March 1971

HETERARCHY IN THE M. I. T. ROBOT

by

Patrick H. Winston

FLASH #8

VISION GROUP

ARTIFICIAL INTELLIGENCE LABORATORY

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Work reported herein was conducted at the Artificial Intelligence Laboratory, an M.I.T. research program supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0002.

Introduction

In the early days of artificial intelligence there was tendency to write programs conducive to the study of particular dimensions of intelligence. A great deal was learned from these programs including knowledge about planning, subgoal formation, heuristic measures and methods, tree search and the like.

With this experience we are now at M.I.T. going into projects that seem to require the integration of all these methods and ideas into powerful, general systems. We are tackling problems like natural language comprehension and robot vision with the hope and expectation that the work will lead us beyond previous efforts to exhibit particular forms of intelligence long recognized and into an understanding of those aspects of intelligence not even named as yet.

Although the way to go is not at all clear, it is characteristic of our science to experiment as well as speculate. It is only natural then that we are working with a real robot that in a limited way perceives, understands, and manipulates the simple objects we put in front of its eye.

Now to talk of understanding an environment, even a simple environment like that of the robot, one must have operational measures of expertise. Otherwise one faces endless haggling about just what it means for a system to understand. Three particular abilities which we feel require and demonstrate a degree of understanding are the ability to describe, to learn, and to copy. Figures 1, 2, and 3 will help explain exactly what I mean.

The machine's problem in figure 1 is to build a natural, heirarchically ordered description of the scene like that a human would produce <1>. Note that the configuration shown is quite a complex scene containing objects of various shapes which relate to one another in a variety of ways. From it we want something like this multi-level description:

> The scene has seven objects.
> All but one is shaped like a brick.
>> In the foreground there is a tower consisting of three objects.
>> The tower is in front of a bridge.
>> The bridge supports a wedge at one end.
>>> The tower consists of medium sized bricks.
>>> The bridge consists of two medium sized bricks, carefully aligned with the long board.

.

.

.

.

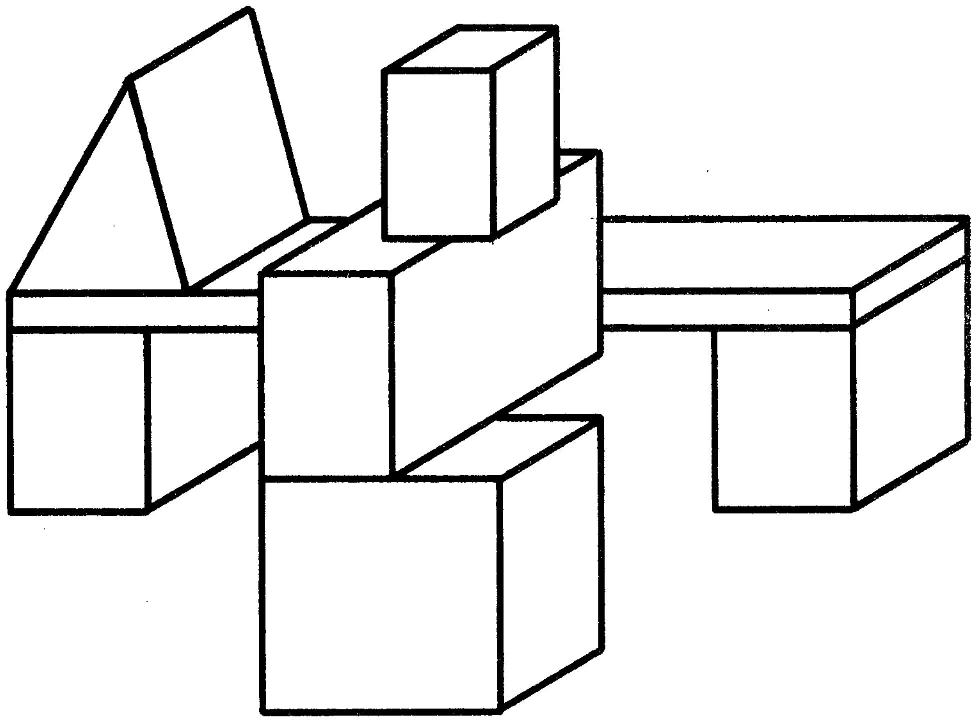

Figure 1: A scene of the sort now understood by drawing analysis programs.

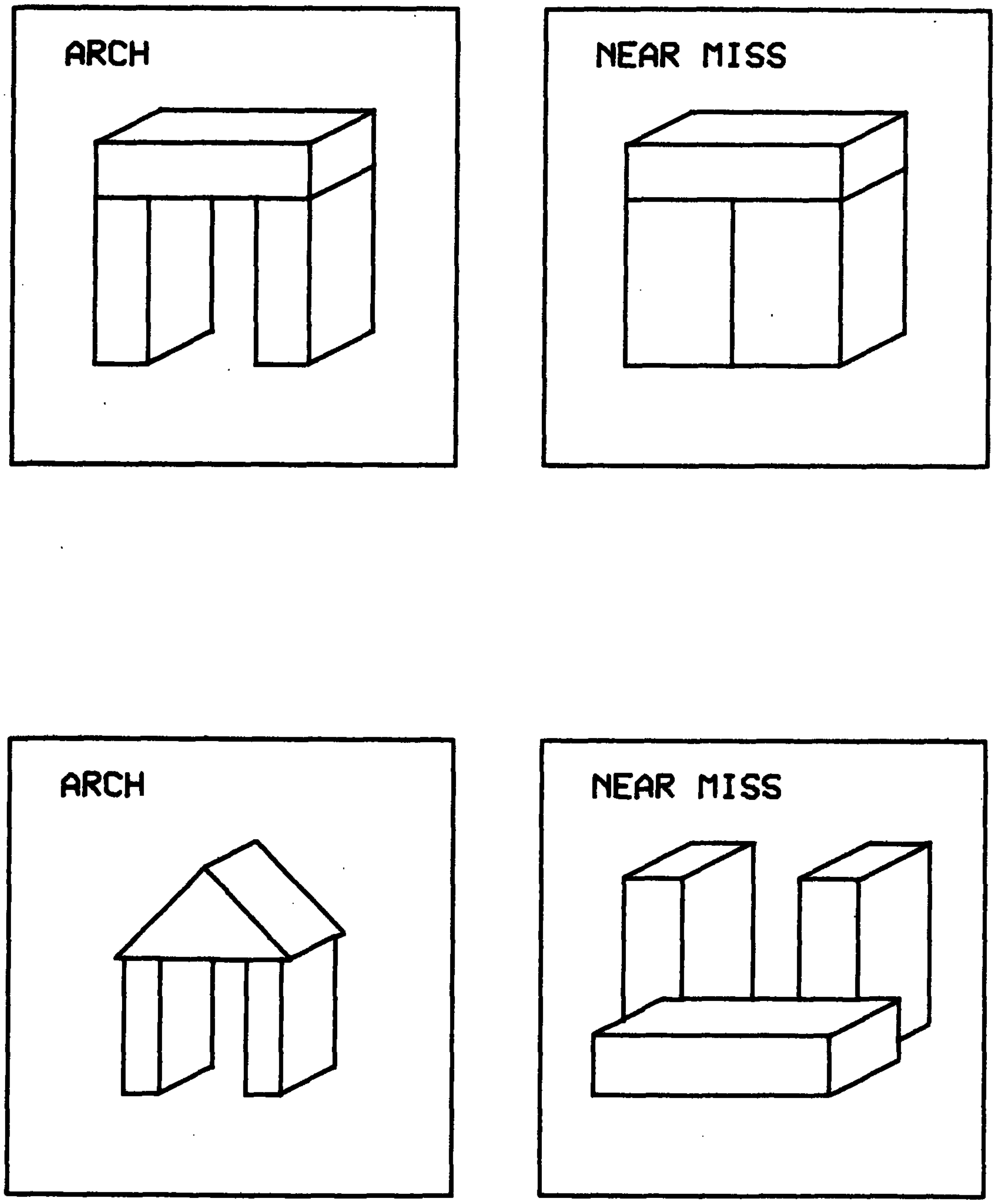

Figure 2: Drawings used in teaching the machine the notion of ARCH.

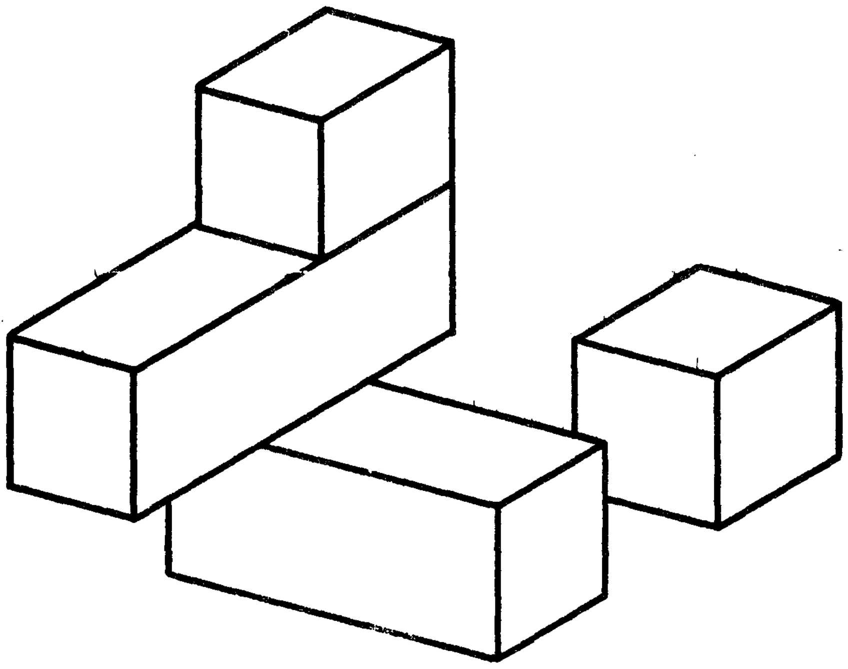

Figure 3: Configuration typical of those copied by the M.I.T robot. Note that objects obscure one another.

Figure 2 defines the particular special kind of learning that I have in mind for the second measure of competence <1>. One presents the machine with a sequence of samples, in this case both examples and near misses to the notion of ARCH.

From these the machine is to create an abstract model with which it can later identify examples of the ARCH concept. The model must be good enough to allow recognition of partially obscured arches and arches differing from those in the learning sequence by way of changes in size and orientation.

The third measure of understanding and the one I want to concentrate on here, is the ability to copy a configuration from spare parts. The world as our system sees it consists of white, plane-faced objects organized into scenes such as the one shown in figure 3. At the moment, we work primarily with bricks and wedges, but many of our modules are quite general and deal successfully with other shapes. Unlike some other systems, ours makes no assumption that the objects are restricted to a fixed set of sizes. Moreover the system expects to see partially hidden objects. Indeed it thrives on configurations where the objects are on top of or in front of one another.

For us this copying task is harder to implement than the previous two tasks because we insist on total machine selfreliance. It does not begin with a line drawing as we usually allow in our other work. Instead we want it to make its own drawing using a random access image dissector camera to gather information directly from the objects themselves. Many attempts at programming for this task failed and no respectable line drawings were produced using our vidisector until Thomas Binford <2> and Arnold Griffith <3> independantly completed important line finders.

The problems associated with all three of these indexes of competence have been under attack for many years now. Some have proved unexpectedly hard, but within the last year all three of the capabilities sketched out here have been demonstrated. I concentrate here on the achievements leading to the ability to copy because I think they best illustrate how this work not only confirms the need for forward-looking ideas in system organization but also contributes to the development of a theory for artificial intelligence.

The Basic Robot Modules

In order to establish a frame of reference for further discussion, I must briefly outline the sort of things that go on in the mind of our seeing robot.

The first thing is the very difficult job of transforming the array of a million or so noisy intensity samples one gets from the camera into the respectable line drawing needed by other modules. Since one must fight all flavors of optical distortion and electron noise, and non-uniform camera sensitivity, it is no embarrassment that our line finding modules rarely find all the lines in a given scene.

In any case, the drawing must be represented in the machine in a way that is convenient for other procedures. Preparing and maintaining such a representation is the job of a module I will simply call the bookkeeper. This program uses essential ideas contributed by H. N. Mahabala <4>.

Beyond this, the system has a module good at partitioning the regions defined by the lines into a set of bodies. It is the job of such a body finder to report that a scene appears to have a certain number of bodies or objects and to indicate the regions associated with each.

Other modules in turn describe various properties of the objects. One module looks for relationships between objects corresponding roughly to our human notions of on-top-of, in-front-of, and aligned-with. This is the job of the body-relationship-describer. Its skill is useful to a module that finds XYZ coordinates and another that speculates on the size of each body. These are the body-position-finder and the body-dimension-finder modules.

There remains in a skeletal system only a structure construction module that plans and executes the steps necessary to build a copy from spare parts.

Now early in this research we believed that it might be possible to write good general programs for each of these jobs and arrange them together under an executive program that would use them to cause an orderly progression of data from intensity values on one end to arm-and-hand commands on the other. However we cannot make such a system work, and we do not believe that such a system has good prospects. The reason is that the system has so much to do, the probability of getting through the whole job without oversight or error is small, even with very good modules.

Heterarchical Organization

To us the way to success therefore lies in a set of attitudes toward system organization that has come to be called heterarchical organization. At the moment, the term remains only vaguely defined, but it certainly includes the following ideas:

1. A powerful vision system must contain or have access to lots of knowledge. Some of the knowledge is apt to be special purpose and some of it quite general. In our system, for example, we have some special knowledge that is useful only for bricks and some more general knowledge that applies to almost all simple plane-faced objects.

2. A powerful system needs not only a variety of good methods, but also knowledge about those methods sufficient to judge when they are inappropriate.

3. If a module perceives some other module has erred and has provided false information, it must be able to complain and request a review. This objection might be that a body of unlikely shape has been proposed or perhaps that a crucial line has likely been overlooked.

4. Communications between modules must be smooth and natural for many reasons, not the least of which is the fact that many programmers must cooperate in such a large project. So far the system represents direct contributions in code from Thomas Binford, Berthold Horn, Arnold Griffith, Eugene Freuder, David Silver, and Patrick Winston. We expect this group to expand quickly now that the skeletal system is complete.

Taken together, these points suggest something quite different from a simple row of processes, each passing over the data base, doing its job, handing control to the next in line, and retiring. Instead one must think more of an interacting community of processes, some narrow experts, others broad generalists, and still others in the role of critics. As the system becomes more powerful, we expect to see more flow of advice, complaints, and requests for help.

To clarify these somewhat general and abstract remarks, I now describe two relevant parts of our system. The first of these is the part that judges how regions go together, the body finder.

Our general method for region conglomeration is a descendant of a program conceived and implemented by Adolfo Guzman <5>. His technique, stripped of all embellishments, is simply to make two passes over the representation of the line drawing, the first producing local evidence from the vertex configurations and the second gathering together and weighing that evidence.

The local evidence pass recognizes the 8 vertex types illustrated in figure 4. Of these the T, K, and L are ignored, while the others each produce one or more units of evidence for believing two adjacent regions belong to the same body. These units of evidence are called links. The second pass then observes which regions are joined by two or more links and announces that they belong to the same body.

Figure 5 shows how this scheme easily handles a simple two brick situation. By iterating the second conglomeration pass, it can handle much more complicated scenes. But attempts to make it fully as good as people at this job have led only to unsatisfying hodge-podge collections of weaker and weaker heuristics that subtract, rather than add to the basic elegance of the scheme. This does not discourage us, however, because part of our philosophy of approach holds that such general schemes without any special knowledge or access to advice cannot do more than a good job.

To see instead how this general program can contribute to the system positively while sharing its job with specialists, consider for a moment the K type vertex. Guzman did not use it in his basic algorithm because without a context, the vertex suggests two equally likely, but incompatible possibilities. Figure 6 illustrates the usual case where two aligned bodies each contribute a pair of adjacent regions to the set of four regions that surround all K's. But which of two pairings is correct cannot be

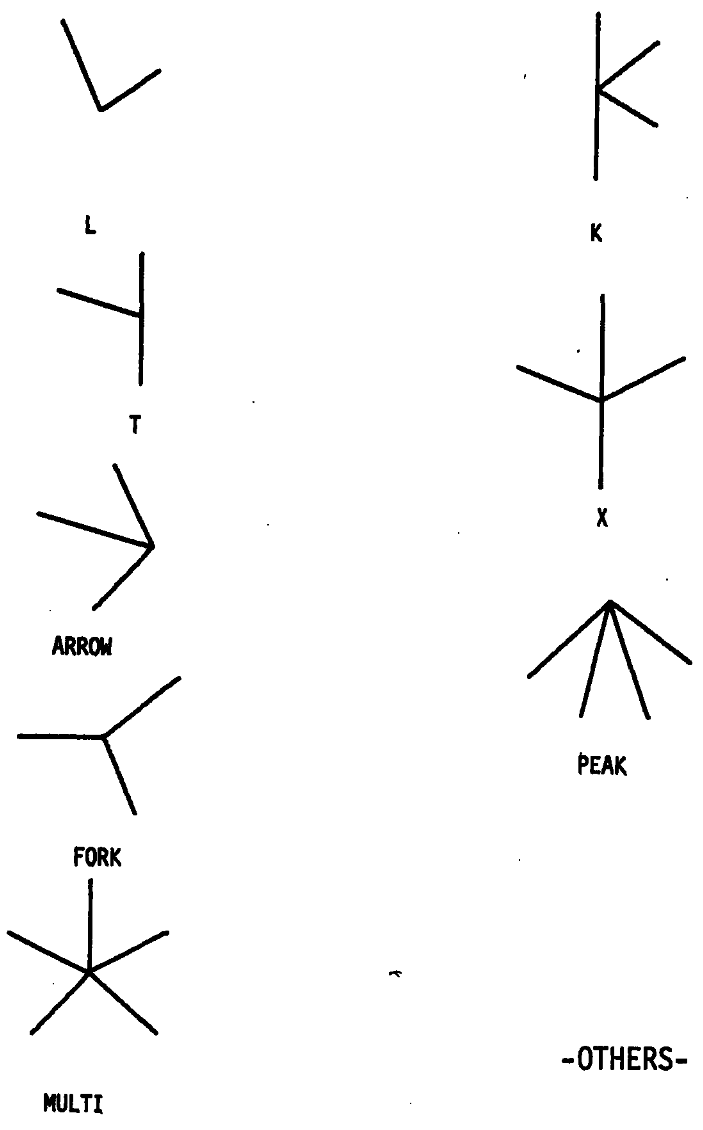

Figure 4: Vertex types as defined by Guzman.

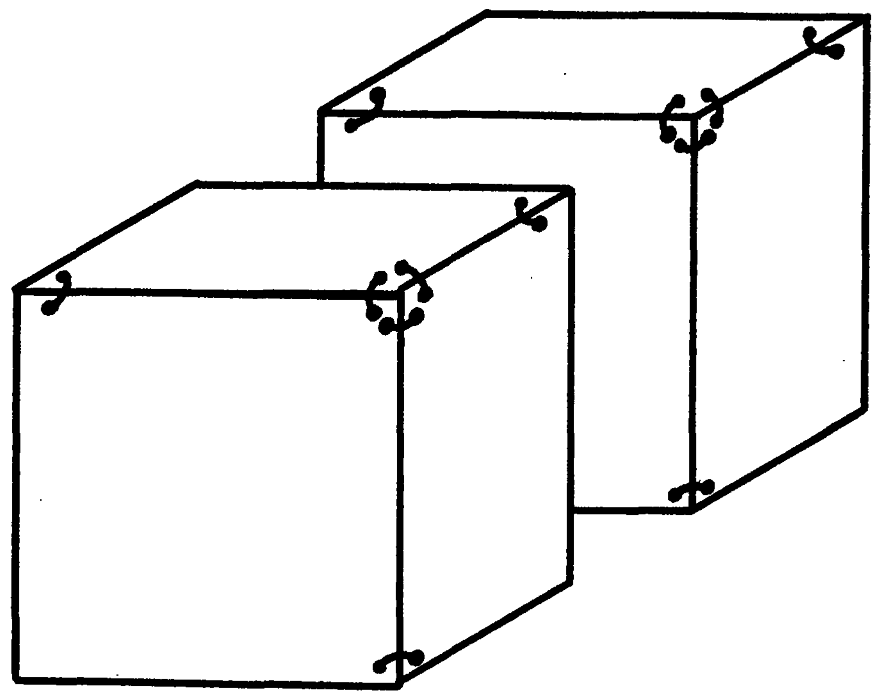

Figure 5: Bodies are found through a process that uses local vertex evidence to link regions together.

determined locally.

Of course there are many ways one might attempt to resolve the ambiguity and get solid linking evidence from the otherwise ambiguous K. But here I only want to observe that K's are frequently resolvable through the use of a specialist for unobscured bricks. This program's only use is the discovery of three adjacent parallelograms compatible with a brick interpretation. It is thus a region conglomerator only under special, easy circumstances. But see how this special case program easily resolves the ambiguous K's in figure 6. Once the supported bricks are seen, the special case program may advise the general case program that the K's have only one interpretation compatible with its findings. Thus the former ambiguity gives way to solid general evidence.

This last example shows how special and general programs work together in our system to accomplish the same task. The next illustrates how another chain of communication extends from a distinct critic through a hypothesizer and on to another specialist.

As previously mentioned, the line finder cannot and should not be expected to find all lines. Rather, it should only report the ones it is reasonably sure of for the analysis of other programs with more knowledge. It can be expected to occasionally miss lines such as the ones dashed in figures 7 and 8 since the illumination gradient across

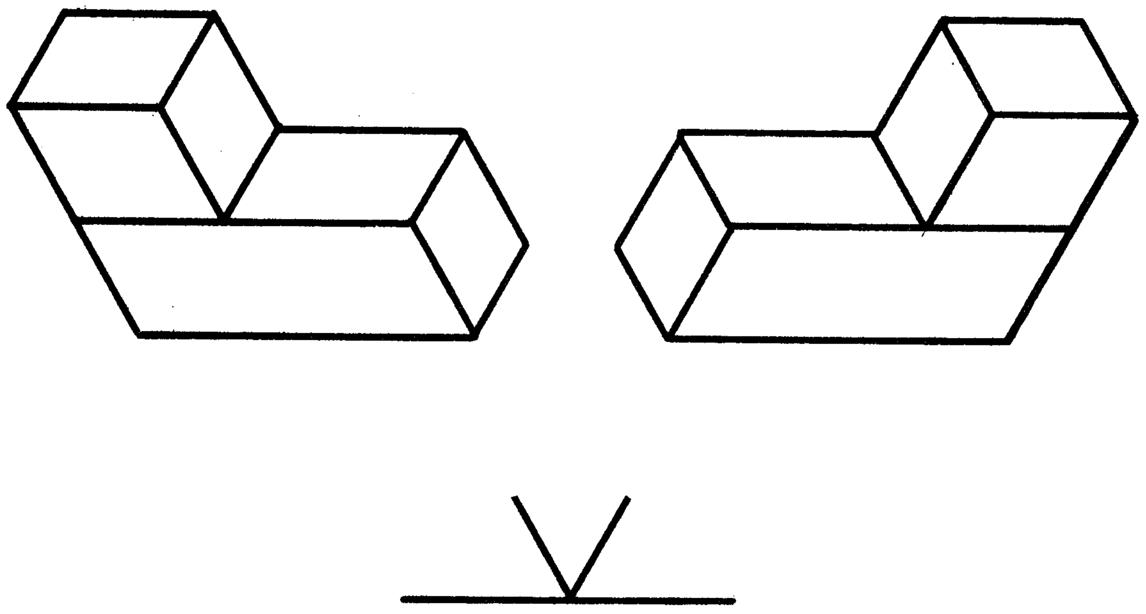

Figure 6: K-type vertexes suggest two bodies but the proper pairing of regions cannot be determined locally.

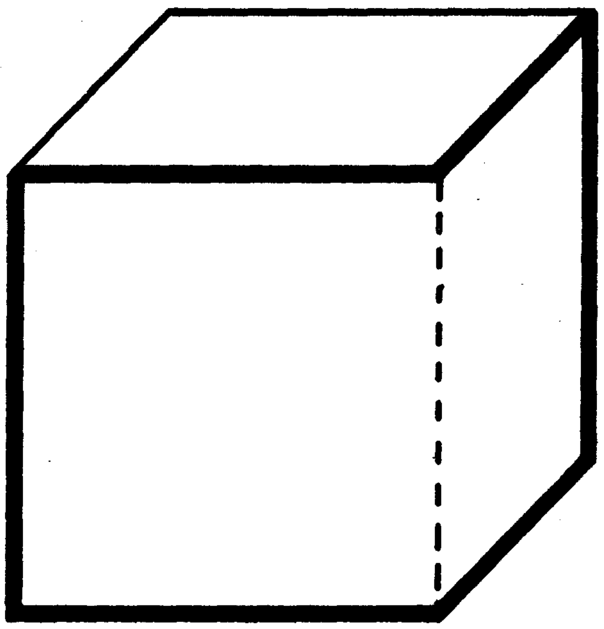

Figure 7: A common type of missing line. The heavy lines outline an region which the region critic finds objectionable.

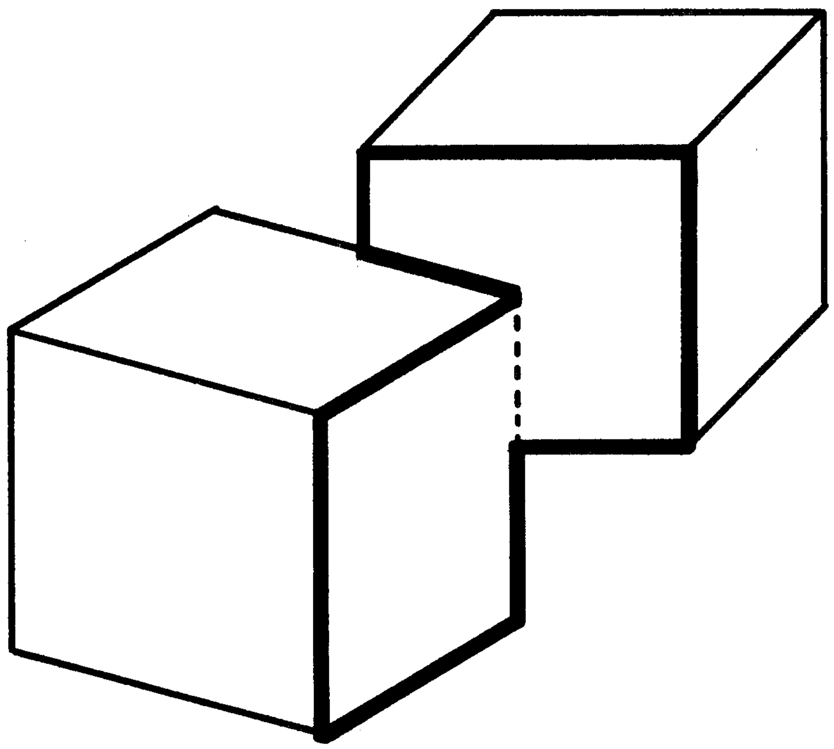

Figure 8: The double L missing line situation.

such lines may be quite small.

Presently the bookkeeper is the system's first line of defense against such oversights. At the moment it has little sophistication in its complaint mechanism. It simply objects to concave regions. The more concavities it notices along a region's periphery, the stronger it objects. Of course concavity only suggests the possibility of missing line error. But if a region genuinely deserves to be concave, the complaint leads to no action and nothing is lost.

Complaints are received by proposers that have buried in their code implicit knowledge of both our line finder and of likely line configurations. One proposer, again a specialist, is designed to react to a missing interior line of a brick. Hence when the peculiar concave shape outlined in figure 7 is observed, this proposer instantly hypothesizes a line which divides the region into two parallelograms.

A more general proposer is good at the sort of doubly concave region heavily outlined in figure 8. Its effort is to find a pair of L type vertexes, one of which has a line whose extension passes through the other vertex. This extension is then the line proposed.

Both of these proposers appeal to a line verifier which closely inspects a small region about the proposed line. One can afford enormously involved computation here because the program only looks at a small portion of the scene. Should a line be considered likely, the verifier reports its presence back to the bookkeeper which initiated the entire chain of events.

The PLANNER Language

It goes without saying that our work places heavy demands on the programming languages we use. LISP, our favorite for many years, is now yielding to the more adequate PLANNER of Carl Hewitt <6>. PLANNER has the features of LISP plus a number of improvements, three of which particularly facilitate our effort. These are roughly the following:

1. Adequate pattern matching and data management. This has greatly shortened program length and debugging time.

2. Automatic backup. If a subroutine fails, all action is unwound back to a specified point where a choice was made. Alternative is selected, and control then passes forward again. This means that if one line of reasoning leads to a dead-end, all its effects are withdrawn and alternatives are tried.

3. Goal orientation. The language allows subroutines to be called through their purpose rather than their name. By analogy this is like the ability to step into a room of strangers and to say "I need to have some boards nailed together. Is anyone good at that?" One need not know in advance who the appropriate people for carpentry are.

We believe that our work will emphasize the need for features such as these and lead to still further advances in language architecture as well as contributions to an overall theory of intelligence.

Conclusion

Our effort in robot research is to get at the problem of organizing a large, highly coupled system that features many different abilities and kinds of knowledge. The examples I have given here represent only our first attempts at breaking away from straightforward design into arrangements that cause us to squarely face the problems of big system design. We soon hope to add more and more specialists and more and more module complaint, advice, and request channels. On the immediate horizon are more than a dozen possibilities, among which are several alternatives to our current monocular object location scheme as well as some specialists that deal primarily with wedge processing.

REFERENCES

(1) Patrick H. Winston, "Learning Structural Descriptions from Examples," Report MAC-TR-76 (Thesis)(Cambridge, Mass.: Project MAC, MIT, September 1970).

(2) Annette Herskovits and Thomas O. Binford, "On Boundary Detection," Artificial Intelligence Laboratory Memo. No. 183 (Cambridge, Mass.: Artificial Intelligence Laboratory, MIT, July 1970).

(3) Arnold Griffith, "Computer Recognition of Prismatic Solids," Report MAC-TR-73 (Thesis)(Cambridge, Mass.: Project MAC, MIT, August 1970)..

(4) H. N. Mahabala, "Preprocessor for Programs which Recognize Scenes," Artificial Intelligence Laboratory Memo. No. 177 (Cambridge, Mass.: Artificial Intelligence Laboratory, MIT, August 1969).

(5) Adolfo Guzman, "Computer Recognition of Three-Dimensional Objects in a Visual Scene," Report MAC-TR-59 (Thesis)(Cambridge, Mass.: Project MAC, MIT, December 1968).

(6) Carl Hewitt, "Description and Theoretical Analysis (Using Schemas) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot," Thesis, Department of Mathematics (Cambridge, Mass.: MIT, January 1971).