use of substitute nouns (such as "one") and pronouns in answers makes the discourse more natural.

19. is there a large block behind a pyramid?
======= YES, THREE OF THEM: A LARGE RED ONE, A LARGE GREEN CUBE AND THE BLUE ONE.

20. put a small one onto the green cube which supports a pyramid.
"A small one" must contrast with the earlier phrase "a large block", so this means "a small block".
======= OK.

21. put the littlest pyramid on top of it.
Words like "littlest" are not in the dictionary but are interpreted from the root forms like "little".
======= OK.

22. 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 - ANWHERE ON TOP OF ?
When there are several interpretations and none is clearly better, the system has to ask the user for more information. The paraphrases were included with the dictionary definition of "on", and the request for more information occurs automatically when the heuristics for resolving ambiguities aren't sufficient. A request like this can also occur when the meaning of a pronoun such as "it" or "they" is unclear.
2 ====== THREE OF THEM.

23. had you touched any pyramid before you put the green one on the little cube?
======= YES, THE GREEN ONE.
The system deduces from its memory what event is being referred to, and can relate other events to it in time. The analysis includes complex tenses of verb groups, and there are special facilities for keeping track of the various forms of irregular verbs like "have".

24. when did you pick it up?
======= WHILE I WAS STACKING UP THE RED CUBE, A LARGE RED BLOCK AND A LARGE GREEN CUBE.
A time is described by naming the main goal which was being worked on.

25. why?

The system remembers immediately previous sentences to understand questions like this and the following ones.

======= TO GET RID OF IT.

By keeping track of selected parts of the original subgoal tree, the system has some understanding of its own motives.

26. why did you do that?

======= TO CLEAN OFF THE RED CUBE.

"That" refers to the action mentioned in the answer.

27. why did you clear off that cube?

Here, "that" means the one just referred to.

======= TO PUT IT ON A LARGE GREEN CUBE.

28. why did you do that?

======= TO STACK UP THE RED CUBE , A LARGE RED BLOCK AND A LARGE GREEN CUBE.

29. why did you do that?

======= BECAUSE YOU ASKED ME TO.

It remembers what it did, not how the request was worded.

30. how did you do it?

"It" refers to the action mentioned in the last question, rather than the answer.

======= 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.

This is a description of the top-level subgoals used to achieve the goal.

31. how many objects did you touch while you were doing it?

Time references can refer to a time span, using words like "while".

======= FOUR OF THEM.

32. what did the red cube support before you started to clean it off?

======= THE GREEN PYRAMID.

By remembering what has been moved, it is possible to deduce things about earlier states of the scene. We can use complex nestings of clauses like "before you started to clean it off".

33. there were five blocks to the left of the box then.

"Then" must refer to the time mentioned in the last sentence.

======= NO, ONLY FOUR OF THEM: THE RED CUBE, TWO LARGE GREEN CUBES AND A LARGE RED BLOCK.

Earlier in the dialog, information about "owning" was accepted at face value. But the system knows that it has complete information about about the basic properties of objects, including location. It checks any statements about location or other such properties to see whether it agrees or disagrees.

34. put the blue pyramid on the block in the box.

This is syntactically ambiguous, but is understood without ambiguity since the parsing corresponding to "put the blue pyramid which is on the block in the box." does not make sense in the current scene, while "put the blue pyramid on the block which is in the box" does.

======= OK.

35. is there anything which is bigger than every pyramid but is not as wide as the thing that supports it?

The system understands quantifiers such as "every", "any", "some", etc., and structures such as "bigger than", "as wide as", etc. Here "it" refers to a noun group within the same sentence but two levels up in the parsing tree.

======= YES, THE BLUE BLOCK.

36. does a steeple

SORRY, I DON'T KNOW THE WORD "STEEPLE".

The system interrupts immediately when it sees a word it can't figure out.

37. a "steeple" is a stack which contains two green cubes and a pyramid.

The quote marks indicate that we are defining a new word.

======= I UNDERSTAND.

38. are there any steeples now?

======= NO.

The new definition is used as a description to recognize steeples.

39. build one.

======= OK.

The description given in the definition is enough to deduce a plan for building new objects as well as recognizing them. The word "one" means "a steeple", and the plan is carried out. First, one of the large green cubes is cleared off by removing the red cube and two pyramids. Then the two green cubes and the tall red pyramid are stacked up to fit the description.

40. call the biggest block "superblock".
======= OK.

We can also give objects names.

41. have you picked up superblock since we began?
======= YES.

42. why did you drop it?
======= BY "IT", I ASSUME YOU MEAN THE SUPERBLOCK.
======= I CAN'T EXPLAIN A NON-EXISTENT EVENT.

There is no record in memory of any block being dropped.

43. is there anything to the right of the red pyramid?
======= YES, FOUR THINGS: THE BOX , THE RED CUBE, THE BLUE PYRAMID AND THE BLOCK NAMED SUPERBLOCK

Once an object has a name, the name is used to help identify it in responses.

44. thank you.
====== YOU'RE WELCOME!

Idioms can be defined without modifying the grammar.

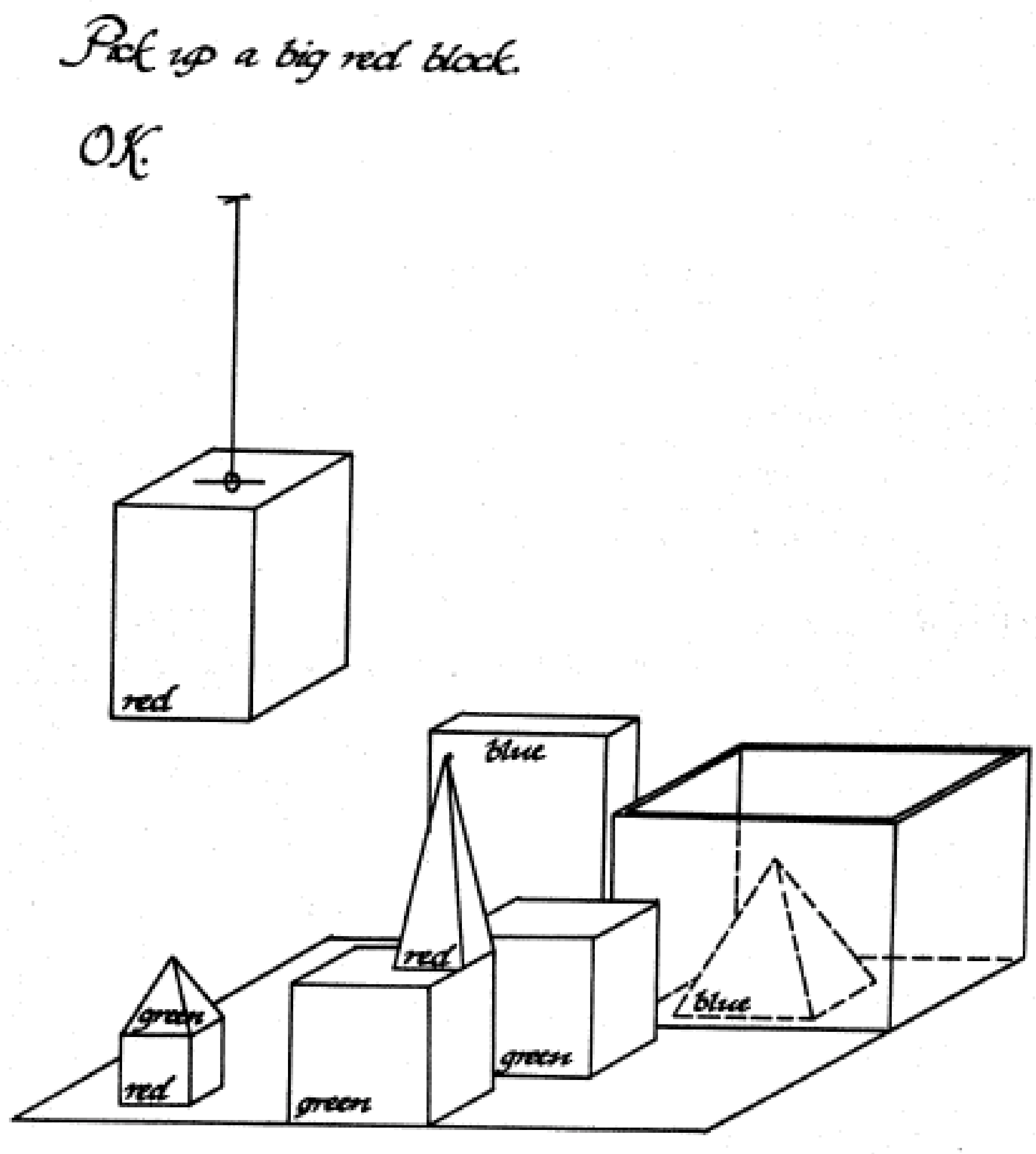

Pick up a big red block.
OK.
red
blue
red
green
red
green
green
blue

Figure 3

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.

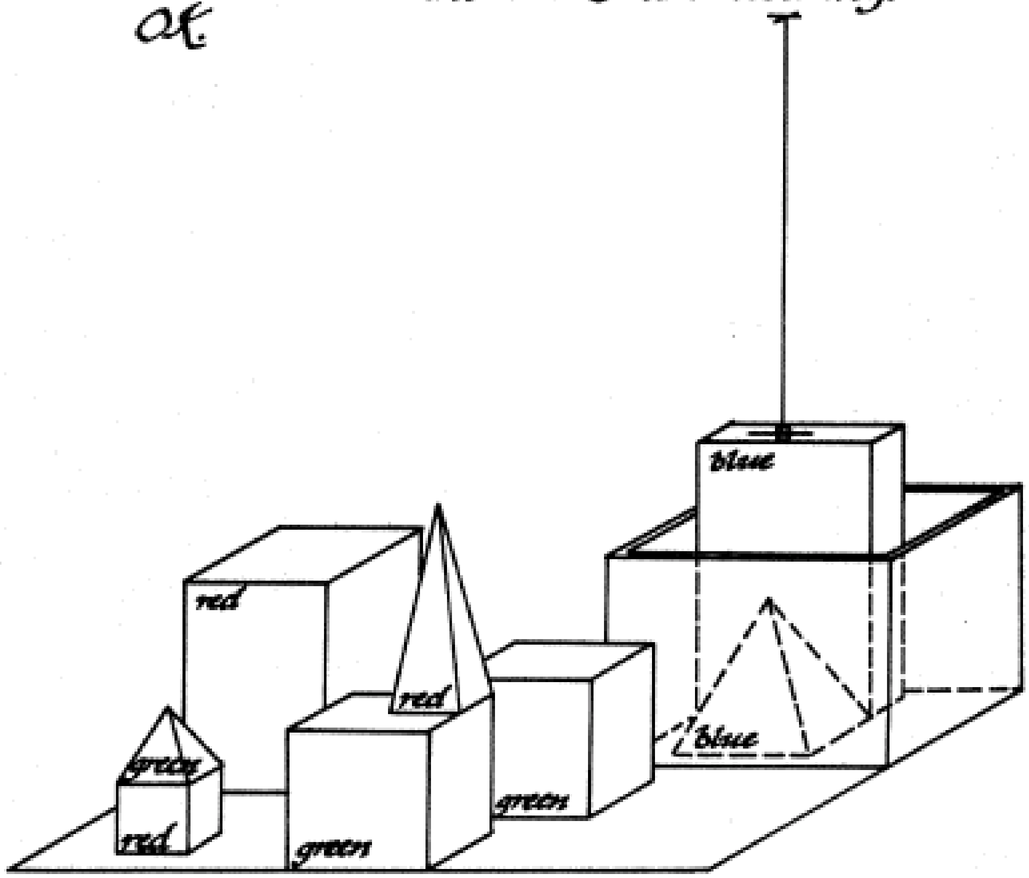


Figure 4

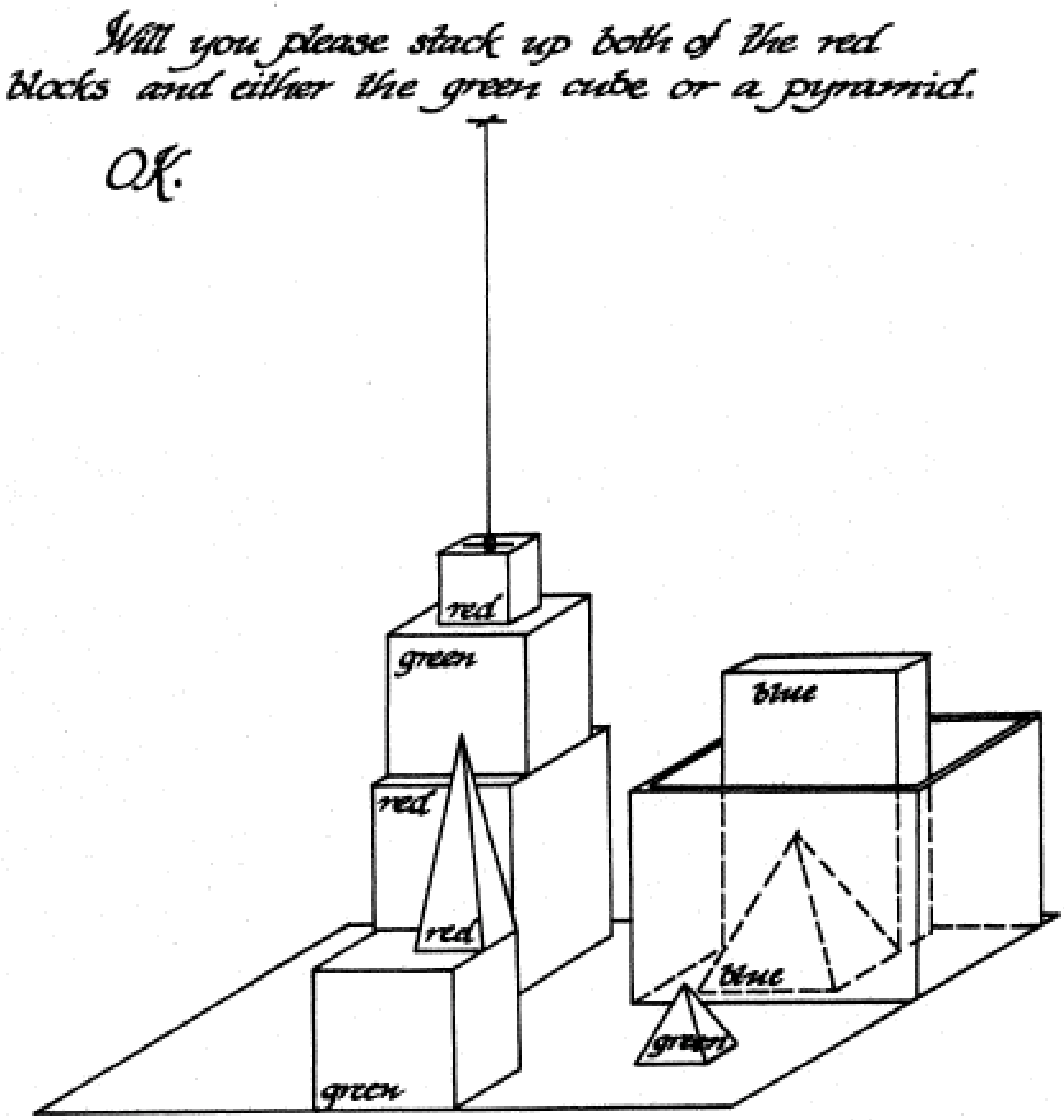
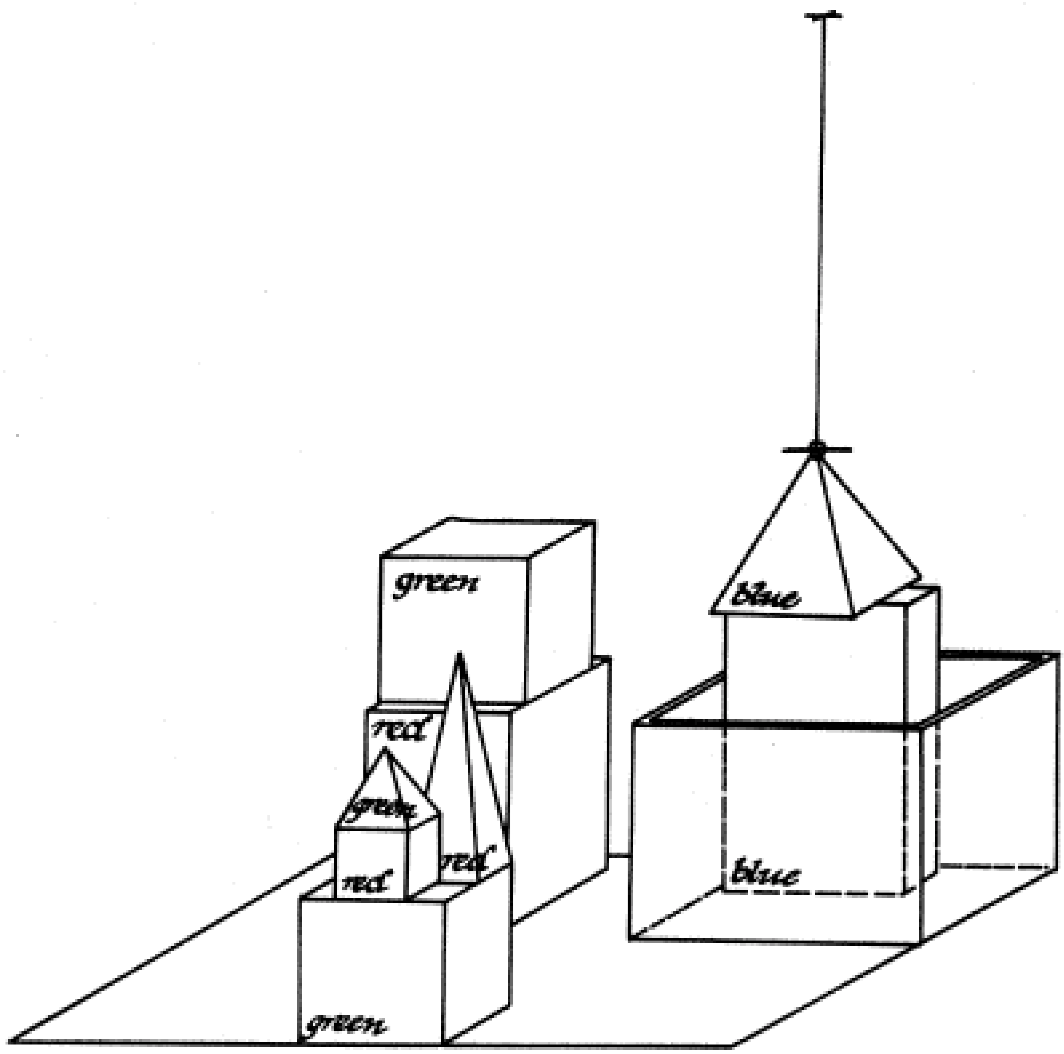

Will you please stack up both of the red
blocks and either the green cube or a pyramid.
OK.
red
green
red
red
green
blue
blue
green

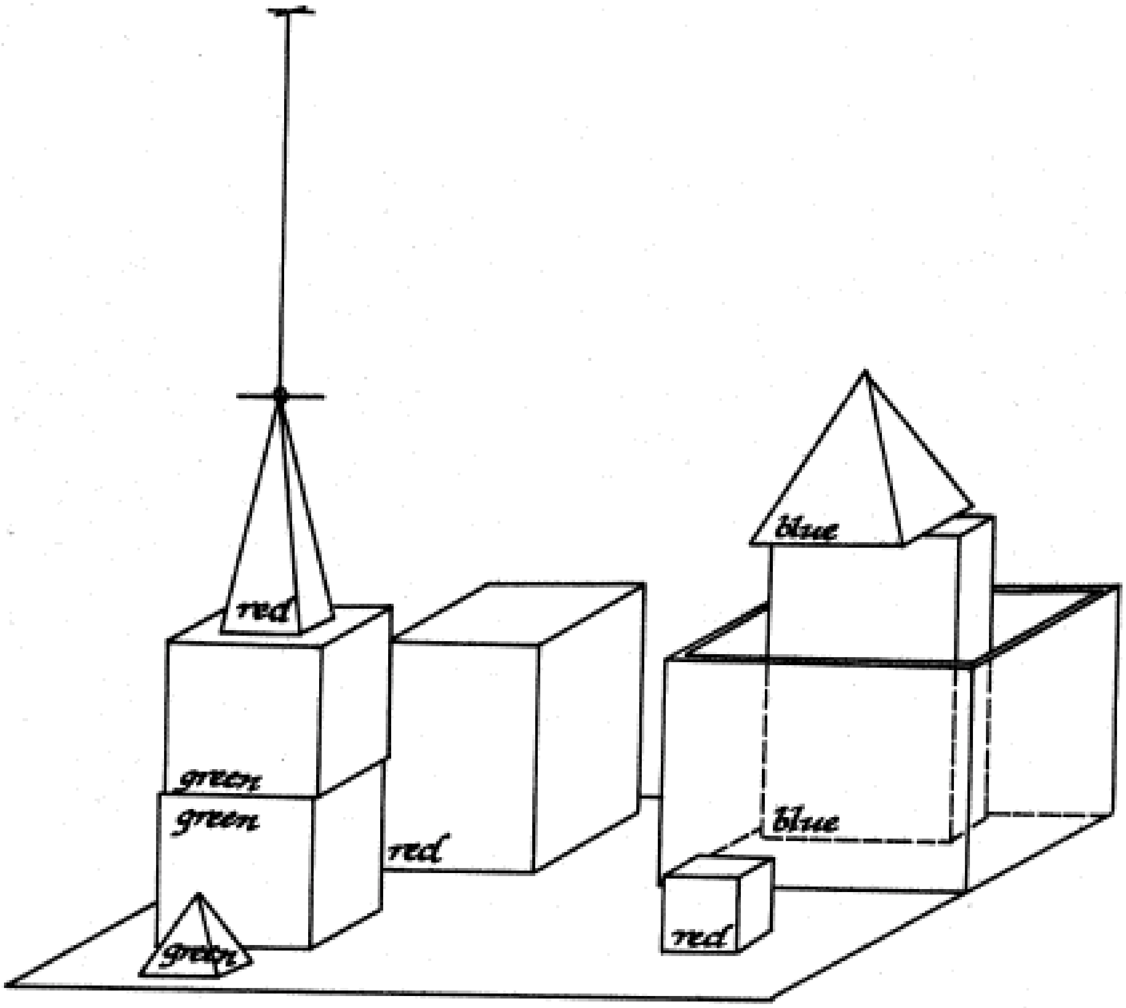
Figure 5

Build one.



Build one.

OK.



47. thank you.

YOU'RE WELCOME!

The special idiom-defining capabilities allow us to include such pleasantries without modifying the grammar.