Marvin Minsky

The limitations of using languages for description

Estratto da
Linguaggi nella società e nella tecnica

EDIZIONI DI COMUNITÀ - MILANO
1970

Marvin Minsky

The limitations of using languages for description

It is time to consider theories that can account both for language and for imagery. Computer models are being constructed that operate in these areas; they are of irritating complexity but of irrefutable performance. One might attack their « validity » as psychological theories on the grounds that they are very arbitrary in their details. No matter! These working models embody a flood of new ideas about the meaning of meaning – ideas that promise, I am sure, to revitalize both psychology and linguistics.

Language is very good for telling stories, but it is not very good for explaining how complicated machines work. While it is very natural to represent a linear sequence of occurrences by a linear sequence of linguistic events, the descriptive methods that speakers and writers have developed do not work well in explaining systems with complicated interactions.

Since language is important in thought as well as in communication we would like to know how difficulties in expression are associated with difficulties in thinking. We are particularly concerned with thinking about Thinking. Since Thought is the behavior of an exceptionally complicated machine, any limitation on our ability to discuss complicated machines is certain to have grave effects upon our own psychological theories and, indeed, upon the introspectives experiences that lead us to those theories.

This is an essay of « half-baked » ideas relating to that subject. When talking about language I do not mean to talk merely about the output of the linguistic machine in our brains, but about what that sort of machine might be like. Of course, we have almost no direct knowledge of how it works on the mechanical level, so

we must make inference from its behavior. Because the only other machines that show substantial linguistic activity are certain computer programs, I will construct parallels and analogies with those programs. I hope readers will agree that we have long passed the era when it was too simple-minded to propose such comparisons; the computer models are already *more* sophisticated and demonstrably workable than any available alternatives.

The description of images, visual and geometric, poses the most familiar example of what seems to be an expressive limitation. Is it merely the one-dimensional character of language that makes it hard to describe pictures? The line by line dissection of television is a logical but obviously unacceptable solution to this. However, as we shall see, there are reasonably good ways to describe visual scenery with symbolic expressions, and though these expressions are not particularly « linguistic », we believe that their use would entail much the same sort of mental problem-solving organization.

Before discussing images we will talk about describing computer programs; here too, one has a simple linear representation: the program's typed text. But again – and this is not so widely appreciated – this linear representation can be an absurdly obscure way to describe what the program « does ». Programs are more important than one might think, as they are our only tool for describing really complex systems, so we will talk first about describing programs, then about some possible relations between programs and mental events, and finally return to the subject of visual images.

The interior monologue

One often hears accounts of the sudden moment of a great discovery. We are told how a critical insight comes without relevant conscious context, hence some complicated unconscious mechanism must be responsible. If you ask people how they solve simple problems and make trivial discoveries you will obtain such

a variety of explanations and non-explanations that you must conclude that people know very little about how their minds work even in simple situations.

It is curious that people should feel that there is something remarkable about the unconsciousness of their own mechanisms. When a machine solves a problem, it does not occur to us that it ought to be able to tell us how it did it. We are trained not to expect from machines anything resembling conscious thought – that is, an expressible awareness of its own activity. Let us explore this question by analogy; by observing how programmers keep track of programs. This is a similar activity, but is more observable.

In developing a large problem-solving program, using new and unfamiliar methods, the inventor usually has trouble passing from a good idea to a working system. He keeps discovering unexpected interactions among different decision-making procedures, priorities and orderings of doing things, and conventions for representing data, processes and intercommunications. Keeping track of all this becomes so complicated that the inventor turns to the computer itself for help, and uses a debugging system of his own or another's making. The debugging system contains model environments – test programs – for his procedures and it makes elaborate records of things that happen when his program runs. If the program goes astray, these records provide information about the sequence of activities that got it into its bad state.

Thus, in the debugging stages of program development, there becomes available one ingredient of self-awareness: information about the recent history and goals of the process. Another ingredient, ability to ask and use the answers to questions about this recent history, is not available to the program, but only to the programmer. In some modern heuristic programs some such information is indeed accessible, and those programs may be considered to know, at least in a primitive way, something about what they are doing. For examples, consider the papers of Slagle, Gelernter et al. and Newell et al., all described in [1]. Each of those programs has one means or another to detect whether a

new problem is (probably) more difficult than, or is a disguised form of, one of the earlier problems that have lead it to its current state.

After the program is debugged, the programmer removes his diagnostic features. The various tests, traps, breakpoints, push-down lists and interpreters all consume time and memory. Wherever he can, he replaces standard methods (that often preserve unneeded data) by more efficient but trickier methods. Interpreters – that is, programs which obey rules written out explicitly in symbolic form – are replaced by compiled programs – in which the rules are implicit in the program structure, and are written in more efficient but much less transparent language. Once the program gets into this form, it had better work well, for the designer will have much more trouble understanding what happened should it go wrong.

This account of programming practices is intended, of course, to suggest an analogous process in the mental process of learning a skill. Something like it probably happens in the transition from novice to expert. At first, one may not do very well at a task but can say a good deal about how he does it. The expert does much better. But to the extent that his efficiency depends on removing interpreters and compiling efficient procedures for the debugged parts of his performance, he will become less conscious of what he is doing. To be sure, there are examples of experts who seem to explain their actions very well – but I suspect that usually this reflects an entirely different talent, the ability to invent good heuristic explanations, and not any exceptionally direct ability to observe one's own programs in action. The great mathematicians' discussions of their own creative processes do not compare especially favorably with those of laymen. I tried to persuade Norbert Wiener, near the end of his career, to attempt such an explanation but he refused, saying he would rather produce another first rate theorem than another third-rate psychological theory.

Most of the simplest cybernetic self-improvement schemes are unsuitable for symbolic use. Consider the quasi-numerical optimization or « tuning-up » techniques such as one finds in Samuel's

checker-player [1] or in the perceptron [2]. In that sort of scheme the « learning » is embodied in the modification of the values of numerical coefficients – rather than in symbolic expressions or in substantial changes in the structure of the procedure. Consequently one can state little reason for an improvement beyond that « it works ». This makes decisions easy to make. But if one wants to be able to use the results of exploration of part of the game tree in deciding where and how to make further explorations, one needs instead a summary analysis of the position's strengths and weaknesses. Present chess programs do not do this, but they generate nearly enough information to make it a reasonable next step. See [3].

While I doubt that « tuning-up » plays a large role in abstract thinking, one is tempted to suppose that it might play significant parts in, for example, the parameters of motor activities. Thus one would expect much of style and grace to be hard to communicate. An explicit symbolic abstract analysis is an essential requirement, I believe for any learning process that can keep growing in power. See the discussions in chapters 1 and 8 of [4].

This line of thinking suggests that some of the laws of consciousness depend upon general principles of computational efficiency and are not entirely arbitrary features of the way the brain happens to be arranged. While interpretative programs are inherently slower than compiled programs, they are accompanied by a monologue in which the interpreter reports what rules it obeys and which data it uses. This would make it feasible for another concurrent process to supervise the activity, making logical and other inferences about how well the activity is serving a variety of goals. The compiled type of program does not lend itself to this, except for answering such questions as were anticipated before the program was compiled. Thus a compiled program, while more efficient, will seem less conscious; its user will be able to say relatively little to others or to himself about its operation.

The technical problems of making efficient compilers seem particularly serious for self-modifying programs; if these difficulties

persist, either for practical or for fundamental reasons, they might provide clues to the nature of those mental processes that most persistently figure in conscious mental activity. One would expect procedures that regularly change themselves to remain conscious.

Images

It is usually assumed that the abilities to visualize scenes, use geometric intuition, or reason using pictures and diagrams depend upon inherently non-verbal modes of thinking. Let us consider the hypothesis that imagery and verbal thinking are not really so different, and examine some computer programs that actually deal with geometric concepts.

The transformations of the retinal image of an object are complicated enough, when it suffers changes in position, illumination, and context. And to « parse » a visual scene into a cognitive structure of « perceived » physical objects and their interrelations is even harder when there are many objects, with some of them obscuring the view of others – a situation characteristic in real life. Surely the complexity of such an analysis is comparable to that needed for the syntactic analysis used in parsing speech at corresponding ages for children or, for that matter, at any age.

Consider the visual scene represented conventionally by the line drawing of figure 1.

The normal comprehension of this scene is to discern twelve objects, with the obvious support relations. How might we parse the scene into this structure? Assuming we have a way to reduce the retinal input signals to the simpler line-drawing form, we still need a system that can group the resulting collection of features – lines and vertices – into a physically plausible hypothesis about spatial objects.

A program developed in our laboratory by Adolfo Guzman solves this problem quite effectively. Its basic elements are the regions bounded by the lines; these are presumed to be projections of partly occluded faces of physical bodies. When two or more

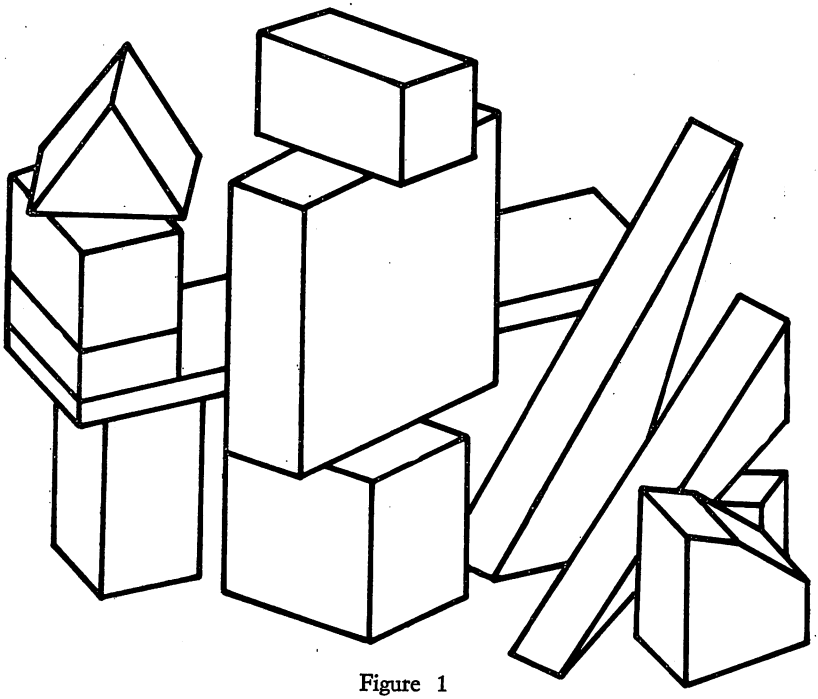


Figure 1

faces meet at a vertex, a number of abstract links are created to bind together some of those faces, according to the exact manner in which their boundary lines meet.

For example, in these three types of vertex configurations

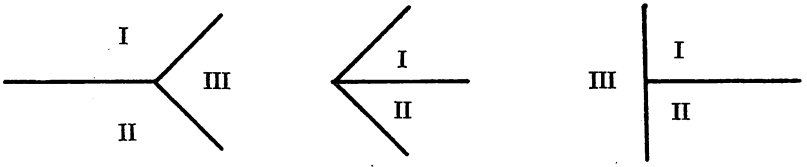
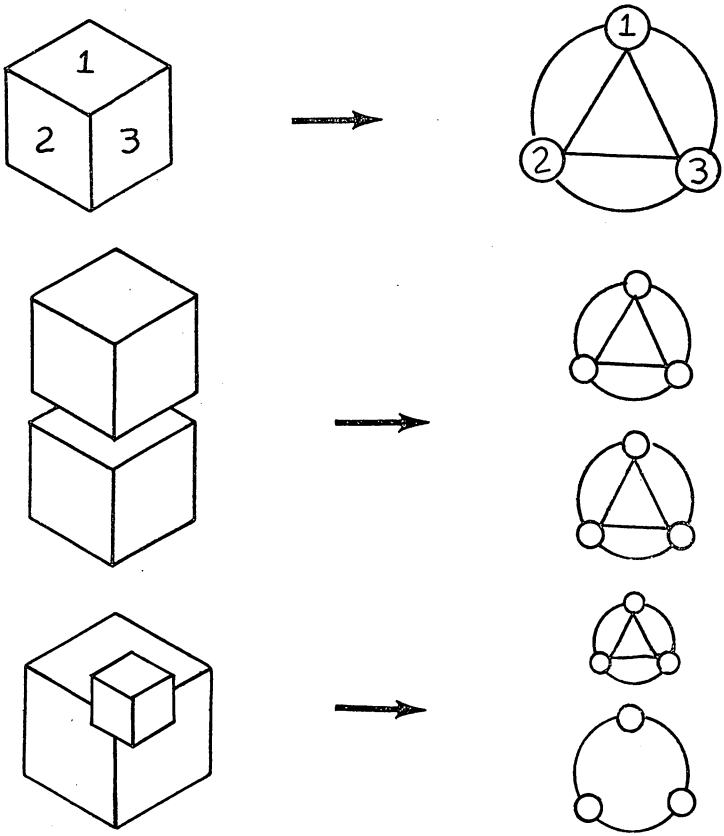


Figure 2

the « Y » provides evidence for linking region I to region II; II to III; and I to III. The « arrow » configuration just links I to II.

The idea is that a Y is probably (but not certainly) the image of a corner of an object that has all three faces on it; an arrow probably means that just the two faces are on the object whose corner is seen. But because a « T » is usually the result of one object covering up part of another it is not considered to establish a link between III and either of I and II; and there is no reason to suppose that the latter two should be linked together, either.

Using only these rules (and they are simplifications of the ones actually used) we can convert pictures into symbolically linked groupings of faces as follows: we represent the Y-links by straight lines and the arrow-links by curves:



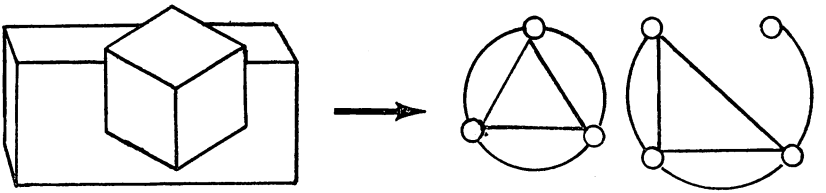


Figure 3

Up to this point we see no difficulty in associating the linked groups of faces to form the objects in the scene. In the next figure things are more complicated:

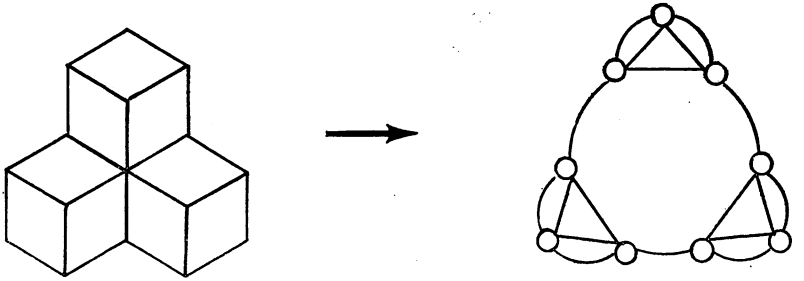


Figure 4

because of false « links » due to the exact superposition of boundaries of different objects. To break such misleading connections the program has a second phase that uses a hierarchical scheme: first it finds subsets of faces that are very tightly linked, e.g., by two or more links. These « nuclei » then compete for other faces. In Examples (1) - (4) there is no such competition, but in (5) the single false links between the cubes are broken by this procedure. We have described only the bare skeleton of the system; it actually uses a variety of other links, not all of the same strength. Particularly important is the effect of T-joints that line up with each other: in figure 5, I and II, and also III and IV, are linked together, even if there are other objects within the region V.

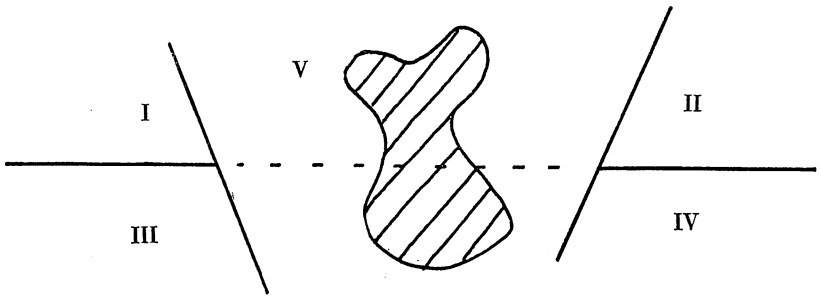


Figure 5

This allows the system to group together parts of an object completely disconnected by occlusions, as is the long board in our first illustration. The system compares quite favorably with human performance, it can be adapted to work with stereo picture-pairs, and it is quite suitably confused by appropriately ambiguous figures.

This program, described in more detail in [5], can be taken as a model of how a visual system might « parse » its input into a structure of mental objects and relations between those objects. While we have no reason to suppose that human vision works in just this sort of fashion, we know of no other theory of comparable economy of structure that can cope with anything approaching the same range of scenic complexity.

In any case, the program's analysis is unlike the usual « syntax-directed » parsing program in vogue among computer language designers, and is presumably different from the mechanisms most linguistic theorists would propose if they were to turn away from their curious preoccupation with ideal sentence-structure. But it does resemble in spirit, though not at all in detail, the kind of processing done by Quillian in his meaning-selection program [4]. Quillian constructs what he calls a « Semantic Network », in computer memory, linking in various ways different « meanings » of each of a number of words. The word « Cry », for example, could be linked to structures representing, among other things: making a sad sound, the call of an animal, or a proclamation. « Comfort »

might have meanings for: something someone needs, making someone less sad, etc. When the program is given a pair of words like « Cry; Comfort » – not a sentence! – it attempts to select the most plausible of each word's alternative meanings in the context of the other's. This is done by initiating two spreading foci of activity, one for each of the input words. The activities spread along the links between meanings, as brush-fires; eventually the two will intersect. The program notes where this first happens, and traces the activity back from this intersection to the original words, determining for each which of the alternate meanings was most directly involved. (For « Cry; Comfort », the program selected meanings for lamenting and consoling, through the word « sad »). A selector of this sort could guide a « meaning-directed » parsing system so as to escape the demonstrably desperate problems that have engulfed the analysis-through-synthesis, generative-grammar, parsing schemes. No one has yet attempted this, yet, but the success of Guzman's visual nucleus scheme favors further explorations of symbolic network manipulators.

Another extremely successful symbolic « image-processing » computer program is the Geometric Analogy program of Evans, also described in [4]. Evans, too, represents geometric scenes by descriptive expressions involving symbols for primitive objects and relations between them; he goes on to represent comparisons of pairs of scenes by expressions that describe almost-linguistic transformations. These transformations describe what must be done to convert one description into another; for example, by deleting one object and removing another from the interior of a third. I cannot summarize this whole system here, but it can be said that it compares favorably with older children's performance on the analogy type of intelligence test – an area that most people would agree involves a large degree of imagery.

REFERENCES

[1] E. A. FEIGENBAUM and J. FELDMAN (Eds.), *Computers and Thought*, New York, McGraw-Hill, 1963.

[2] M. MINSKY and S. PAPERT, *Perceptrons*, Cambridge, Mass., MIT Press, 1969.

[3] R. D. GREENBLATT, D. E. EASTLAKE and S. D. CROCKER, « The Greenblatt Chess Program », *Proc. Fall Joint Computer Conf.*, 1967, pp. 901-910, Thompson Books, Washington D. C.

[4] M. MINSKY (Ed.), *Semantic Information Processing*, Cambridge, Mass., MIT Press, 1968.

[5] A. GUZMAN, « Decomposition of a visual scene into bodies », *Proc. Fall Joint Computer Conf.*, 1968.