SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AI-TR-310
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): New Progress in Artificial Intelligence

5. TYPE OF REPORT & PERIOD COVERED: Annual Progress Report January 1972 - June 1974

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Patrick H. Winston

8. CONTRACT OR GRANT NUMBER(s): N00014-70-A-0003, N00014-70-A-0005, EC1049, EC40708X

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
The Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS:
Advanced Research Projects Agency
1400 Wilson Blvd.
Arlington, Virginia 22209

12. REPORT DATE: January 1972 - June 1974

13. NUMBER OF PAGES: 350

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
Office of Naval Research
Information Systems
Arlington, Virginia 22217

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report):
Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):

18. SUPPLEMENTARY NOTES:
None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial Intelligence	Heterarchy	Problem Solving
Assimilating Knowledge	Image Analysis	Productivity Technology
Automatic Debugging	Learning	Representation
CONNIVER	Machine Vision	Scene Analysis
Frame Systems	Manipulators	Understanding English

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report concentrates on progress during the last two years at the M.I.T. Artificial Intelligence Laboratory. Topics covered include the representation of knowledge, understanding English, learning and debugging, understanding vision and productivity technology. It is stressed that these various areas are tied closely together through certain fundamental issues and problems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

NEW PROGRESS IN ARTIFICIAL INTELLIGENCE

BY PATRICK H. WINSTON

AND THE STAFF OF

THE ARTIFICIAL INTELLIGENCE LABORATORY

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PATRICK H. WINSTON

ACTING DIRECTOR

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research was provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contracts N00014-70-A-0362-0003 and N00014-70-A-0362-0005. Support for the laboratory's education research was provided in part by the National Science Foundation under grants EC1049 and EC40708X.

TABLE OF CONTENTS

1 PURPOSE AND STRUCTURE

In assembling this report on progress, several objectives were kept in mind:

1. To enumerate our main achievements in the two years since our last major report.
2. To characterize the current M.I.T. point of view with respect to methodology and areas to be explored.
3. To introduce advanced topics in Artificial Intelligence.

In working toward these objectives, we have edited, abridged, and assembled together certain of the laboratory's Technical Reports, Memoranda, and Working Papers authored by laboratory members whose names appear in the section headings. Some of the shorter pieces appear nearly in full so as to convey a feeling for the detail and precision required in implementing a system. Most of the major works have been cut down considerably so as to avoid the bulk that might discourage readers from reading the report straight through. This necessarily means that in many cases a section will describe what can be done but will not describe how. Excited readers should think of these as hors d'oeuvres to be followed by entrees accessible through the bibliography.

ORGANIZATION

The material is arranged in the following sections:

REPRESENTING KNOWLEDGE
UNDERSTANDING ENGLISH
LEARNING AND DEBUGGING
UNDERSTANDING VISION
PRODUCTIVITY TECHNOLOGY
EDUCATION
POTPOURRI

Other arrangements are possible of course. Many of the topics might equally well have been in a different section as the sections are far from mutually exclusive.

REFERENCES

References to laboratory publications are cited by the following code:
WP ==> Working Paper
AIM ==> Artificial Intelligence Memo
TR ==> Technical Report

PREPARATION

The report was prepared by Patrick H. Winston working with Suzin L. Jabari, Cheryl Goodman, and Eva I. Kampits. Suzin L. Jabari illustrated sections 2.3, 4.3, 5.1.1, 5.2.3, 6.2.2, 6.2.3, and 6.3.1. Allison Platt illustrated sections 2.1, 3.3, 4.3, 4.4, and 5.2.3. Cover design by Suzin L. Jabari.

2 REPRESENTING KNOWLEDGE

The problem of representation seems central to major artificial intelligence projects. It is therefore appropriate to begin this progress report by describing some pieces of work that bear directly on representation issues.

The first is Minsky's new theory of how to deal with the knowledge required in understanding the world of commonplace situations -- a visual scene, a children's birthday party, or whatever. Minsky proposes organization into chunks called frames and goes on to show how manipulation of those frames can do such things as effect changes in visual viewpoint or supply unstated, common-sense facts in story analysis.

MINSKY'S FRAME SYSTEM THEORY

Here is the essence of the frame theory: When one encounters a new situation (or makes a substantial change in one's view of a problem), one selects from memory a structure called a frame. This is a remembered framework to be adapted to fit reality by changing details as necessary.

A frame is a data-structure for representing a stereotyped situation like being in a certain kind of living room or going to a child's birthday party. Attached to each frame are several kinds of information. Some of this information is about how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed.

We can think of a frame as a network of nodes and relations. The "top levels" of a frame are fixed, and represent things that are always true about the supposed situation. The lower levels have many terminals -- "slots" that must be filled by specific instances or data. Each terminal can specify conditions its assignments must meet. (The assignments themselves are usually smaller "sub-frames.") Simple conditions are specified by markers that might require a terminal assignment to be a person, an object of sufficient value, or a pointer to a sub-frame of a certain type. More complex conditions can specify relations among the things assigned to several terminals.

Collections of related frames are linked together into frame-systems. The effects of important actions are mirrored by transformations between the frames of a system. These are used to make certain kinds of calculations economical, to represent changes of emphasis and attention, and to account for the effectiveness of "imagery."

For visual scene analysis, the different frames of a system describe the scene from different viewpoints, and the transformations between one frame and another represent

the effects of moving from place to place. For non-visual kinds of frames, the differences between the frames of a system can represent actions, cause-effect relations, or changes in conceptual viewpoint. Different frames of a system share the same terminals; this is the critical point that makes it possible to coordinate information gathered from different viewpoints.

Much of the phenomenological power of the theory hinges on the inclusion of expectations and other kinds of presumptions. A frame's terminals are normally already filled with "default" assignments. Thus, a frame may contain a great many details whose supposition is not specifically warranted by the situation. These have many uses in representing general information, most likely cases, techniques for by-passing "logic," and ways to make useful generalizations.

The default assignments are attached loosely to their terminals, so that they can be easily displaced by new items that fit better the current situation. They thus can serve also as "variables" or as special cases for "reasoning by example," or as "textbook cases," and often make the use of logical quantifiers unnecessary.

The frame-systems are linked, in turn, by an information retrieval network. When a proposed frame cannot be made to fit reality -- when we cannot find terminal assignments that suitably match its terminal marker conditions -- this network provides a replacement frame. These inter-frame structures make possible other ways to represent knowledge about facts, analogies, and other information useful in understanding.

Once a frame is proposed to represent a situation, a matching process tries to assign values to each frame's terminals, consistent with the markers at each place. The matching process is partly controlled by information associated with the frame (which includes information about how to deal with surprises) and partly by knowledge about the system's current goals. There are important uses for the information, obtained when a matching process fails; it can be used to select an alternative frame that better suits the situation.

LOCAL AND GLOBAL THEORIES FOR VISION

When we enter a room we seem to see the entire scene at a glance. But seeing is really an extended process. It takes time to fill in details, collect evidence, make conjectures, test, deduce, and interpret in ways that depend on our knowledge, expectations and goals. Wrong first impressions have to be revised. Nevertheless, all

this proceeds so quickly and smoothly that it seems to demand a special explanation.

Would parallel processing help? This is a more technical question than it might seem. At the level of detecting elementary visual features, texture elements, stereoscopic and motion-parallax cues, it is obvious that parallel processing might be useful. At the level of grouping features into objects, it is harder to see exactly how to use parallelism, but one can at least conceive of the aggregation of connected "nuclei" (Guzman TR-228), or the application of boundary line constraint semantics (Waltz TR-271), performed in a special parallel network.

At "higher" levels of cognitive processing, however, one suspects fundamental limitations in the usefulness of parallelism. Many "integral" schemes were proposed in the literature on "pattern recognition" for parallel operations on pictorial material -- perceptrons, integral transforms, skeletonizers, and so forth. These mathematically and computationally interesting schemes might quite possibly serve as ingredients of perceptual processing theories. But as ingredients only! Basically, "integral" methods work only on isolated figures in two dimensions. They fail disastrously in coping with complicated, three-dimensional scenery.

The new, more successful symbolic theories use hypothesis formation and confirmation methods that seem, on the surface at least, more inherently serial. It is hard to solve any very complicated problem without giving essentially full attention, at different times, to different sub-problems. Fortunately, however, beyond the brute idea of doing many things in parallel, one can imagine a more serial process that deals with large, complex, symbolic structures as units! This opens a new theoretical "niche" for performing a rapid selection of large substructures; in this niche our theory hopes to find the secret of speed, both in vision and in ordinary thinking.

SEEING A CUBE

In the tradition of Guzman and Winston, we assume that the result of looking at a cube is a structure something like that in figure 1. The substructures "A" and "B" represent details or decorations on two faces of the cube. When we move to the right, face "A" disappears from view, while the new face decorated with "C" is now seen. If we had to analyse the scene from the start, we would have to

(1) lose the knowledge about "A,"
(2) recompute "B," and
(3) compute the description of "C."

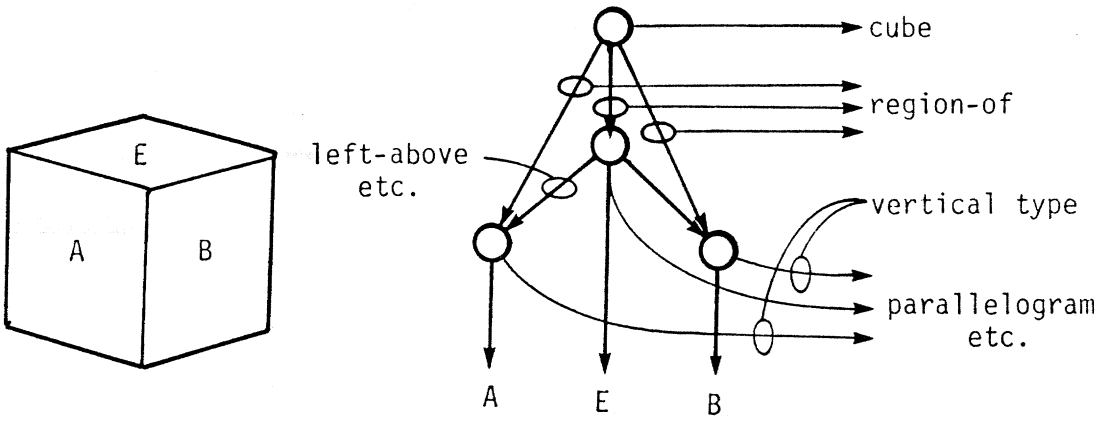


FIGURE 1

But since we know we moved to the right, we can save "B" by assigning it also to the "left face" terminal of a second cube-frame. To save "A" (just in case!) we connect it also to an extra, invisible face-terminal of the new cube-schema as in figure 2.

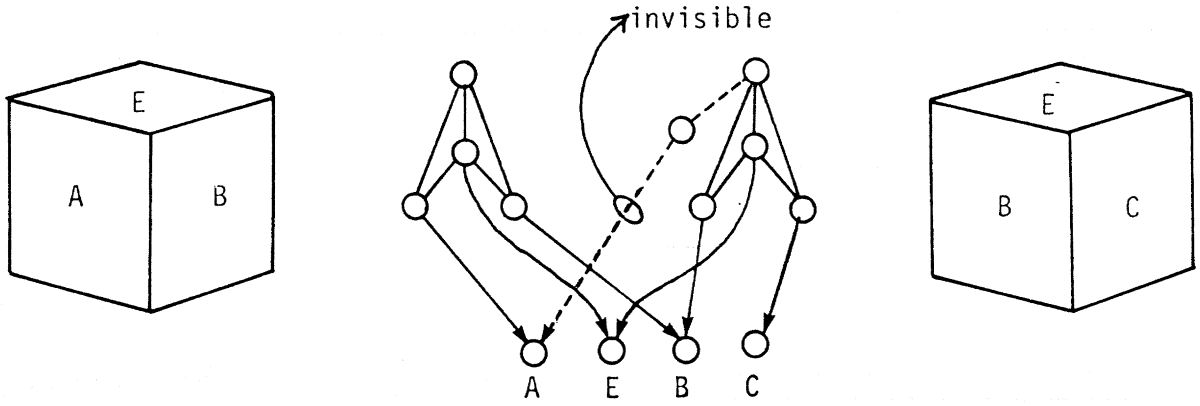
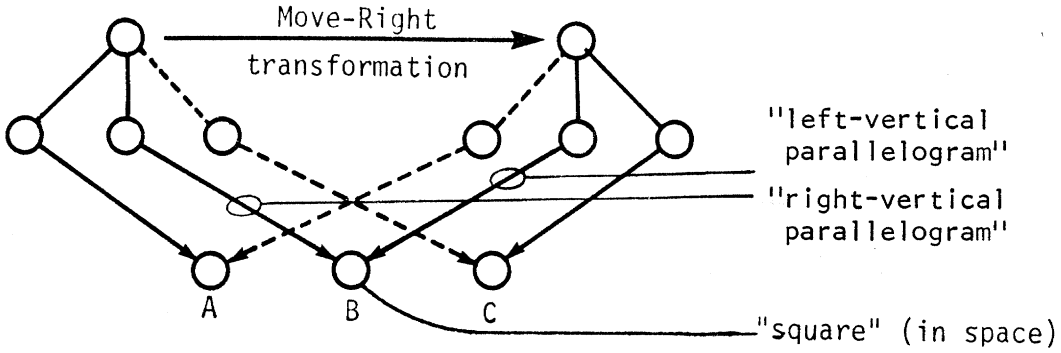


FIGURE 2

If later we move back to the left, we can reconstruct the first scene without any perceptual computation at all: just restore the top-level pointers to the first cube-

frame. We now need a place to store "C"; we can add yet another invisible face to the right in the first cube-frame! See figure 3. We could extend this to represent further



excursions around the object. This would lead to a more comprehensive frame system, in which each frame represents a different "perspective" of a cube. In figure 4 there are three frames corresponding to 45-degree MOVE-RIGHT and MOVE-LEFT actions. If we pursue this analysis, the resulting system can become very large; more complex objects need even more different projections. It is not obvious either that all of them are normally necessary or that just one of each variety is adequate. It all depends.

It is not proposed that this kind of complicated structure is recreated every time one examines an object. It is imagined instead that a great collection of frame systems is stored in permanent memory, and one of them is evoked when evidence and expectation make it plausible that the scene in view will fit it. How are they acquired? We propose that if a chosen frame does not fit well enough, and if no better one is easily found, and if the matter is important enough, then an adaptation of the best one so far discovered will be constructed and remembered for future use.

Each frame has terminals for attaching pointers to substructures. Different frames can share the same terminal, which can thus correspond to the same physical feature as seen in different views. This permits us to represent, in a single place, view-independent information gathered at different times and places. This is important also in non-visual applications.

The matching process which decides whether a proposed frame is suitable is

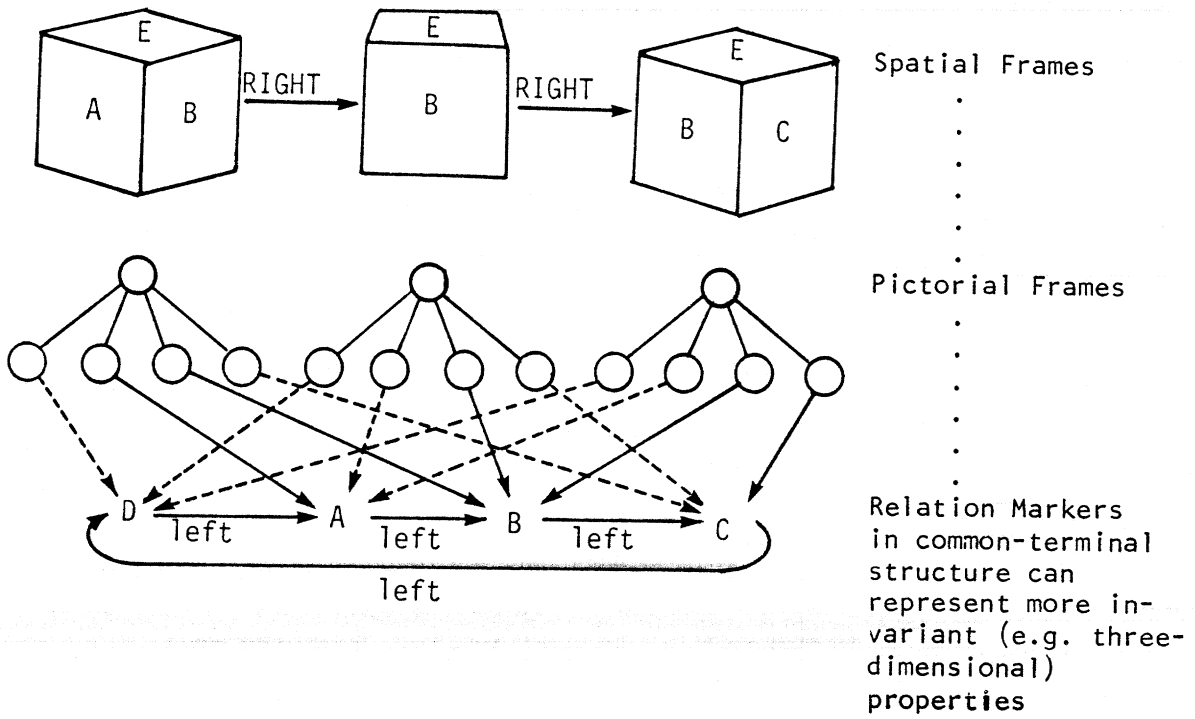


FIGURE 4

controlled partly by one's current goals and partly by information attached to the frame; the frames carry terminal markers and other constraints, while the goals are used to decide which of these constraints are currently relevant. Generally, the matching process could have these components:

(1) A frame, once evoked on the basis of partial evidence or expectation, would first direct a test to confirm its own appropriateness, using knowledge about recently noticed features, loci, relations, and plausible Sub-frames. The current goal list is used to decide which terminals and conditions must be made to match reality.

(2) Next it would request information needed to assign values to those terminals that cannot retain their default assignments. For example, it might request a description of face "C," if this terminal is currently unassigned, but only if it is not marked

"invisible." Such assignments must agree with the current markers at the terminal. Thus, face "C" might already have markers for such constraints or expectations as:

* Right-middle visual field.
* Must be assigned.
* Should be visible; if not, consider moving right.
* Should be a cube-face sub-frame.
* Share left vertical boundary terminal with face "B."
* If failure, consider box-lying-on-side frame.
* Same background color as face "B."

(3) Finally, if informed about a transformation (e.g., an impending motion) it would transfer control to the appropriate other frame of that system.

Within the details of the control scheme are opportunities to embed many kinds of knowledge. When a terminal-assigning attempt fails, the resulting error message can be used to propose a second-guess alternative. Later it is shown how memory can be organized into a "Similarity Network" as proposed in Winston's thesis (TR-231).

IS VISION SYMBOLIC?

Can one really believe that a person's appreciation of three-dimensional structure can be so fragmentary and atomic as to be representable in terms of the relations between parts of two-dimensional views? Let us separate, at once, the two issues: is imagery symbolic? and is it based on two-dimensional fragments? The first problem is one of degree; surely everyone would agree that at some level vision is essentially symbolic. The quarrel would be between certain naive conceptions on one side -- in which one accepts seeing either as picture-like or as evoking imaginary solids -- against the confrontation of such experimental results of Piaget (1956) and others in which many limitations that one might fear would result from symbolic representations are shown actually to exist!

As for our second question: the issue of two- vs. three-dimensions evaporates at the symbolic level. The very concept of dimension becomes inappropriate. Each type of symbolic representation of an object serves some goals well and others poorly. If we attach the relation labels left-of, right-of, and above between parts of the structure, say, as markers on pairs of terminals, certain manipulations will work out smoothly; for

example, some properties of these relations are "invariant" if we rotate the cube while keeping the same face on the table. Most objects have "permanent" tops and bottoms. But if we turn the cube on its side such predictions become harder to make; people have great difficulty keeping track of the faces of a six-colored cube if one makes them roll it around in their mind.

If one uses instead more "intrinsic" relations like next-to and opposite-to, then turning the object on its side disturbs the "image" much less. In Winston's thesis we see how systematic replacements (e.g., of "left" for "behind," and "right" for "in-front-of") can deal with the effect of spatial rotation.

SEEING A ROOM

Visual experience seems continuous. One reason is that we move continuously. A deeper explanation is that our "expectations" usually interact smoothly with our perceptions. Suppose you were to leave a room, close the door, turn to reopen it, and find an entirely different room. You would be shocked. The sense of change would be hardly less striking if the world suddenly changed before your eyes. A naive theory of phenomenological continuity is that we see so quickly that our image changes as fast as does the scene. There is an alternative theory: the changes in one's frame-structure representation proceed at their own pace; the system prefers to make small changes whenever possible; and the illusion of continuity is due to the persistence of assignments to terminals common to the different view-frames. Thus, continuity depends on the confirmation of expectations which in turn depends on rapid access to remembered knowledge about the visual world.

Just before you enter a room, you usually know enough to "expect" a room rather than, say, a landscape. You can usually tell just by the character of the door. And you can often select in advance a frame for the new room. Very often, one expects a certain particular room. Then many assignments are already filled in.

The simplest sort of room-frame candidate is like the inside of a box. Following our cube-model, the room-frame might have the top-level structure shown in figure 5.

One has to assign to the frame's terminals the things that are seen. If the room is familiar, some are already assigned. If no expectations are recorded already, the first priority might be locating the principal geometric landmarks. To fill in LEFT WALL one might first try to find edges "a" and "d" and then the associated corners "ag" and "gd." Edge "g," for example, is usually easy to find because it should intersect any eye-level horizontal scan from left to right. Eventually, "ag," "gb," and "ba" must not be too

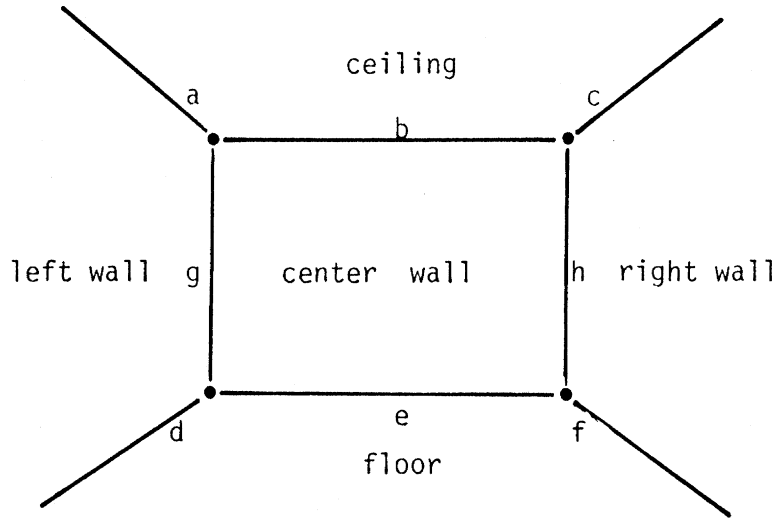


FIGURE 5

inconsistent with one another -- because they are the same physical vertex.

However the process is directed, there are some generally useful knowledge-based tactics. It is probably easier to find edge "e" than any other edge, because if we have just entered a normal rectangular room, then we may expect that

* Edge "e" is a horizontal line.
* It is below eye level.
* It defines a floor-wall texture boundary.

Given an expectation about the size of a room, we can estimate the elevation of "e," and vice versa. In outdoor scenes, "e" is the horizon and on flat ground we can expect to see it at eye-level. If we fail quickly to locate and assign this horizon, we must consider rejecting the proposed frame: either the room is not normal or there is a large obstruction.

The room-analysis strategy might try next to establish some other landmarks. Given "e," we next look for its left and right corners, and then for the verticals rising from

them. Once such gross geometrical landmarks are located, we can guess the room's general shape and size. This might lead to selecting a new frame better matched to that shape and size, with additional markers confirming the choice and completing the structure with further details.

SCENE ANALYSIS AND SUBFRAMES

If the new room is unfamiliar, no pre-assembled frame can supply fine details; more scene-analysis is needed. Even so, the complexity of the work can be reduced, given suitable subframes for constructing hypotheses about substructures in the scene. How useful these will be depends both on their inherent adequacy and on the quality of the expectation process that selects which one to use next. One can say a lot even about an unfamiliar room. Most rooms are like boxes, and they can be categorized into types: kitchen, hall, living room, theater, and so on. One knows dozens of kinds of rooms and hundreds of particular rooms; one no doubt has them structured into some sort of similarity network for effective access. This will be discussed later.

A typical room-frame has three or four visible walls, each perhaps of a different "kind." One knows many kinds of walls: walls with windows, shelves, pictures, and fireplaces. Each kind of room has its own kinds of walls. A typical wall might have a 3 x 3 array of region-terminals (left-center-right) x (top-middle-bottom) so that wall-objects can be assigned qualitative locations. One would further want to locate objects relative to geometric inter-relations in order to represent such facts as "Y is a little above the center of the line between X and Z."

In three dimensions, the location of a visual feature of a subframe is ambiguous, given only eye direction. A feature in the middle of the visual field could belong either to a Center Front Wall object or to a High Middle Floor object; these attach to different subframes. The decision could depend on reasoned evidence for support, on more directly visual distance information derived from stereo disparity or motion-parallax, or on plausibility information derived from other frames: a clock would be plausible only on the wall-frame while a person is almost certainly standing on the floor.

Given a box-shaped room, lateral motions induce orderly changes in the quadrilateral shapes of the walls as in figure 6. A picture-frame rectangle, lying flat against a wall, should transform in the same way as does its wall. If a "center-rectangle" is drawn on a left wall it will <u>appear</u> to project out because one makes the default assumption that any such quadrilateral is actually a rectangle hence must lie in a plane that would so project. In figure 7A, both quadrilaterals could "look like" rectangles, but the one to the right

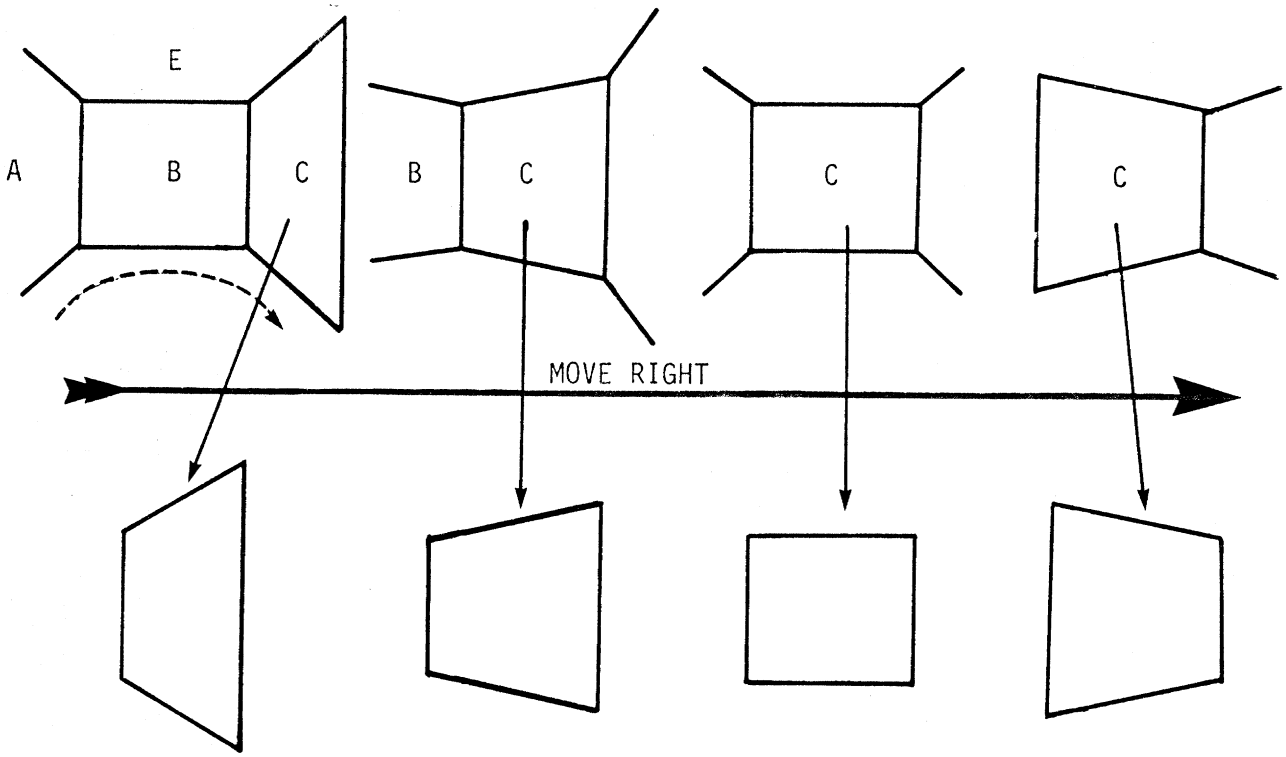


FIGURE 6

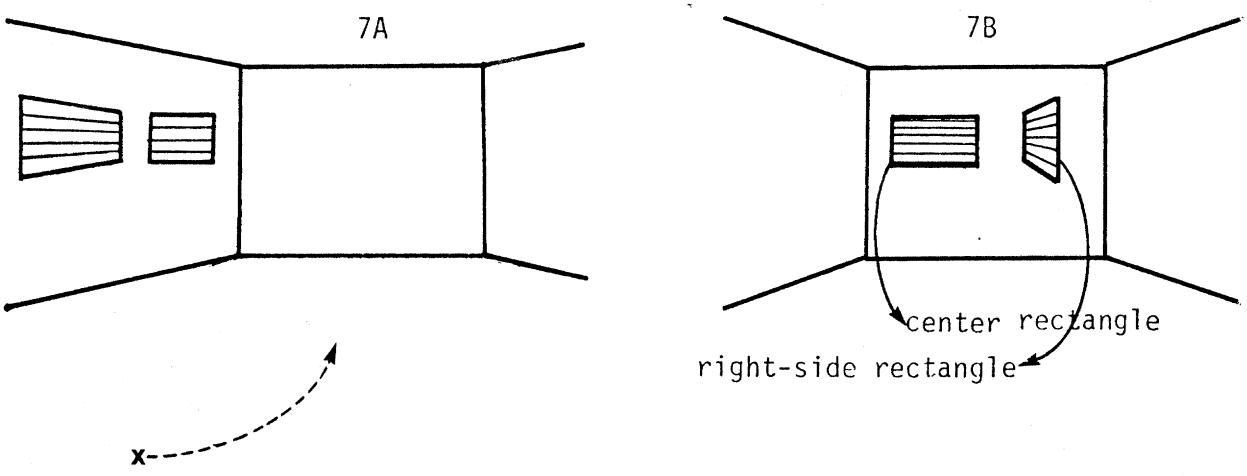
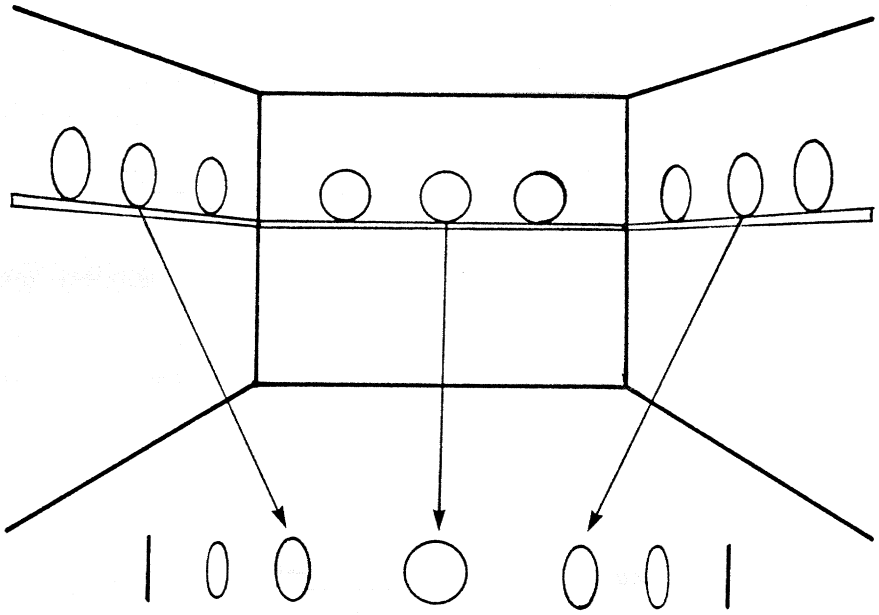


FIGURE 7

does not match the markers for a "left rectangle" subframe (these require, e.g., that the left side be longer than the right side). That rectangle is therefore represented by a center-rectangle frame, and seems to project out as though parallel to the center wall.

Thus we must not simply assign the label "rectangle" to a quadrilateral but to a particular frame of a rectangle-system. When we move, we expect whatever space-transformation is applied to the top-level system will be applied also to its subsystems as suggested in figure 7B.

Similarly the sequence of elliptical projections of a circle contains congruent pairs that are visually ambiguous as shown in figure 8. But because wall objects usually lie



Two figures are congruent, but transform differently.

FIGURE 8

flat, we assume that an ellipse on a left wall is a left-ellipse, expect it to transform the same way as the left wall, and are surprised if the prediction is not confirmed.

DEFAULT ASSIGNMENT

While both Seeing and Imagining result in assignments to frame terminals, Imagination leaves us wider choices of detail and variety of such assignments. Frames are probably never stored in long-term memory with unassigned terminal values. Instead, what really happens is that frames are stored with weakly-bound default assignments at every terminal! These manifest themselves as often-useful but sometimes counter-productive stereotypes.

Thus in the sentence "John kicked the ball," you probably cannot think of a purely abstract ball, but must imagine characteristics of a vaguely particular ball; it probably has a certain default size, default color, default weight. Perhaps it is a descendant of one you first owned or were injured by. Perhaps it resembles your latest one. In any case your image lacks the sharpness of presence because the processes that inspect and operate upon the weakly-bound default features are very likely to change, adapt, or detach them.

WORDS, SENTENCES AND MEANINGS

The concepts of frame and default assignment seem helpful in discussing the phenomenology of "meaning." Chomsky (1957) points out that such a sentence as

(A) "colorless green ideas sleep furiously"

is treated very differently than the non-sentence

(B) "furiously sleep ideas green colorless"

and suggests that because both are "equally nonsensical," what is involved in the recognition of sentences must be quite different from what is involved in the appreciation of meanings.

There is no doubt that there are processes especially concerned with grammar. Since the meaning of an utterance is "encoded" as much in the positional and structural relations between the words as in the word choices themselves, there must be processes concerned with analysing those relations in the course of building the structures that will more directly represent the meaning. What makes the words of (A) more effective and predictable than (B) in producing such a structure -- putting aside

the question of whether that structure should be called semantic or syntactic -- is that the word-order relations in (A) exploit the (grammatical) convention and rules people usually use to induce others to make assignments to terminals of structures. This is entirely consistent with grammar theories. A generative grammar would be a summary description of the exterior appearance of those frame rules -- or their associated processes -- while the operators of transformational grammars seem similar enough to some of our frame transformations.

We certainly cannot assume that "logical" meaninglessness has a precise psychological counterpart. Sentence (A) can certainly generate an image! The dominant frame is perhaps that of someone sleeping; the default system assigns a particular bed, and in it lies a mummy-like shape-frame with a translucent green color property. In this frame there is a terminal for the character of the sleep -- restless, perhaps -- and "furiously" seems somewhat inappropriate at that terminal, perhaps because the terminal does not like to accept anything so "intentional" for a sleeper. "Idea" is even more disturbing, because one expects a person, or at least something animate. One senses frustrated procedures trying to resolve these tensions and conflicts more properly, here or there, into the sleeping framework that has been evoked.

Utterance (B) does not get nearly so far because no subframe accepts any substantial fragment. As a result no larger frame finds anything to match its terminals, hence finally, no top level "meaning" or "sentence" frame can organize the utterance as either meaningful or grammatical. By combining this "soft" theory with gradations of assignment tolerances, one could develop systems that degrade properly for sentences with "poor" grammar rather than none; if the smaller fragments -- phrases and sub-clauses -- satisfy subframes well enough, an image adequate for certain kinds of comprehension could be constructed anyway, even though some parts of the top level structure are not entirely satisfied. Thus, we arrive at a qualitative theory of "grammatical:" if the top levels are satisfied but some lower terminals are not we have a meaningless sentence; if the top is weak but the bottom solid, we can have an ungrammatical but meaningful utterance.

DISCOURSE

Linguistic activity involves larger structures than can be described in terms of sentential grammar, and these larger structures further blur the distinctness of the syntax-semantic dichotomy. Consider the following fable, as told by W. Chafe (1972):

> There was once a Wolf who saw a Lamb drinking at a river and wanted an excuse to eat it. For that purpose, even though he himself was upstream, he accused the Lamb of stirring up the water and keeping him from drinking...

To understand this, one must realize that the Wolf is lying! To understand the key conjunctive "even though" one must realize that contamination never flows upstream. This in turn requires us to understand (among other things) the word "upstream" itself. Within a declarative, predicate-based "logical" system, one might try to formalize "upstream" by some formula like:

[A upstream B] AND [Event T, Stream muddy at A] ===>
[Exists [Event U, Stream muddy at B]] AND [Later U, T]

But an adequate definition would need a good deal more. What about the fact that the order of things being transported by water currents is not ordinarily changed? A logician might try to deduce this from a suitably intricate set of "local" axioms, together with appropriate "induction" axioms. I propose instead to represent this knowledge in a structure that automatically translocates spatial descriptions from the terminals of one frame to those of another frame of the same system. While this might be considered to be a form of logic, it uses some of the same mechanisms designed for spatial thinking.

In many instances we would handle a change over time, or a cause-effect relation, in the same way as we deal with a change in position. Thus, the concept river-flow could evoke a frame-system structure something like the following, where S1, S2, and S3 are abstract slices of the flowing river shown in figure 9.

There are many more nuances to fill in. What is "stirring up" and why would it keep the wolf from drinking? One might normally assign default floating objects to the S's, but here S3 interacts with "stirring up" to yield something that "drink" does not find acceptable. Was it "deduced" that stirring river-water means that S3 in the first frame should have "mud" assigned to it; or is this simply the default assignment for stirred water?

Almost any event, action, change, flow of material, or even flow of information can be represented to a first approximation by a two-frame generalized event. The frame-system can have slots for agents, tools, side-effects, preconditions, generalized trajectories, just as in the "trans" verbs of "case grammar" theories, but we have the additional flexibility of representing changes explicitly. To see if one has understood an event or action, one can try to build an appropriate instantiated frame-pair.

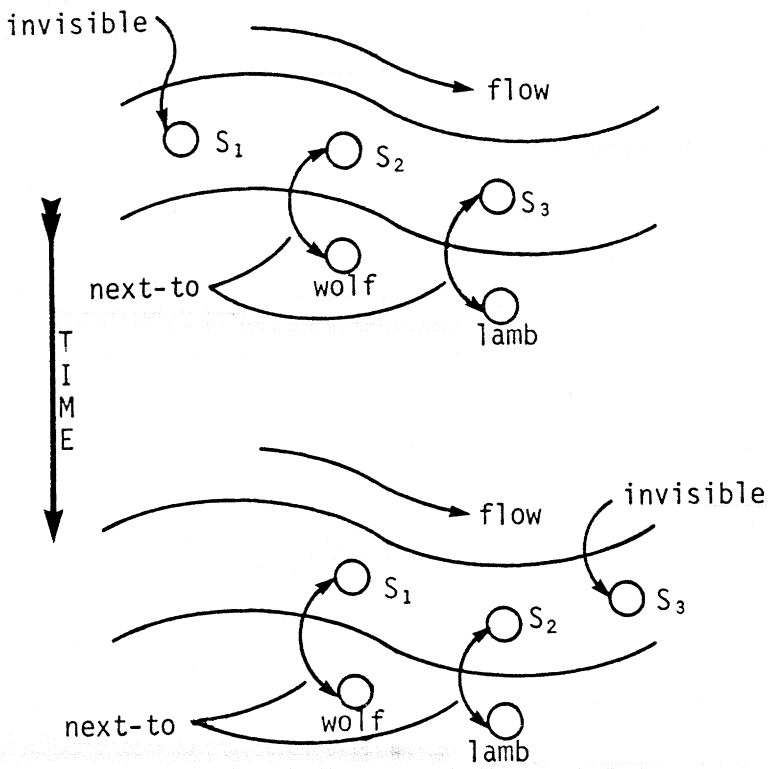


FIGURE 9

However, in representing changes by simple "before-after" frame-pairs, we can expect to pay a price. Pointing to a pair is not the same as describing their differences. This makes it less convenient to do planning or abstract reasoning; there is no explicit place to attach information about the transformation. As a second approximation, we could label pairs of nodes that point to corresponding terminals, obtaining a structure like the "comparison-notes" in Winston (TR-231), or we might place at the top of the frame-system information describing the differences more abstractly. Something of this sort will be needed eventually.

SCENARIOS

We condense and conventionalize, in language and thought, complex situations and sequences into compact words and symbols. Some words can perhaps be "defined" in

elegant, simple structures, but only a small part of the meaning of "trade" is captured by:

first frame			second frame	
		--->		
A has X	B has Y		B has X	A has Y

Trading normally occurs in a social context of law, trust and convention. Unless we also represent these other facts, most trade transactions will be almost meaningless. It is usually essential to know that each party usually wants both things but has to compromise. It is a happy but unusual circumstance in which each trader is glad to get rid of what he has. To represent trading strategies, one could insert the basic maneuvers right into the above frame-pair scenario: in order for A to make B want X more (or want Y less) we expect him to select one of the familiar tactics:

Offer more for Y.
Explain why X is so good.
Create favorable side-effect of B having X.
Disparage the competition.
Make B think C wants X.

These only scratch the surface. Trades usually occur within a scenario tied together by more than a simple chain of events each linked to the next. No single such scenario will do; when a clue about trading appears it is essential to guess which of the different available scenarios is most likely to be useful.

Charniak's thesis (TR-266) studies questions about transactions that seem easy for people to comprehend yet obviously need rich default structures. We find in elementary school reading books such stories as:

Jane was invited to Jack's Birthday Party.
She wondered if he would like a kite.
She went to her room and shook her piggy bank.
It made no sound.

We first hear that Jane is invited to Jack's Birthday Party. Without the party scenario, or at least an invitation scenario, the second line seems rather mysterious:

She wondered if he would like a kite.

To explain one's rapid comprehension of this, we make a somewhat radical proposal: to represent explicitly, in the frame for a scenario structure, pointers to a collection of the most serious problems and questions commonly associated with it. In fact we shall consider the idea that the frame terminals are exactly those questions.

Thus, for the birthday party:

Y must get P for X -------- Choose P!
X must like P ------------ Will X like P?
Buy P ---------- Where to buy P?
Get money to buy P ---- Where to get money?
(Sub-questions of the "present" frame?)
Y must dress up ---------- What should Y wear?

Certainly these are one's first concerns when one is invited to a party.

The reader is free to wonder whether this solution is acceptable. The question "Will X like P?" certainly matches "She wondered if he would like a kite?" and correctly assigns the kite to P. But is our world regular enough that such question sets could be pre-compiled to make this mechanism often work smoothly? The answer is mixed. We do indeed expect many such questions; we surely do not expect all of them. But surely "expertise" consists partly in not having to realize, ab initio, what are the outstanding problems and interactions in situations. Notice, for example, that there is no default assignment for the Present in our party-scenario frame. This mandates attention to that assignment problem and prepares us for a possible thematic concern. In any case, we probably need a more active mechanism for understanding "wondered" which can apply the information currently in the frame to produce an expectation of what Jane will think about. The key words and ideas of a discourse evoke substantial thematic or scenario structures, drawn from memory with rich default assumptions.

In any event, the individual statements of a discourse lead to temporary representations -- which seem to correspond to what contemporary linguists call "deep structures" -- which are then quickly rearranged or consumed in elaborating the growing scenario representation. In order of "scale," among the ingredients of such a structure there might be these kinds of levels:

Surface Syntactic Frames --- Mainly verb and noun structures.
Prepositional and word-order indicator conventions.

Surface Semantic Frames ---Action-centered meanings of words.
Qualifiers and relations concerning participants, instruments, trajectories and strategies, goals, consequences and side-effects.

Thematic Frames --- Scenarios concerned with topics, activities, portraits, setting.
Outstanding problems and strategies commonly connected with topics.

Narrative Frames --- Skeleton forms for typical stories, explanations, and arguments.
Conventions about foci, protagonists, plot forms, development, etc., designed to help a listener construct a new, instantiated Thematic Frame in his own mind.

REQUESTS TO MEMORY

We can now imagine the memory system as driven by two complementary needs. On one side are items demanding to be properly represented by being embedded into larger frames; on the other side are incompletely-filled frames demanding terminal assignments. The rest of the system will try to placate these lobbyists, but not so much in accord with "general principles" as in accord with special knowledge and conditions imposed by the currently active goals.

When a frame encounters trouble -- when an important condition cannot be satisfied -- something must be done. We envision the following major kinds of accomodation to trouble.

MATCHING: When nothing more specific is found, we can attempt to use some "basic" associative memory mechanism. This will succeed by itself only in relatively simple situations, but should play a supporting role in the other tactics.

EXCUSE: An apparent misfit can often be excused or explained. A "chair" that meets all other conditions but is much too small could be a "toy."

ADVICE: The frame contains explicit knowledge about what to do about the trouble. Below, we describe an extensive, learned "Similarity Network" in which to embed such

knowledge.

SUMMARY: If a frame cannot be completed or replaced, one must give it up. But first one must construct a well-formulated complaint or summary to help whatever process next becomes responsible for reassigning the subframes left in limbo.

MATCHING

When replacing a frame, we do not want to start all over again. How can we remember what was already "seen?" We consider here only the case in which the system has no specific knowledge about what to do and must resort to some "general" strategy. No completely general method can be very good, but if we could find a new frame that shares enough terminals with the old frame, then some of the common assignments can be retained, and we will probably do better than chance.

The problem can be formulated as follows: let E be the cost of losing a certain already assigned terminal and let F be the cost of being unable to assign some other terminal. If E is worse than F, then any new frame should retain the old subframe. Thus, given any sort of priority ordering on the terminals, a typical request for a new frame should include:

(1) Find a frame with as many terminals in common with [a,b,...,z] as possible, where we list high priority terminals already assigned in the old frame.

But the frame being replaced is usually already a subframe of some other frame and must satisfy the markers of its attachment terminal, lest the entire structure be lost. This suggests another form of memory request, looking upward rather than downward:

(2) Find or build a frame that has properties [a,b,...,z]

If we emphasize differences rather than absolute specifications, we can merge (2) and (1):

(3) Find a frame that is like the old frame except for certain differences [a,b,...,z] between them.

One can imagine a parallel-search or hash-coded memory to handle (1) and (2) if the terminals or properties are simple atomic symbols. (There must be some such mechanism, in any case, to support a production-based program or some sort of pattern matcher.) Unfortunately, there are so many ways to do this that it implies no specific design requirements.

Although (1) and (2) are formally special cases of (3), they are different in practice because complicated cases of (3) require knowledge about differences. In fact (3) is too general to be useful as stated, and we will later propose to depend on specific, learned, knowledge about differences between pairs of frames rather than on broad, general principles.

It should be emphasized again that we must not expect magic. For difficult, novel problems a new representation structure will have to be constructed, and this will require application of both general and special knowledge.

EXCUSES

We can think of a frame as describing an "ideal." If an ideal does not match reality because it is "basically" wrong, it must be replaced. But it is in the nature of ideals that they are really elegant simplifications; their attractiveness derives from their simplicity, but their real power depends upon additional knowledge about interactions between them! Accordingly we need not abandon an ideal because of a failure to instantiate it, provided one can explain the discrepancy in terms of such an interaction. Here are some examples in which such an "excuse" can save a failing match:

OCCLUSION: A table, in a certain view, should have four legs, but a chair might occlude one of them. One can look for things like T-joints and shadows to support such an excuse.

FUNCTIONAL VARIANT: A chair-leg is usually a stick, geometrically; but more important, it is <u>functionally</u> a support. Therefore, a strong center post, with an adequate base plate, should be an acceptable replacement for all the legs. Many objects are multiple purpose and need functional rather than physical descriptions.

BROKEN: A visually missing component could be explained as in fact physically missing, or it could be broken. Reality has a variety of ways to frustrate ideals.

PARASITIC CONTEXTS: An object that is just like a chair, except in size, could be (and probably is) a toy chair. The complaint "too small" could often be so interpreted in contexts with other things too small, children playing, peculiarly large "grain," and so forth.

In most of those examples, the kinds of knowledge to make the repair -- and thus salvage the current frame -- are "general" enough usually to be attached to the thematic context of a superior frame.

ADVICE AND SIMILARITY NETWORKS

In moving about a familiar house, we already know a dependable structure for "information retrieval" of room frames. When we move through Door D, in Room X, we expect to enter Room Y (assuming D is not the Exit). We could represent this as an action transformation of the simplest kind, consisting of pointers between pairs of room frames of a particular house system.

When the house is not familiar, a "logical" strategy might be to move up a level of classification: when you leave one room, you may not know which room you are entering, but you usually know that it is some room. Thus, one can partially evade lack of specific information by dealing with classes -- and one has to use some form of abstraction or generalization to escape the dilemma of Bartlett's commander.

Winston's thesis (TR-231) proposes a way to construct a retrieval system that can represent classes but has additional flexibility. His retrieval pointers can be made to represent goal requirements and action effects as well as class memberships.

> What does it mean to expect a chair? Typically, four legs, some assortment of rungs, a level seat, an upper back. One expects also certain relations between these "parts." The legs must be below the seat, the back above. The legs must be supported by the floor. The seat must be horizontal, the back vertical, and so forth.
>
> Now suppose that this description does not match; the vision system finds four legs, a level plane, but no back. The "difference" between what we expect and what we see is "too few backs." This suggests not a chair, but a table or a bench.

Winston proposes pointers from each description in memory to other descriptions, with each pointer labelled by a difference marker. Complaints about mismatch are

matched to the difference pointers leaving the frame and thus may propose a better candidate frame. Winston calls the resulting structure a Similarity Network.

Is a Similarity Network practical? At first sight, there might seem to be a danger of unconstrained growth of memory. If there are N frames, and K kinds of differences, then there could be as many as K*N*N interframe pointers. One might fear that:

(1) If N is large, say 10^7, then N*N is very large -- of the order of 10^{14} -- which might be impractical, at least for human memory.

(2) There might be so many pointers for a given difference and a given frame that the system will not be selective enough to be useful.

(3) K itself might be very large if the system is sensitive to many different kinds of issues.

But, according to contemporary opinions (admittedly, not very conclusive) about the rate of storage into human long-term memory there are probably not enough seconds in a lifetime to cause a saturation problem.

So the real problem, paradoxically, is that there will be too few connections! One cannot expect to have enough time to fill out the network to saturation. Given two frames that should be linked by a difference, we cannot count on that pointer being there; the problem may not have occurred before. However, in the next section we see how to partially escape this problem.

CLUSTERS, CLASSES, AND A GEOGRAPHIC ANALOGY

To make the Similarity Network act more "complete," consider the following analogy. In a city, any person should be able to visit any other; but we do not build a special road between each pair of houses; we place a group of houses on a "block." We do not connect roads between each pair of blocks; but have them share streets. We do not connect each town to every other; but construct main routes, connecting the centers of larger groups. Within such an organization, each member has direct links to some other individuals at his own "level," mainly to nearby, highly similar ones; but each individual has also at least a few links to "distinguished" members of higher level groups. The result is that there is usually a rather short sequence between any two individuals, if one can but find it.

At each level, the aggregates usually have distinguished foci or capitols. These serve as elements for clustering at the next level of aggregation. There is no non-stop airplane service between New Haven and San Jose because it is more efficient overall to share the "trunk" route between New York and San Francisco, which are the capitols at that level of aggregation.

The non-random convergences and divergences of the similarity pointers, for each difference d, thus tend to structure our conceptual world around

(1) the aggregation into d-clusters
(2) the selection of d-capitols

Note that it is perfectly all right to have several capitols in a cluster, so that there need be no one attribute common to them all. The "crisscross resemblances" of Wittgenstein are then consequences of the local connections in our similarity network, which are surely adequate to explain how we can feel as though we know what is a chair or a game -- yet cannot always define it in a "logical" way as an element in some class-hierarchy or by any other kind of compact, formal, declarative rule. The apparent coherence of the conceptual aggregates need not reflect explicit definitions, but can emerge from the success-directed sharpening of the difference-describing processes.

The selection of capitols corresponds to selecting stereotypes or typical elements whose default assignments are unusually useful. There are many forms of chairs, for example, and one should choose carefully the chair-description frames that are to be the major capitols of chair-land. These are used for rapid matching and assigning priorities to the various differences. The lower priority features of the cluster center then serve either as default properties of the chair types or, if more realism is required, as dispatch pointers to the local chair villages and towns. Difference pointers could be "functional" as well as geometric. Thus, after rejecting a first try at "chair" one might try the functional idea of "something one can sit on" to explain an unconventional form. This requires a deeper analysis in terms of forces and strengths. Of course, that analysis would fail to capture toy chairs, or chairs of such ornamental delicacy that their actual use would be unthinkable. These would be better handled by the method of excuses, in which one would bypass the usual geometrical or functional explanations in favor of responding to contexts involving art or play.

ANALOGIES AND ALTERNATIVE DESCRIPTIONS

Suppose your car battery runs down. You believe that there is an electricity shortage and blame the generator.

The generator can be represented as a mechanical system: the rotor has a pulley wheel driven by a belt from the engine. Is the belt tight enough? Is it even there? The output, seen mechanically, is a cable to the battery or whatever. Is it intact? Are the bolts tight? Are the brushes pressing on the commutator?

Seen electrically, the generator is described differently. The rotor is seen as a flux-linking coil, rather than as a rotating device. The brushes and commutator are seen as electrical switches. The output is current along a pair of conductors leading from the brushes through control circuits to the battery.

The differences between the two frames are substantial. The entire mechanical chassis of the car plays the simple role, in the electrical frame, of one of the battery connections. The diagnostician has to use both representations. A failure of current to flow often means that an intended conductor is not acting like one. For this case, the basic transformation between the frames depends on the fact that electrical continuity is in general equivalent to firm mechanical attachment. Therefore, any conduction disparity revealed by electrical measurements should make us look for a corresponding disparity in the mechanical frame. In fact, since "repair" in this universe is synonymous with "mechanical repair," the diagnosis must end in the mechanical frame. Eventually, we might locate a defective mechanical junction and discover a loose connection, corrosion, wear, or whatever.

One cannot expect to have a frame exactly right for any problem or expect always to be able to invent one. But we do have a good deal to work with, and it is important to remember the contribution of one's culture in assessing the complexity of problems people seem to solve. The experienced mechanic need not routinely invent; he already has engine representations in terms of ignition, lubrication, cooling, timing, fuel mixing, transmission, compression, and so forth. Cooling, for example, is already subdivided into fluid circulation, air flow, thermostasis, etc. Most "ordinary" problems are presumably solved by systematic use of the analogies provided by the transformations between pairs of these structures. The huge network of knowledge, acquired from school, books, apprenticeship, or whatever is interlinked by difference and relevancy pointers. No doubt the culture imparts a good deal of this structure by its conventional use of the same words in explanations of different views of a subject.

SUMMARIES: USING FRAMES IN HEURISTIC SEARCH

Over the past decade, it has become widely recognized how important are the details of the representation of a "problem space"; but it was not so well recognized that descriptions can be useful to a program, as well as to the person writing the program. Perhaps progress was actually retarded by ingenious schemes to avoid explicit manipulation of descriptions. Especially in "theorem-proving" and in "game-playing" the dominant paradigm of the past might be schematized so:

> The central goal of a Theory of Problem Solving is to find systematic ways to reduce the extent of the Search through the Problem Space.

Sometimes a simple problem is indeed solved by trying a sequence of "methods" until one is found to work. Some harder problems are solved by a sequence of local improvements, by "hill-climbing" within the problem space. But even when this solves a particular problem, it tells us little about the problem-space; hence yielding no improved future competence. The best-developed technology of Heuristic Search is that of game-playing using tree-pruning, plausible-move generation, and terminal-evaluation methods. But even those systems that use hierarchies of symbolic goals do not improve their understanding or refine their understanding or refine their representations. But there is a more mature and powerful paradigm:

> The primary purpose in problem solving should be better to understand the problem space, to find representations within which the problems are easier to solve. The purpose of search is to get information for this reformulation, not -- as is usually assumed -- to find solutions; once the space is adequately understood, solutions to problems will more easily be found.

The value of an intellectual experiment should be assessed along the dimension of success - partial success - failure, or in terms of "improving the situation" or "reducing a difference." An application of a "method," or a reconfiguration of a representation can be valuable if it leads to a way to improve the strategy of subsequent trials. Earlier formulations of the role of heuristic search strategies did not emphasize these possibilities, although they are implicit in discussions of "planning."

Papert (1972, see also Minsky 1972) is correct in believing that the ability to diagnose and modify one's own procedures is a collection of specific and important

"skills." Debugging, a fundamentally important component of intelligence, has its own special techniques and procedures. Every normal person is pretty good at them or otherwise he would not have learned to see and talk! Goldstein (AIM-305) and Sussman (TR-297) have designed systems which build new procedures to satisfy multiple requirements by such elementary but powerful techniques as:

1. Make a crude first attempt by the first order method of simply putting together procedures that separately achieve the individual goals.
2. If something goes wrong, try to characterize one of the defects as a specific (and undesirable) kind of interaction between two procedures.
3. Apply a "debugging technique" that, according to a record in memory, is good at repairing that specific kind of interaction.
4. Summarize the experience, to add to the "debugging techniques library" in memory.

These might seem simple-minded, but if the new problem is not too radically different from the old ones, then they have a good chance to work, especially if one picks out the right first-order approximations. If the new problem is radically different, one should not expect any learning theory to work well. Without a structured cognitive map -- without the "near misses" of Winston, or a cultural supply of good training sequences of problems -- we should not expect radically new paradigms to appear magically whenever we need them.

SOME RELEVANT READING

Abelson, R. P. "The Structure of Belief Systems." Computer Models of Thought and Language. Ed. R. Schank and K. Colby. San Francisco: W. H. Freeman, 1973.

Bartlett, F. C. Remembering. Cambridge: Cambridge University Press, 1967.

Berlin, I. The Hedgehog and the Fox. New York: New American Library, 1957.

Celce-Murcia, M. Paradigms for Sentence Recognition. Los Angeles; Univ. of California, Dept. of Linguistics, 1972.

Chafe, W. First Tech. Report, Contrastive Semantics Project. Berkeley: Univ. of California, Dept. of Linguistics, 1972.

Chomsky, N. "Syntactic Structures." (Originally published as "Strukturen der Syntax") Janua Linguarum Studia Memoriae, 182 (1957).

Fillmore, C. J. "The Case for Case." Universals in Linguistic Theory. Ed. Bach and Harms. Chicago: Holt, Rinehart and Winston, 1968.

Freeman, P. and A. Newell. "A Model for Functional Reasoning in Design." Proc. Second Intl. Conf. on Artificial Intelligence. London: Sept. 1971.

Gombrich, E. H. Art and Illusion, A Study in the Psychology of Pictorial Representation. Princeton: Princeton University Press, 1969.

Hogarth, W. The Analysis of Beauty. Oxford: Oxford University Press, 1955.

Huffman, D. A. "Impossible Objects as Nonsense Sentences." Machine Intelligence 6. Ed. D. Michie and B. Meltzer. Edinburgh: Edinburgh University Press, 1972.

Koffka, K. Principles of Gestalt Psychology. New York: Harcourt, Brace and World, 1963.

Kuhn, T. The Structure of Scientific Revolutions. 2nd ed. Chicago: University of Chicago Press, 1970.

Lavoisier, A. Elements of Chemistry. Chicago: Regnery, 1949.

Levin, J. A. Network Representation and Rotation of Letters. Publication of the Dept. of Psychology, University of California, La Jolla, 1973.

Minsky, M. "Form and Content in Computer Science." 1970 ACM Turing Lecture. Journal of the ACM, 17, No. 2 (April 1970), 197-215.

Minsky, M. and S. Papert. Perceptrons. Cambridge: M.I.T. Press, 1969.

Moore, J. and A. Newell. "How can MERLIN Understand?" Knowledge and Cognition. Ed. J. Gregg. Potomac, Md.: Lawrence Erlbaum Associates, 1973.

Newell, A. Productions Systems: Models of Control Structures, Visual Information Processing. New York: Academic Press, 1973.

Newell, A. "Artificial Intelligence and the Concept of Mind." Computer Models of Thought and Language. Ed. R. Schank and K. Colby. San Francisco: W. H. Freeman, 1973.

Newell, A. and H. A. Simon. Human Problem Solving. Englewood-Cliffs, N.J.: Prentice-Hall, 1972.

Norman, D. "Memory, Knowledge and the Answering of Questions." Loyola Symposium on Cognitive Psychology, Chicago, 1972.

Papert, S. "Teaching Children to be Mathematicians vs. Teaching about Mathematics." Int. J. Math. Educ. Sci. Technol., 3 (1972), 249-262.

Piaget, J. Six Psychological Studies. Ed. D. Elkind. New York: Vintage, 1968.

Piaget, J. and B. Inhelder. The Child's Conception of Space. New York: The Humanities Press, 1956.

Pylyshyn, Z. W. "What the Mind's Eye Tells the Mind's Brain." Psychological Bulletin. 80 (1973), 1-24.

Roberts, L. G. Machine Perception of Three Dimensional Solids, Optical and Optoelectric Information Processing. Cambridge: M.I.T. Press, 1965.

Sandewall, E. "Representing Natural Language Information in Predicate Calculus." Machine Intelligence 6. Ed. D. Michie and B. Meltzer. Edinburgh: Edinburgh University Press, 1972.

Schank, R. "Conceptual Dependency: A Theory of Natural Language Understanding." Cognitive Psychology (1972), 552-631. see also Schank, R. and K. Colby, Computer Models of Thought and Language. San Francisco: W. H. Freeman, 1973.

Simmons, R. F. "Semantic Networks: Their Computation and Use for Understanding English Sentences." Computer Models of Thought and Language. Ed. R. Schank and K. Colby. San Francisco: W. H. Freeman, 1973.

Underwood, S. A. and C. L. Gates, Visual Learning and Recognition by Computer, TR-123, Publications of Elect. Res. Center, University of Texas, April, 1972.

Wertheimer, M. Productive Thinking. Evanston, Ill.: Harper & Row, 1959.

Wilks, Y. "Preference Semantics." Memo AIM-206, Publications of Stanford Artificial Intelligence Laboratory, Stanford University, July, 1973.

Wilks, Y. "An Artificial Intelligence Approach to Machine Translation." Computer Models of Thought and Language. Ed. R. Schank and K. Colby. San Francisco: W. H. Freeman, 1973.

SUSSMAN AND McDERMOTT'S CONNIVER LANGUAGE

Advanced ideas on representing knowledge require advanced programming languages if those ideas are to be implemented. We therefore follow the section on frames with a discussion of the CONNIVER language of Sussman and McDermott, now well established as a milestone in A. I. programming language development. Historically, the language owes much to Hewitt's earlier PLANNER from which CONNIVER inherited a number of central ideas -- a pattern-directed data base search and pattern-directed subroutine call in particular. CONNIVER, however, goes considerably beyond PLANNER in providing the programmer with flexibility, power, and expressiveness, while at the same time avoiding PLANNER's bias toward a particular problem solving methodology.

THE PROBLEM WITH PLANNER

A higher level language derives its great power from the fact that it tends to impose structure on the problem solving behavior of the user. Besides providing a library of useful subroutines with a uniform calling sequence, the author of a higher level language imposes his theory of problem solving on the user. By choosing what primitive data structures, control structures, and operators he presents, he makes the implementation of some algorithms more difficult than others, thus discouraging some techniques and encouraging others. So, to be good, a higher level language must not only simplify the job of programming by providing features which package programming structures commonly found in the domain for which the language was designed, it must also do its best to discourage the use of structures which lead to bad algorithms.

A subset of Hewitt's PLANNER was rather haphazardly implemented by G. J. Sussman, T. Winograd and E. Charniak. This operational system is called MICRO-PLANNER (AIM-203A). It incorporates three basic PLANNER ideas: automatic backtrack control structure, pattern-directed data-base search, and pattern-directed invocation of procedures. Basically, backtracking is a way of making tentative decisions which can be taken back if they don't pan out. The pattern-directed data base search allows the user to ask for the data items called assertions which match a given pattern, and is intimately linked via the GOAL function to pattern-directed procedure invocation, which gives the user the ability to say "Find and run a program whose declared purpose matches this pattern." This type of program, called a theorem, is expected to instantiate the pattern (succeed), and thus simulate an assertion. In fact, it simulates a whole class of them, since failures back into any such theorem cause it to make different choices and succeed

with different instances.

How these mechanisms are related can best be illustrated by an example. The statement (GOAL (?A IN ?B)) is expected to assign the question-marked variables that do not have values already, or fail if it cannot, causing a backup to the last decision point in the program. GOAL instantiates its pattern by matching it against successive assertions, like (BLOCK2 IN BOX1). If it finds none, or enough failures propagate back to the GOAL to use up those it has found, it calls theorems with matching patterns, such as:

```
(CONSEQUENT (X Y Z) (?X IN ?Y)
        (GOAL (?X IN ?Z))
        (GOAL (?Z IN ?Y)) )
```

which expresses one facet of the notion that IN is transitive. A PLANNER program executing (GOAL (BLOCK2 IN ?B)) first checks to see if it "knows" the answer, and if so sets B to it. If not, it binds X to BLOCK2, links Y and B, enters the theorem, and looks for a Z containing BLOCK2 and contained in some Y. Its net effect is to assign a value to B.

If a failure propagates back into the theorem, it finds another Y containing Z, and hence generates another B; enough failures to use up those Y's drive it to find another Z; and a few more will make it and the original GOAL fail themselves. Backtrack control structure is the heart of this apparatus.

Automatic backtracking is implemented as follows: A PLANNER program, as it runs, grows a chronological stack of failpoints each of which corresponds to either a side effect or a decision point (a place where a choice is made between several alternative possibilities). A failpoint carries with it all information necessary to reconstruct the state of the running process at the time the failpoint was made. At some time, the process may decide to fail, perhaps because some decision made at a previous decision point led the process into a blocked state from which there are no viable alternatives. The failure then pops off the latest failpoint on the chronological stack. If this failpoint was a side effect, then it is undone, and the process continues failing. If this failpoint was a decision point, and there is another alternative, execution proceeds from that failpoint with the next choice taken. If there are none the failure continues to propagate. In these terms, GOAL finds exactly one assertion or theorem each time it is reached, but sets a failpoint which regains control if a failure should occur later.

Now after studying the experience of laboratory workers using PLANNER, Sussman and McDermott voice the following opinions:

1. Programs which use automatic backtracking are often the worst algorithms for solving a problem.

2. The most common use of backtracking can almost always be replaced by a purely recursive structure which is not only more efficient but also clearer, both semantically and syntactically.

3. Superficial analysis of problems and poor programming practice are encouraged by the ubiquity of automatic backtracking and by the illusion of power that a function like GOAL gives the user merely by brute force use of invisible failure-driven loops.

4. The attempt to fix these defects by the introduction of "failure messages" is unnatural and ineffective.

We must stress that PLANNER has introduced many valuable constructs into our way of thinking, the most important of which is pattern-directed search of a hierarchical data base. But automatic backtrack control structure has not been so valuable. PLANNER almost forces inefficiency in exactly the applications for which it was designed.

BUILDING CONNIVER

What then is to be done. What of PLANNER should be preserved and what replaced and improved upon? We have noted that PLANNER encourages a linkage of the mechanism for generating alternatives with the mechanism that restores the environment after the investigation of each one.

It seems to be the linkage of these two mechanisms in the GOAL statement that is at fault. As an alternative, imagine that we are not allowed to clean things up upon failure; that everything each goal-directed subroutine does stays done. Then, if the speculation it has indulged in is not to have effects in the environment of its caller (the program considering the alternatives), it must have a local environment of its own in which to make changes. These changes may make its model of the problem conflict with its superior's model, or may simply be hypothetical additions to it. The important point is that a simple return to the caller will be sufficient to make the changes invisible.

This concept can be made clear by analogy with the familiar notions of "control environment" (a stack, for example), and "access environment" (where variables are bound; the term is Bobrow and Wegbreit's (1972)); in CONNIVER, we generalize the latter to "data base environment," or context. Just as LISP 1.5 supports a tree of access environments ("association lists"), so CONNIVER supports a tree of contexts, in which each daughter-context represents a data base incrementally different from her parent.

This tree, it will be made clear, must be grown and maintained in conjunction with a control environment of equal complexity. But the control structure exists only to exploit the data base, so we return to it later.

CONNIVER contexts contain items, which are simple list structures like PLANNER's assertions (without the theorem-proving connotations that surround the latter term). An item such as (SQUARE48 PAWN3) may be added to the current context with

(ADD '(SQUARE48 PAWN3))

and taken out with

(REMOVE '(SQUARE48 PAWN3)).

The arguments to ADD and REMOVE are skeletons, list structures that define items after substitution of the values of their variables. Variables are indicated by ",". Thus, if X = PAWN2, (ADD '(SQUARE49 ,X)) adds the item (SQUARE49 PAWN2) to the current context.

Now, if the presence or absence of an item is to be reflected only in a local data base, that is, be "hypothetical," the data environment must be "pushed down" before doing ADD's and REMOVE's of this sort. Since, in CONNIVER, a context is a data type, and the current context is assigned to the variable CONTEXT, all we need to write is:

```
(PROG "AUX" ((CONTEXT (PUSH-CONTEXT)))
  (ADD '(SQUARE48 PAWN3))
       .
       .
       . )
```

CONNIVER syntax is roughly that of LISP, but a declaration of local variables must be explicitly indicated with the atom "AUX", and each such local must be given an explicit initial value, if it is not to be unassigned, by being declared as "(variable value)" instead of just "variable." This PROG thus rebinds CONTEXT to the value returned by the system function PUSH-CONTEXT. The current context has had one more context-frame pushed onto it. New changes apply to this frame only. After the body of the PROG has been executed in this "hypothetical" context, the PROG's control frame will be exited. CONTEXT will be unbound, restoring its old value, in which the action of the ADD is invisible; in effect, a data frame has been exited as well.

From now on, we shall abbreviate a construction such as the above as

(ASSUMING (SQUARE48 PAWN3) ...),

to emphasize concisely the nature of the computation "..." as proceeding in a slightly

new environment. However, it should not be forgotten that, since contexts are data structures, they can be returned as values of functions, assigned to global variables, etc., so that in fact the user has available a tree of contexts his program has left behind, in the same way that using functional arguments (closures of functions) in LISP creates a tree of variable-value associations.

Items can be retrieved from the current context by means of the CONNIVER primitive FETCH, which finds all items present in the context that match a pattern. For example, if we let the presence of the item (obj1 ON obj2) mean "object obj1 is resting on top of object obj2," we can find all the objects on BOX2 with

(FETCH '(?X ON BOX2)).

Roughly as in PLANNER, the "?" indicates that the variable X is to be assigned a value by matching the pattern (?X ON BOX2) against some item. However, FETCH does not make the assignment. Since backtracking has been exorcised from CONNIVER, it simply returns a possibilities list which points to all the matching items, rather than hiding them in a failpoint in GOAL, to be handed to us coyly, one per failure. The user can manipulate this list in any way he chooses; one way is with the system function TRY-NEXT, which pops off and returns the first item in the list, and assigns the pattern variables as the possibility directs. For example, if PYRAMID6 is the first object found, (TRY-NEXT (FETCH '(?X ON BOX2))) sets X to PYRAMID6.

A useful "canned-loop" function, which is implemented using FETCH and TRY-NEXT, is FOR-EACH, defined so that

(FOR-EACH pattern ...)

performs the computation "..." for each assignment of the pattern's variables corresponding to an item in the data base. Hence,

```
(FOR-EACH (?X ON TABLE)
   (PRINT X))
```

prints the names of all objects resting on TABLE.

As in PLANNER, we want to be able to include a set of items in the current context on the basis of some procedural criterion instead of their actual presence. In PLANNER, this ability was attained by the use of consequent theorems, which behave as failure-driven assertion-generators when invoked by GOAL statements. Since there is no backtracking in CONNIVER, the analogous CONNIVER structure, the if-needed method, must return all of its possibilities in a single bundle. The simplest type of if-needed uses the primitive NOTE to save the current instantiation of its pattern with the values

of the variables, which were not assigned when it was entered, but have been computed by the method. When the method is exited, it returns a possibilities list which includes all the NOTEd instances.

For example, if the presence of an item of the form (obj1 SUPPORTS obj2) is to mean "obj1 is helping maintain obj2 in its present position," the following method expresses part of the idea that if an object is on another, it is supported by it:

```
(IF-NEEDED SUP-ON (!X SUPPORTS !?Y) "AUX" (X Y)
   (FOR-EACH (?Y ON ,X)
      (NOTE) ))
```

The "!" and "!?" prefix characters mark method-pattern variables that are to receive or to avoid receiving a value, respectively, when the method is entered. The "!?"-variables are to be assigned in the body of the method (here, by the TRY-NEXT of a FOR-EACH loop); hence, this method returns a list of all generated item possibilities for Y's supported by a given X.

Methods like SUP-ON are treated by the system, as other data, as present or absent in a given context. If a method matching a FETCH or FOR-EACH pattern is found, a method possibility is put into the possibilities list produced, to be invoked by TRY-NEXT when the possibilities before it are used up. Upon returning, the method replaces itself on the caller's possibility list by the instances it generated. Hence, the presence of SUP-ON in a context represents the presence of the items it can produce on demand.

As an illustration, suppose the items (TABLE SUPPORTS APPLE3), (PLATE1 ON TABLE), and (BOX2 ON TABLE) are present. Then the loop

```
(FOR-EACH (TABLE SUPPORTS ?OBJ)
   (PRINT OBJ))
```

prints

```
APPLE3
PLATE1
BOX2,
```

with the method SUP-ON simulating the two items for the second two objects.

A more complex method may require the creation of a hypothetical data base:

```
(IF-NEEDED SUP-UN (!X SUPPORTS !?Y) "AUX" (X Y)
   (ASSUMING (,X VANISHES)
      (FOR-EACH (PHYSICAL-OBJECT ?Y)
         (COND ((UNSTABLE Y)
               (NOTE)) )) )).
```

This method creates a hypothetical context in which X no longer exists, and sees which objects are no longer stably immobile according to the (sophisticated) function UNSTABLE. These are NOTEd, and, as before, all objects X supports (by this criterion) are found before SUP-UN returns.

Our concept of generator appears simpler than PLANNER's; methods like these dump all their instances into the caller's possibilities list and return, leaving their control environments and any bindings of CONTEXT to be collected as garbage. Even if its caller wants only one new item possibility, such generators give him all of them. The scheme we have described does not allow the caller to influence the order or selection of objects to be generated. If each generation is expensive, as, in SUP-UN, a call to UNSTABLE might very well be, a lot of unnecessary overhead may be incurred if not all members of the set at issue are wanted. If, in fact, this set is infinite, the scheme breaks down completely.

We have returned to our original problem: how can we maintain in existence the control and context structure of a generator while returning from it with only a few of the possibilities it can find? The answer lies in the structure and function of the possibilities list; to invoke a method found in such a list is to replace the method by its value, itself a list of possibilities. If this value list contains a generalized tag back to the generator's activation, its environment will be preserved and accessible. Not only that, but if TRY-NEXT comes upon such a thing in a possibilities list, it is bound to transfer control to it. Now the method can generate items in finite groups, asking to be reawakened if none of the items satisfies its caller. A new version of SUP-UN that works this way looks like:

```
(IF-NEEDED SUP-UN2 (!X SUPPORTS !?Y) "AUX" (X Y)
   (ASSUMING (,X VANISHES)
      (FOR-EACH (PHYSICAL-OBJECT ?Y)
         (COND ((UNSTABLE Y)
               (NOTE)
               (AU-REVOIR)) ))) ).
```

AU-REVOIR causes an immediate return from the method when the first Y is found, but also returns a tag to its own activation. Remember that SUP-UN2 is interacting with a possibilities list (perhaps hidden in a FOR-EACH) at a higher level. The method itself was found there, representing a set of simulated items; now when it returns, it leaves one new supportee, plus an AU-REVOIR tag which similarly represents the set of remaining possibilities it knows about. From the point of view of a FOR-EACH loop, this type of possibilities list is equivalent to the previous exhaustive list, but it differs in several crucial ways.

First, the list is as short as the generator wishes to make it, no matter how large the set it can generate if requested. Second, such a list represents a generation-in-progress which is not complete; the calling process that asked for it is in a position to intervene and advise the generator how to proceed. Third, an AU-REVOIR tag can be treated explicitly as a datum representing a parallel problem investigation and associated world-view. A generator's caller can use such a tag to do relative evaluations, close functions, and fiddle with its binding of CONTEXT. Notice, for example, that a program that uses SUP-UN2 has available a pointer to an incompatible environment where the object X no longer exists; a more sophisticated version could use this ability to communicate the context and remaining physical object possibilities to take advice from its caller on how to generate more objects supported by X.

The requirement that there be generalized tags, tags that mention whole control environments, makes it necessary that CONNIVER maintain a control tree similar in structure to the context tree it serves. All such still-viable environments form a set of processes cooperating to solve a problem. Some of these are generators, using possibilities lists as communication channels with their callers, but this by no means exhausts the alternative ways of interacting. In particular, CONNIVER's generalized control structure makes it easy to put all of failure and backtracking back in if the user wants them, but he has the duty (or privilege) of designing and maintaining control over what he builds.

SOME RELEVANT READING

McCarthy, J., et al. LISP 1.5 Programmer's Manual. Cambridge: M.I.. Press, 1962.

Winston, P. H. "The M.I.T. Robot." Machine Intelligence 7. Ed. D. Michie and B. Meltzer. Edinburgh: Edinburgh University Press, 1972.

Greenblatt, R., et al. "The Greenblatt Chess Program." Proc. Fall Joint Computer Conference, 31 (1967), 801-810. Also A. I. Memo 174, Publications of the Artificial Intelligence Laboratory, M.I.T., 1967.

Bobrow, D. G. and B. Wegbreit. A Model and Stack Implementation of Multiple Environments. Report No. 2334, Publications of Bolt, Beranek and Newman, Inc., Cambridge, Massachusetts, 1972.

FAHLMAN'S CONSTRUCTION PLANNER

Having reviewed CONNIVER, we will turn to Fahlman's BUILD program in part to demonstrate what can be done with CONNIVER. The BUILD program plans complicated Blocks world construction sequences using substructures, counterweights, and scaffolds as required. In the course of this planning, heavy use is made of CONNIVER's sophisticated data and control structure whenever some instability erupts and requires interaction between higher level decision making routines and lower level block movers and stability analyzers.

BUILD is passed two world models, each showing blocks of various sizes and shapes piled upon a table. One model represents the current situation, and the other specifies a desired goal state. BUILD must create a plan for moving from one state to the other using only those primitive actions which could be accomplished by a one-handed robot: single block movements and the movement of stable block sub-assemblies. None of the intermediate block configurations created by this plan may be gravitationally unstable.

It is one-handed operation that makes BUILD's world interesting. Consider, for example, how the "seesaw" of figure 1 might be constructed. Straightforward bottom-up assembly is no good, since block C will topple if either A or B is placed alone. There are solutions, however: The A-B-C structure can be assembled on the table, then grasped by C and lifted into place. If there are some spare blocks available, they can be used to create a temporary support or "scaffold" to hold up one side of C while A and B are placed. Figure 2 shows a variable-height scaffold made of two bricks and a wedge. Finally, a temporary counterweight can be used to hold C down while A and B are placed, as in figure 3.

BUILD can employ any of these tricks when necessary -- can even combine several tricks to build complicated structures -- but far more important is its flexibility in dealing reasonably with all of the little problems and combinations of problems that confront it at every turn. Specifically, BUILD can do the following sorts of things:

* Dig up needed blocks that are buried under other blocks.

* Find or clear table space for unwanted blocks and for sub-assembly construction.

* Make use of any pre-existing portions of the goal structure, if practical.

* Borrow scaffold parts from other structures.

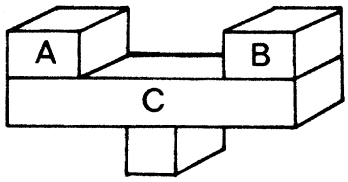


FIGURE 1

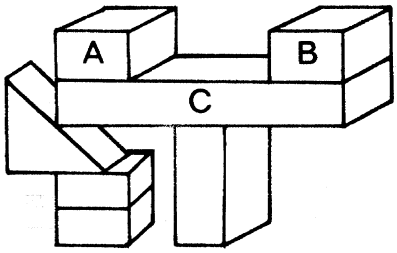


FIGURE 2

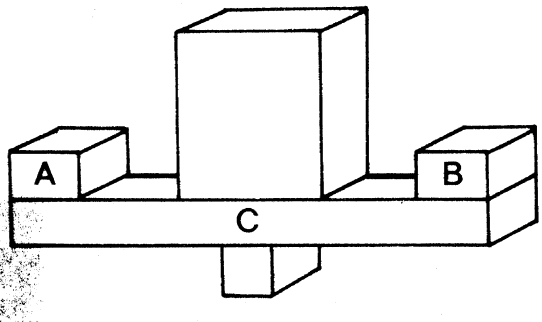


FIGURE 3

* Pick the best (least-buried) blocks to use when a choice is possible.

* Work backwards from the goal when this is easier.

* Create and study hypothetical states as intermediate goals.

* Go back over its own plans to make minor changes or eliminate redundant steps.

* Accept suggestions ("Try counterweight first") and restrictions ("Don't move any block more than once") easily and naturally, although not in English.

* Plan in a headlong manner, worrying about low-probability disasters only if they are encountered.

* Switch control among several search paths, depending upon which looks best at the moment.

Build is able to maintain several world models at once by keeping each model in a separate context in the CONNIVER data base. Routines are provided for determining face and vertex positions from the shape and position of a block and for checking whether two objects are touching. A very powerful stability test is able to trace out the support relationships even in complex scenes involving friction. If a structure is unstable, this test can determine exactly which blocks will fall, slide, or pivot in which directions. Such complete information about the nature of a failure is of immense value to BUILD in finding an appropriate remedy.

BUILD employs a "virtual data base" storage scheme. When it needs some piece of information -- the position of a vertex, perhaps, or the support relationship between two blocks -- BUILD sends a pattern match request off to the data base. If the desired fact is there, fine; if not, an appropriate routine is called to derive the needed information from other facts, and the answer is saved in the data base for future reference. Thus, the system has direct access to any fact that is within its power to compute, without having to work everything out before it is needed. Nothing is computed and never used; nothing is computed twice. BUILD can jump off to any routine at any time, without worrying about the state of preparations in the data base.

BUILD's flexibility and power are made possible by its control structure, which fully exploits CONNIVER's multi-process capability. Any function invocation in CONNIVER can

be kept alive even after the function has returned or passed control elsewhere. This is done by creating a tag into the body of the function and saving this tag in some external location. Using this tag, other programs can ask any questions they want about the suspended process or its environment, or they can restart the process by GOing to the tag. It is this mechanism that allows BUILD to switch control easily from one search path to another, though only one process is ever running at any given time.

Errors and failures are dealt with in BUILD by something called a CHOICE-GRIPE mechanism. Whenever a major choice is made, a tag into the choosing routine is pushed onto a stack and the system proceeds. If some sub-goal later runs into trouble (a block structure collapses, for instance) the losing procedure generates a GRIPE consisting of a tag to itself and a brief message describing the failure, and control is passed to a gripe-catcher associated with one of the previous choice-tags. Through its tags, this catcher has access to any information it needs, either about the choice or about the disaster that resulted from it, and is in an ideal position to propose a fix. Earlier systems either received no information about a failure or had to make do with a terse and probably cryptic message.

The catcher can react in any of several ways: It can clean up the mess and try a different choice. It can splice a minor patch into the losing plan or slightly modify the losing sub-goal's instructions if it appears that such actions will help. It can pass the gripe on up to a catcher associated with a more global choice. If the gripe is non-fatal, the catcher can either dismiss it, perhaps sending down some advice about how to proceed, or it can suspend the complaining process while other alternatives are tried. This ability to analyze a failure and select an appropriate response is something which has long been discussed in A.I.circles, but BUILDis one of the first programs to achieve this.

Thus BUILD represents an attempt to bring together in a concrete form a large number of problem-solving ideas that have been in the air for some time. We believe it will serve as a useful paradigm for problem-solving efforts in many domains.

3 UNDERSTANDING ENGLISH AND ASSIMILATING KNOWLEDGE

Much of the newer work on understanding English text is of a continuing nature and will be featured in future reports. This section is confined, for the most part, to completed pieces of work.

The section opens with a brief review of Winograd's blocks-world discourse system because it established the context in which the later work has been done (TR-235). Our last major report described Winograd's ideas at greater length.

WINOGRAD'S DISCOURSE SYSTEM

Winograd's system contains the syntactic, semantic, and deductive expertise required to engage in discourse about a simulated Blocks world like that of figure 1. SHRDLU, the robot, can answer questions and respond to commands, all using quite natural English.

Winograd concentrated on bringing out the semantic issues in his work. His grammar, for example, is one whose purpose is to find and understand groups of words that convey meaning. There are four simple kinds of groups:

NOUN GROUP
VERB GROUP
ADJECTIVE GROUP
PREPOSITIONAL GROUP

On a higher level, various combinations of groups may form a clause. These clauses in turn may legally appear as parts of the various groups, allowing a recursive quality to be exhibited in sentences understood by the analysis routines.

It is often convenient to show the containment relationships among the various groups and words of a sentence by using a tree-like diagram like that in figure 2.

In order to understand a sentence, Winograd's syntactic system builds descriptive properties called features into the nodes in the tree. See figure 3.

For the most part the meaning of the features used can be guessed from their mnemonic names. The syntax specialist picks up many of them from the word dictionary. Others are deduced from known rules constraining how sentences are put together. For example, as soon as the syntax specialist has organized the features derived from the fragment "Pick up..." it knows that only a noun group or a so-called rank shifted clause

The Robot's World

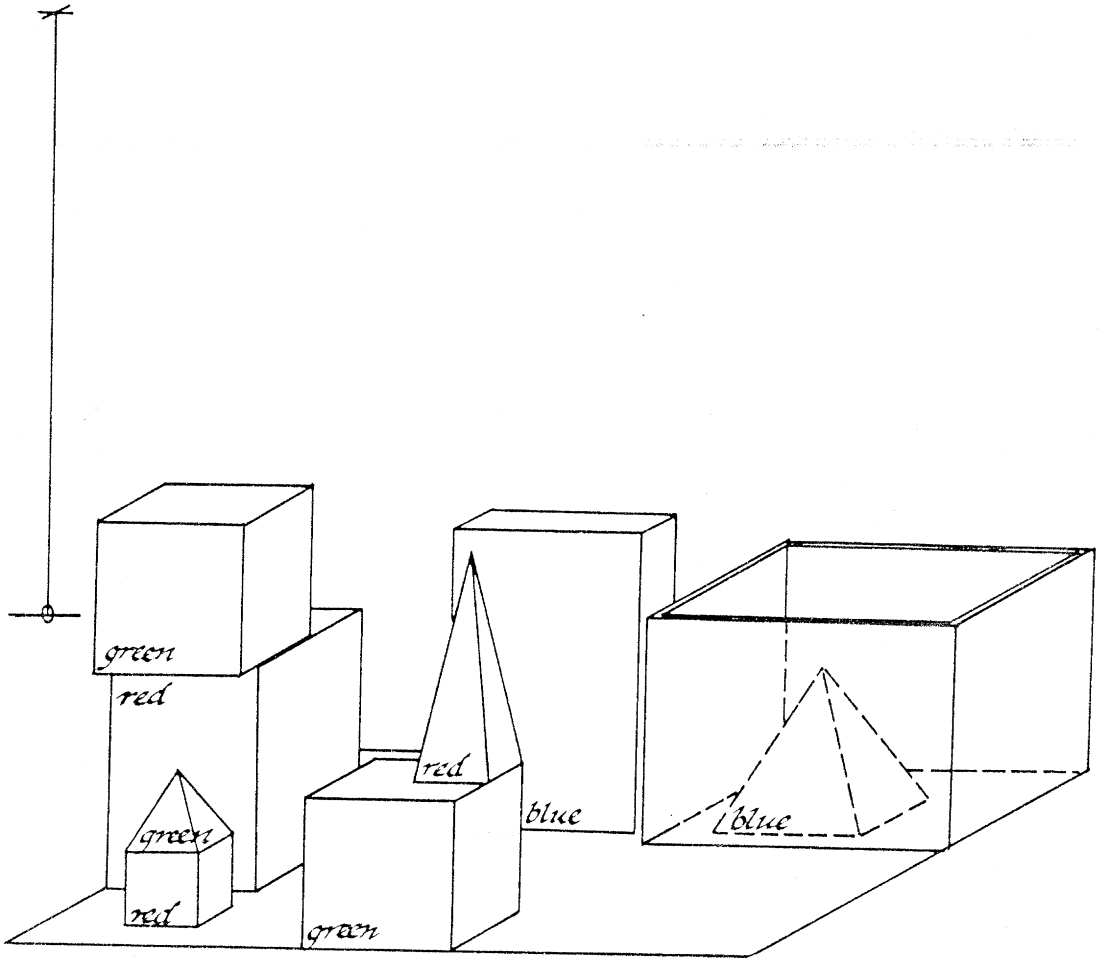


FIGURE 1

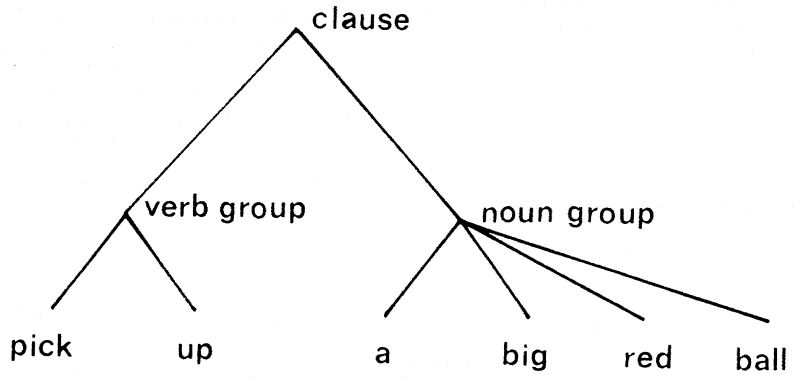


FIGURE 2

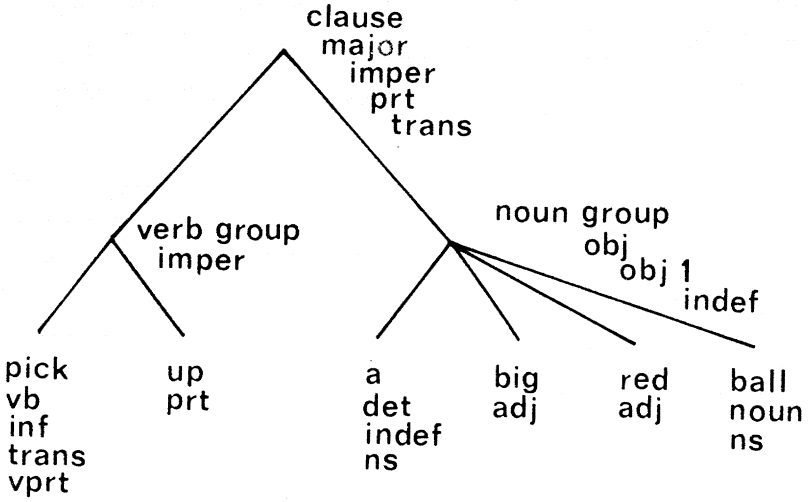


FIGURE 3

can follow. This together with the dictionary features for "a" allow a node with the noun-group, object and object1 features to be established immediately. Thus accumulated features guide the remaining analysis.

A key idea in Winograd's work is the idea that descriptions in English can be translated into descriptions in the form of programs. The system translated the fragment "a big red block which supports the pyramid" into a program matched to it as follows:

```
A
BIG        (FIND >0 (GOAL (BIG ?X))
RED                 (GOAL (RED ?X))
BLOCK                (GOAL (BLOCK ?X))
WHICH
SUPPORTS              (FIND =1 (GOAL (SUPPORTS ?X ?Y))
THE
PYRAMID                       (GOAL (PYRAMID ?Y)) ))
```

The program fragments are found in the word dictionary under entries for the words big, red, block, support, and pyramid. And when assembled as shown the result is a program which inspects the data base in an effort to find the name, say B18, of some big red block which in fact supports the pyramid. The English description has become a program whose execution is relevant to answering questions about or manipulating the object described.

What about the other words in the fragment? What do they contribute? They are indeed important because the features of the words a, the, and which, provide the instructions on exactly how the dictionary code fragments must be put together. The use of >0 and =1 in the example derive directly from such word-feature considerations.

CHARNIAK'S CHILDRENS STORY SYSTEM

Winograd's system proved very talented indeed at discourse confined to the simulated blocks world. It is so good that people often think more of the natural language problem has been solved than is in fact the case. Deep issues remain, among them the problem of bringing general common-sense facts to bear on particular story-like situations. We now turn to Charniak's brave work on children's stories to see what sort of things are involved.

A typical story studied by Charniak might start:

> Fred was going to the store.
> Today was Jack's birthday and Fred was going to get a present.

Some typical questions would be:

> Why is Fred going to the store?
> Who is the present for?
> Why is Fred buying a present?

There are two points which we should note about such questions. First, they are not answered explicitly in the text. That is, the story did not say "Fred was going to the store because he ...." The story does not even contain a full implicit answer; one cannot logically deduce an answer from the statements in the story without using general knowledge about the world such as:

> Objects "got" at stores are usually "bought."
> Presents are often bought at stores.
> If a person is having a birthday, he is likely to get presents.

To explicate the role of world knowledge in understanding, Charniak proposes the following strategy:

1) The problem is divided into two parts. The first half concerns taking the natural language and translating it into an "internal representation." This internal representation is a form which is convenient for making deductions. "Internal translation," as we shall call the first part, is restricted to processes which could

be performed by a person seeing the line completely out of context. The second part of the two part division is called "Deep Semantic Processing (DSP).

2) Rather than consider the entire problem of natural language comprehension, only DSP is considered.

3) A DSP model should try to "fill in the blanks" of the story on a line by line basis. That is, as it goes along, it will try to make connections between events in the story (usually causal connections) and fill in missing facts which seem important (such as who was going to receive the present).

4) The model will build up a data base which contains the facts given by the story along with any deductions made in accordance with (3) above. The model will try to keep this data base consistent and non-redundant.

5) A fact is only "made available" for making deductions when there is reason to believe that it is relevant to the story. For example, going back to the first story, the fact "people get presents from others on their birthdays" would be associated with the concept "birthday," and would hence be introduced to the data base by the line "Today was Jack's birthday." Typically the model will immediately try to use the newly introduced fact. However, in the "birthday" example it is not until the next line, "Fred was going to get a present" that we find anybody obtaining a present, and so it is then that our fact can be used and allow us to conclude that the present is for Jack. So in many cases facts introduced earlier in the story are used to interpret lines which occur later in the story. Charniak thinks of such facts as being "demon-like."

6) The model attempts to handle many kinds of statements about intentions, desires, and responsibilities which frequently come up in children's stories. In particular, they cause portions of general knowledge to become relevant to the story. Hence this information can affect later lines, as in (5).

AN EXAMPLE

To provide a better idea of what is involved, this section is devoted to a detailed example. This example was handled by a computer program, but the model of comprehension reflected in the computer program was preliminary and not identical to the model Charniak ultimately proposed.

WHAT THE PROGRAM DOES

The program's input (and output) is not English, but "preprocessed" English, in the sense that it has already been put into a form which closely resembles the internal format. This however leaves open the question of what information is explicit in the input, and what the program deduces for itself. The only information which can be included in the input is information which a normal reader can get solely out of an individual sentence without recourse to context. For example, since any noun phrase might be referring to an object which was introduced by an earlier line of the story, such reference decisions are not resolved in the input language.

This story is taken directly from "Up and Away," a first grade reader (Mc Kee et al. 1966). Several sentences have been deleted for brevity.

THE EXAMPLE

Jack and Janet were in the house. Jack was holding a box of pencils and a box of paints.

> EVERYTHING INDENTED AT THIS LEVEL IS COMMENTARY FOR THE READER. The first two lines of the story are not in the original version which has a picture instead.

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

Question: Did Daddy get the paints for Janet?
Answer: No, Daddy got the paints for Jack.

It should be noted that the original sentence about getting the paints and pencils for Jack and Janet was ambiguous. It might have meant that they were for the children to share. If we had asked this question 3 lines ago the program would have responded "I don't know."

Janet said to herself, "I want the paints."
Jack began to paint a picture of a red 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.

This question involves several points. The program interprets "funny" as "bad" in this context (this is not done for it in the input format). Janet really said that the picture was bad, and it is necessary to transfer this to the paints. We must know that if you want something another person has, you may make nasty comments about it in order to get it. Also, 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." At this point it should be realized that the program needs a lot of information about wanting, trading, giving, owning, and strategies for getting something from another person. While this example shows that the program has some knowledge of these topics, it knows virtually nothing about everything else.

Janet continues "I will let you have the pencils. I will take the paints."

Question: What does Janet want to do?
Answer: Trade the pencils for the paints.

Jack looked at the pencils.
"No, thank you, Janet," he said. "I want to paint more pictures."
Janet said to herself, "I do not want these pencils."
Soon Janet came back with something for Jack.
"Jack, here is a toy cat to put your money into," she said. "I will give you this cat and all my pencils for your paints."

Question: Why did Janet get the toy cat?
Answer: To trade with Jack.

The program realizes that "something" (in "Soon Janet came back with something for Jack.") and toy cat are the same object. (Again, this is not done in the input language.) It has to know that returning to a place where you just were with a given object constitutes getting the object. We might note that we could remove the "for your paints" from "I will give you this cat and all my pencils for your paints," and still understand Janet as meaning "trade." Of course, "give" does not always mean trade; only in certain contexts. This and the "funny" example point out the fact that inference is also necessary for deciding "meanings."

Question: Why did Janet offer to give Jack the cat and the pencils?
Answer: She wanted the paints, and Jack would not trade for the pencils.

"I have no money," said Jack. "What do I want with a toy cat? I want to paint more pictures."

Question: Will Jack trade?
Answer: No.
Question: Why won't Jack trade?
Answer: He doesn't want the cat and he wants the paints.

Actually "What do I want with an X" is treated as an idiom, meaning "I don't want an X" and is input this way to the program.

"I'll have to get some money for Jack," Janet said to herself.
Soon she came back with it. "Jack, here is some money to put into the toy cat," she said. "Now you will have some money, a toy cat, and all these pencils."

The phrases "some money" and "a toy cat" out of context may or may not refer to the cat and money Janet is holding. The program decides that they do since it is anticipating a trade offer, and this can only be construed as a trade if the objects in question are Janet's.

Jack laughed and said, "Take your things and go away.

Question: Will Jack trade?
Answer: No.

The program interprets the phrase "go away" as implying that the person who said it does not want to do any current or suggested activity.

I want to paint pictures."

Question: Why did Janet kick Jack?
Answer: I was not aware that Janet kicked Jack.
Question: Where is Bill?
Answer: I don't know, but it seems like a silly question since no Bill is mentioned.
Question: When did Janet get the cat?
Answer: Before she offered to trade the pencils and the cat for the paints.

"When" questions are answered by mentioning an event which happened at roughly the same time. The major problem is making sure that the event used is important enough in the story so that the reader is certain to remember it. In a longer story this becomes very important, though even here it makes a difference. The response could have been "Just after Jack looked at the paints" which occurred a few lines before Janet goes to get the cat. However, this response seems much less natural than the one given. Currently the test for importance is checking that the line has causal links with other parts of the story.

Question: Does Jack know that Janet wants the paints?
Answer: Yes, he was there when she offered to trade which implied that she wanted the paints.

When Janet said "I want the paints" she said it to herself. While the program can make use of this information, Jack cannot. So instead the system notes that "wanting the paints" is the reason for Janet offering to trade. It then assumes that Jack could figure this out also.

AN OVERVIEW OF A PROPOSED SYSTEM

Charniak did not implement his ideas other than those required for the handling of the dialogue fragment given above, believing the specification of a very ambitious system to be a better allocation of resources than the implementation of a more modest one. The following sections describe some of his resulting views.

DEEP SEMANTIC PROCESSING

There are four main components of inference or deep semantic processing (DSP):

Demons - Facts which are introduced by "concepts" occurring in the story are called "demons" since in many cases they must wait for further information. In such cases we can think of them "looking" for the appropriate fact. So "not being willing to trade" might put in a demon looking for a better offer.

Base routines - These constitute what we know about a "concept" independent of "context." So, for example, if A gives B to C then C now "has" B. This is not dependent on what happened earlier in the story.

Bookkeeping - This does chores like keeping the data base relatively consistent and non-redundant. So, should a person in the story change location, the old location statement must be updated.

Fact finders - These are utility routines for doing standard deductions which are not worth asserting separately. A typical fact finder might say, "If you want to know if person P knows fact F, just see if when F occurred, (or was said by some character in the story) P was around."

Let us now consider what these notions might involve in more detail.

DEMONS

Consider a fact like:

(4) "If it is raining" and

"If person P is outside" ==> "P will get wet"

We have an intuitive belief that (4) is a fact about "rain," rather than, say, a fact about "outside." Many things happen outside and getting wet is a very small part of them. On the other hand only a limited number of things happen when it rains.

This belief is embodied by associating (4) with "rain" so that only when "rain" comes up in the story will we even consider using rule (4). Rain is the "topic concept" of (4). When a concept is brought up in a story, the facts associated with it are "made available" for later use. (The facts are "put in" or "asserted.") So, if "circus" say, has never come up, the program will not be able to make deductions using those facts associated only with "circus."

Note however that "rain" does not need to be mentioned explicitly in the story before we can use (4). It is only necessary that there be a "rain" assertion put into the data base. Other parts of the story may provide facts which cause the program to assert that it is raining. For example:

(5) One afternoon Jack was outside playing ball with Bill. Bill looked up and noticed that the sky was getting dark. "I think we should stop" said Bill. "We will get wet if we keep playing."

Here, the sky getting dark in the afternoon suggests that it is going to rain. If this is put into the data base it will be sufficient to bring in facts associated with "rain." (Actually, to account for (5) we would need to modify (4) slightly, since (4) requires that it is raining, and in (5) we only suggest that it will rain.

Also note that a topic concept need not be a single "key word." A given set of facts may not become available to the system until a complex set of relations appear in the data base.

LOOKING FORWARD, LOOKING BACK

Only considering facts after we have seen the "topic" concept allows us to see the topic concept before we have all the information needed to make use of the fact. This would be no problem in a delayed deduction scheme because a rule is only used when the user askes a question. If the facts which enable the rule to work are missing, it simply means that the rule will not be used. But, when making deductions "on the fly," if

the necessary information comes in after the rule has been introduced we want to make the deduction when the information comes in. So we might have:

(6) Jack was outside. It was raining.
(7) It was raining. Jack was outside.

In (6) there is no problem. When we introduce "rain" we have sufficient information to use (4) and deduce that Jack is going to get wet. But in (7), we only learn that Jack is outside after we have mentioned rain. If we want to use (4) we will need some way to have our fact "look forward" in the story. To do this we will break a fact up into two parts, a pattern and a body (a program). We will execute the body of the fact only when an assertion is added to the data base which matches the pattern. So with (4) the pattern would be "someone outside." Hence in (7) we introduce (4) when we see "rain." At that time no assertion matches the pattern. But the next line will create a matching assertion, so the body of the fact will then be executed. Hence we will say that a fact is "looking forward" when the assertion which matches its pattern comes after the assertion which made the fact relevant to the story. When the assertion which matches the pattern comes before we will say that the fact is "looking backwards."

We can see how important looking forward is with a few examples.

(8) "Janet and Penny went to the store to get a present for Jack. Janet said 'I will get Jack a top' 'Don't get Jack a top.' said Penny. 'He has a top. He will make you take it back.'" We interpret the line "Jack has a top" as meaning that he did not want another. The common sense knowledge is the fact that in many cases having an X means that one will not want another X. This piece of information would probably be filed under "things to consider when about to get something for somebody else." Naturally it was an earlier line which mentioned that Janet was thinking of getting Jack a top.

(9) "Jack was having a birthday party. Mother baked a cake." The second line is interpreted as meaning "for the party" on the basis of information about birthday parties brought in by the first line.

(10) "Bill offered to trade his pocket knife for Jack's dog Tip. Jack said 'I will ask Janet. Tip is her dog too.'" The last line is interpreted as the reason Jack will ask Janet because of information about the relation between trading and ownership.

(11) "Janet wanted to get some money. She found her piggy bank and started to shake it. She didn't hear anything." The last line means that there was nothing in the piggy bank on the basis of facts about piggy banks.

In each of these cases it is an earlier line which contains the information which is used to assign the interpretation. So in the first example there is nothing inherent in the line "Jack has a top" which means "don't get him another." Suppose there were. Changing the example to "Jack has a ball," something in the line would have to key a check for the following situations as well:

(12) "Bill and Dick wanted to play baseball. When Jack came by Bill said "There is Jack. He has a ball."

(13) Tom asked his Father if he would buy him a ball. "Jack has a ball" said Tom.

(14) Bill's ball of string was stuck in the tree. He asked Jane how he could get it out. Jane said "You should hit it with something. Here comes Jack. He has a ball."

Those familiar with PLANNER might notice that the "facts" look quite similar to Planner antecedent theorems, with the exception that facts can "look back" as well as "look forward." Antecedent theorems are only designed to look forward. Another difference is that while antecedent theorems are automatically executed when a relevant assertion is entered into the data base, demons will be called some time after the assertion has been created.

POSSIBLE FUTURES

Charniak requires that a concept can be introduced by the story before facts associated with it become available for making further deduction. One aspect of this decision is the significance it gives to statements about future events. In general,

statements in future tenses are problematic. While it is usually clear what a present tense statement means, what is one to make of some statement that "Jack must buy a new suit?" It does not mean that we can be certain that at some point in the future we will see Jack buying a new suit. It certainly does not mean that Jack is currently at the clothing store. So what is one to do with such a statement? Such statements ("possible futures") introduce facts that can be used to interpret some of the things which may happen in the story. Once we see "must buy a new suit" we will be looking (forward) for the person's going to a clothing store and, when and if he does so, we will know why. Note that this analysis applies equally well to "can buy a new suit," "will buy...," "should buy...," etc. This is not to say that all these phrases mean the same thing. Rather, the concept of possible future is designed to capture what they have in common. Their differences will have to be accounted for by other means.

BASE ROUTINES

So far we have said that demons are introduced to the story when the proper concept has been mentioned. But this implies that there is something attached to the concept name telling us what demons should be put in.

If we look at a particular example, say (10), it is Bill's offer to trade the pocket knife for Tip, which sets up the context for the rest of the fragment. So we must have some information, which this line somehow accesses, telling us to activate certain demons. We will assume that this information is in the form of a program. Such routines, which are available to set up demons, will be called "base routines" and will be designated by -BASE at the end of their name, as in TRADE-BASE which is activated when an assertion with the symbol TRADE is placed in the data base..

These base routines will be responsible for more than setting up demons. Suppose we are told that Jack had a ball, and Bill a top. Then Jack traded his ball to Bill for the top. One question we might ask is "Who now has the top?" Naturally since questions of "who has what" are important in understanding stories we will want to keep tabs on such information. In this particular case, it must again be the "trade" statement which tells us to switch possession of the objects. To be more formal, we would say "trade" entails the fact that the objects have switched owners. (Though note that I am not using the formal logical definition of "entails.") Since every time a trade occurs we will want to exchange objects, it must be the case that whenever we see "trade" we execute TRADE-BASE. Of course, the program can't be too simple minded, since it must also handle "I will trade ..." and perhaps even "Will you trade ...?"

There might be some question as to whether demons and base routines are really distinct entities. When a demon is looking forward it is waiting until an assertion is added to the data base which matches its pattern. This is exactly what a base routine does, but while the demon will only be asserted when we have encountered the proper concept, base routines are always available.

Of course, we are only looking at children's stories. Will we want the same set of base routines for reading both fiction and chemistry, or will base routines also be shuffled in and out, like demons, but only on a broader scale? Rather than being added and removed within the confines of a single story, they might be added and removed depending on the type of literature we are reading.

BOOKKEEPING

Up to this point we have introduced two parts of Charniak's DSP, demons and base routines. In this section we will introduce bookkeeping.

Again let us consider the situation when Jack had a ball, Bill a top, and they traded. When we say that Bill now has the ball, it implies that Jack no longer does. That is to say, we must somehow remove the fact that Jack has the ball from the data base. Actually we don't want to remove it, since we may be asked "Who had the ball before Bill did?" In this particular case we might be able to construct the answer from the fact that there was a trade, though we would also have to be on the lookout for "give" and even Jack losing the ball and Bill finding it. While this would be difficult, in other cases such an approach would be seemingly impossible. For example, suppose Janet wanted a doll, and she did several things to get it, but then decided that she didn't want a doll, but a paint set. If we were to erase the fact "Janet wants a doll" we would no longer be able to answer questions like "Why did Janet go to the toy store?" since we would have erased the reason. Charniak proposed to mark the assertions in some way to indicate that they have been updated. So, going back to Bill, Jack and the trade, we might have:

(15) (N1 HAVE JACK1 BALL1) TROUBLE: (NEG-UPDATE N4)
(N2 HAVE BILL1 TOP1) TROUBLE: (NEG-UPDATE N5)
(N3 TRADE BILL1 JACK1 TOP1 BALL1)
(N4 HAVE BILL1 BALL1)
(N5 HAVE JACK1 TOP1)

In proposing this scheme, Charniak was heavily influenced by the then in vogue MICROPLANNER programming language. It is interesting to speculate on how his ideas would have been different if CONNIVER were available to ease the handling of multiple contexts.

FACT FINDERS

Even deciding that one statement updates another requires special knowledge. Suppose we have:

(16) Jack was in the house. Sometime later he was at the store.

If we ask "Is Jack in the house?" we want to answer "No, he is at the store." But how is bookkeeping going to figure this out? There is a simple rule which says that (<state> A B) updates (<state> A C) where C is not the same as B. So (AT JACK FARM) would update (AT JACK NEW-YORK). But in (16) we can't simply look for Jack AT <someplace which is not the store>, since he is IN the house. To make things even worse, we could have:

(17) Jack was in the house. Sometime later he was in the kitchen.

To solve this problem we will add a theorem which knows about location. Such a theorem, we will call it AT-NOT-FF, might go:

(18) To establish that PERSON is not at location LOC

Find out where PERSON is, call it X,
If X = LOC , then theorem is false so return "No."
If X is part of LOC then return "No."

This handles, given appropriate
information, cases like (17).

If LOC is part of X, then try to find a different X.
Else return "Yes"

In (16) the bookkeeper would try to prove that Jack is not at the store, and it would succeed by using AT-NOT-FF and the statement that Jack is in the house. The bookkeeper would then mark the earlier statement as updated. Theorems like (18), called fact finders, will be indicated by -FF at the end of their names, such as AT-NOT-FF.

Like demons, fact finders have a pattern and a body. A particular fact finder is called when something else (either a demon, base routine or bookkeeping) wants to establish a goal which matches the pattern of the fact finder. This is different from demons which are called when we encounter a given fact. In MICROPLANNER fact finders are Consequent theorems, while demons, as we have already mentioned, are Antecedent theorems.

We introduced fact finders via bookkeeping. However, fact finders are needed elsewhere, and in fact, are a more secure part of the model than is bookkeeping. To take one example, we could have:

(19) Jack was in the house. Janet was at the park.

We would then ask the program "Is Jack at the park?" and AT-NOT-FF would supply the answer. To take another case, typically when a person offers a trade, we will want to make sure he owns the object being traded. One good rule of thumb is if you don't have an explicit "own" assertion, then the person who introduced the object into the story is the owner. Naturally, this rule is also a fact finder.

The basic idea behind fact finders is that they are used to establish facts which are comparatively unimportant, so that we do not want to assert them and hence have them

in the data base. So in (16) we do not want to assert "Jack is not in the house" as well as "Jack is at the store." In the same way we will have a fact finder which is able to derive the fact "<person> knowns <fact>" by asking such questions as "was the <person> there when <fact> was mentioned or took place?" Again, since this information is easily derivable, and not all that important, so there would seem to be no reason to include it explicitly in the data base.

McDERMOTTS BELIEF SYSTEMS

We now move to a review of McDermott's TOPLE which also deals with the issues of how general knowledge can and should interact with information in a particular story line.

TOPLE is a system that attempts to understand new sentences about a simple world by using a set of programs which embody a logical model of that world. It includes an understanding of the sorts of causal progressions and assumed presuppositions that are a large part of the unspoken, but important, contents of speech. Again it does not deal directly with English sentences, but interprets simple semantic structures such as might be produced by a natural-language parser.

The world that TOPLE lives in is almost as simple as the BLOCKS world of Winograd's (TR-235) language understander, but involves more complex notions of process and action. The world is that of monkey and experimenter in a single room. The monkey (named Spiro) is capable of several actions, and driven by simple motives of hunger, thirst, and curiosity. Wolfgang, the experimenter, can do similar things, and his motivation is assumed to be seeing what Spiro will do in response to his actions. The contents and characteristics of the room are not as capturable as in the BLOCKS world; the program tries to make allowances for sloppiness and incompleteness in describing the layout of the room. The program listens to a present-tense account of goings-on in this room, and attempts to understand why things happen, and what can be expected as the story progresses. It tells us at the end of every sentence what new assertions it has assumed as a result of hearing it.

Although, in some sense, question-answering ability is a measure of understanding, TOPLE does not answer questions. The user must interrogate the data base "by hand," or keep track of all the assumptions the program has made.

TOPLE operates as follows. Each chunk of the input language causes a tree of hypothetical worlds to be created and pondered, one for each interpretation of it. This tree is built by a function named PLAUSIBLE? which calls methods to turn input formulae into assertions. Assertions are modeled as items with property-list structures that relate them to other beliefs in ways TOPLE's subsystems can utilize. Such methods and their subroutines attempt to fit what they are told into what they know with as few conflicts as possible. If a formula cannot be understood by itself, the tree of possible worlds is preserved while succeeding formulae are read, which hopefully resolves the problems and points to a single interpretation.

The overall static structure of TOPLE is illustrated by figure 1.

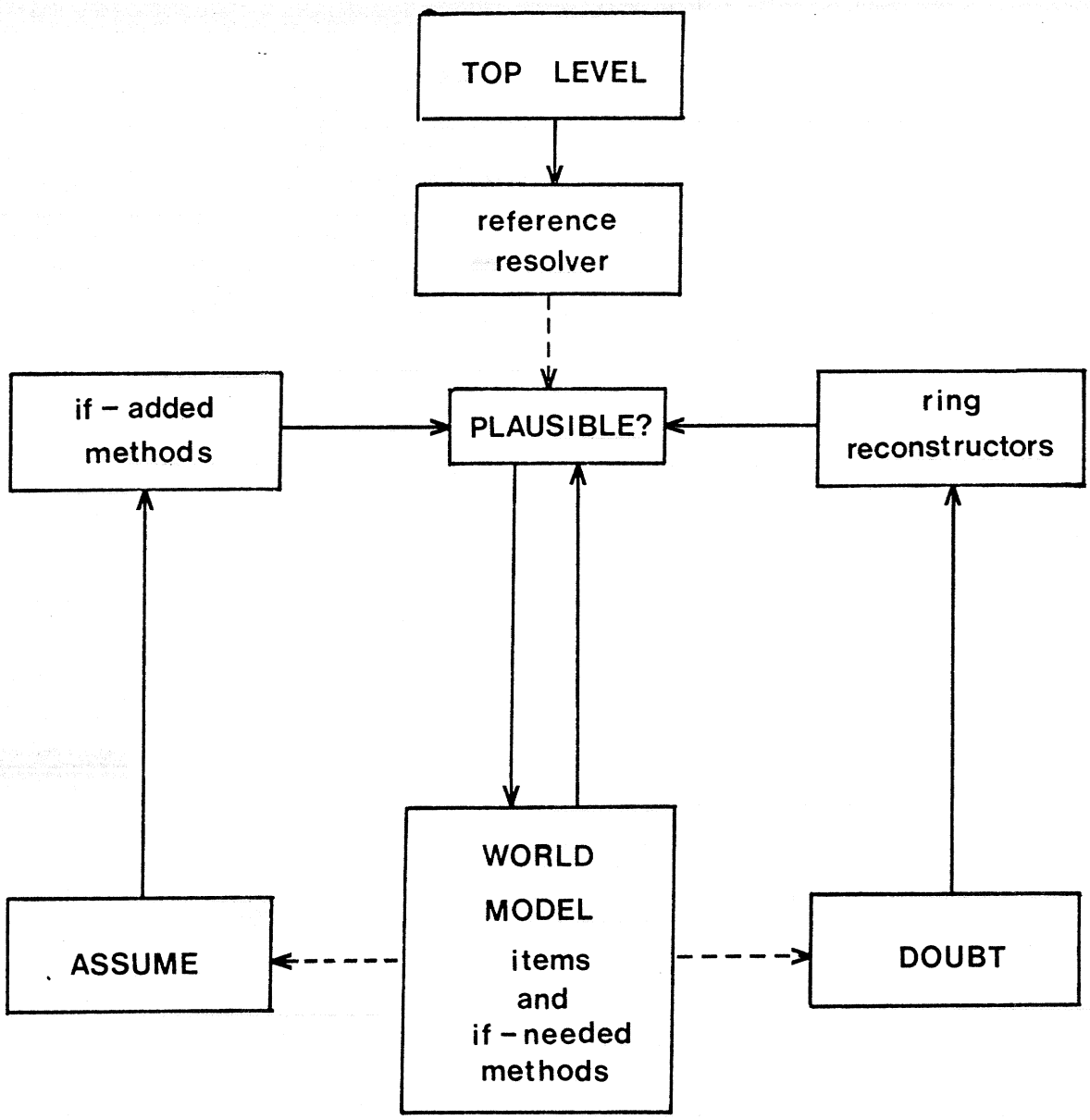

TOP LEVEL
reference resolver
if – added methods
PLAUSIBLE?
ring reconstructors
WORLD MODEL
items and if – needed methods
ASSUME
DOUBT
means "calls"
means "calls to change model" (BELIEVE+ mode)

FIGURE 1

The center of the system is marked "WORLD MODEL;" this is a set of CONNIVER if-needed methods and items which embody TOPLE's knowledge of the world. They are called by PLAUSIBLE?, either to deduce consequences of already-known items, or to make changes to the set of items. The methods call the functions PLAUSIBLE?, ASSUME, and DOUBT to effect needed dependent changes. The boxes marked "if-added methods" and "ring reconstructors" contain programs with additional knowledge about making additions to or deletions from the world model.

As an example, consider the following "protocol" of a conversation with TOPLE. In each case, there is an English sentence, which you are to imagine is actually in a more formal-looking input language. There is also a description of the actions and assumptions the machine takes. (The bracketed numbers identify such actions and assumptions. They are not generated by TOPLE.)

"The banana is under the table, by the ball."

The program treats this as two statements: that the banana is under the table, and that it is by the ball. The program finds the references first and attempts to resolve them. In fact, it knows of no banana or table, so its response is to let BAN1 and TAB1 name two new objects. The program then files away:

(AT BAN1 (PLACE TAB1 UNDER)) [1]
(AT TAB1 (PLACE FLOOR1 ON)) [2]

as its response. The latter statement is an assumption made by the spatial reasoning routines when they hear of something without knowing its supports.

Now it attacks "..., by the ball." This reference is resolved as the others were, and

(AT BAN1 (PLACE BALL1 BY)) [3]
(AT BALL1 (PLACE TAB1 UNDER)) [4]

become its new "visualization" of the situation. This system of beliefs corresponds to accepting figure 2(a) rather than figure 2(b), or some representation of indifference or uncertainty between them. The motivation for forcing a choice is important. TOPLE tries to visualize concretely a situation surrounding what you tell it, rather than wait for a question or other problem that would force it to make up its mind. This saves on

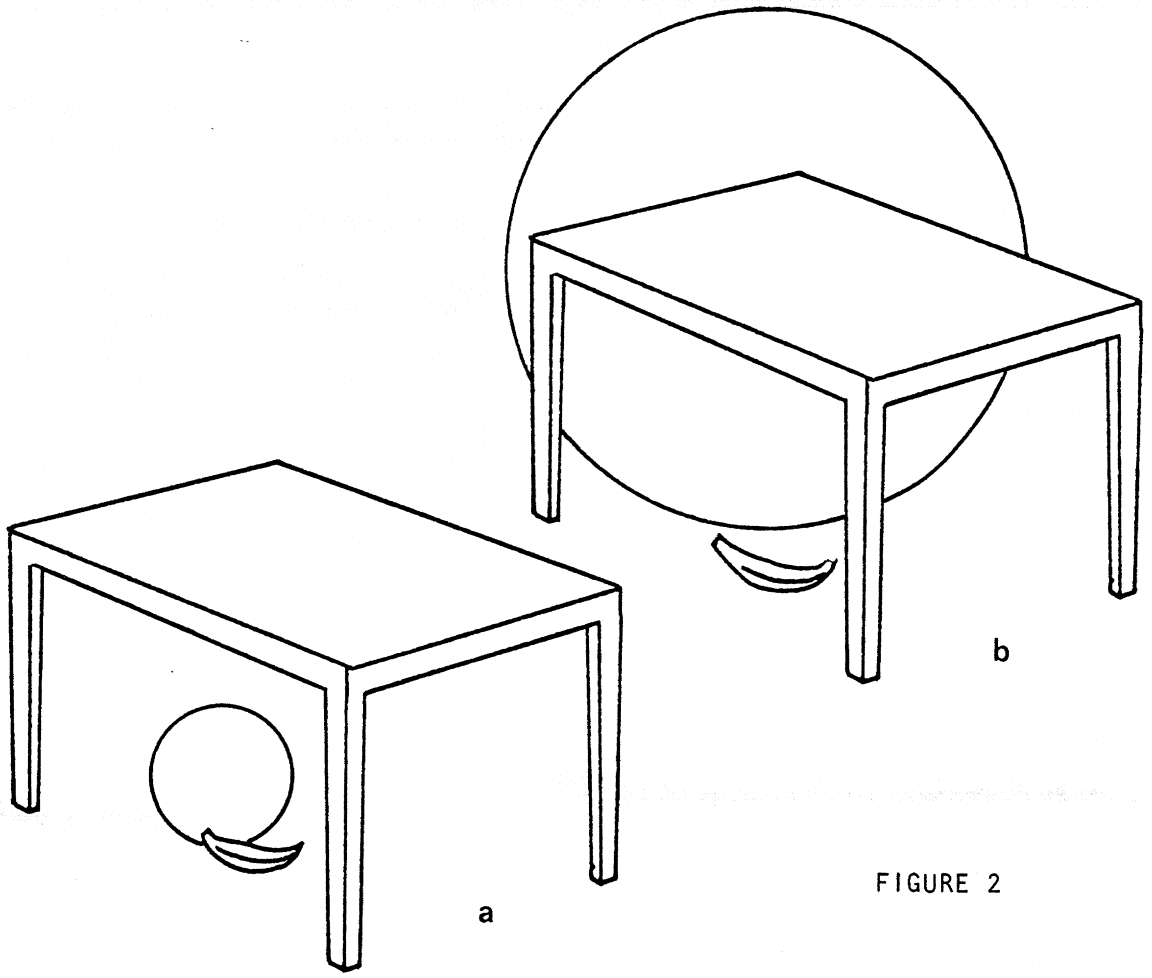


FIGURE 2

computation, but makes it necessary occasionally to undo bugged reasoning.

In this case, the choice is based on its belief that the object sizes of figure 2(a) are more likely.

In the sentence, "The monkey goes over to the table and picks up the bannana," the two clauses are not unambiguously related, as they were before. "And" is a word which has (among other meanings) a simultaneous reading, in which the clauses on either side of it are both true at once, and a more common reading in which they are temporally (and possibly causally) related. The first embodies the meaning of "and" in "The monkey was gone and its mate was scared"; the second, that in, "He stuck his finger in and wiggled it around."

When the program encounters such an ambiguity, it is capable of splitting into two or more parallel investigations to decide which interpretation is most likely. However, in the case of "and," there is a nicer way to resolve the ambiguity, at least in simple cases. This is to hand the two clauses of the conjunction to the plausibility checkers in order,

and allow the second one to be believed in whatever situation the first one leaves. For simple actions, this reduces to assuming "and" means "first one, then the other;" for conditions, that it means "both now."

This way of interpreting "and" is an instance of TOPLE's general inclination not to investigate unlikely alternatives at all unless some difficulty in assimilating the first choice arises. In this case, the reference to MONK1 is resolved as a reference to Spiro, the only monkey the program knows about. Similarly, TAB2 becomes TAB1, the same table as before.

Consequently, TOPLE must accept (GO SPIRO (PLACE TAB1)). To do so, it must establish that the monkey can and wants to perform this action. It does not doubt that Spiro can go to the table, but is uncertain whether he does it out of interest in the table, the ball, or the banana that are in that spot. There is no reason it can imagine for Spiro being interested in tables, but balls are fun to play with and bananas can be eaten. The uncertainty between these two is reflected in the fact that it generates two alternative versions of what Spiro wants and what he will do:

In situation 0:
    (WANT SPIRO (HOLD SPIRO BAN1)) [5]
    (WANT SPIRO (EAT SPIRO BAN1)) [6]
    (HUNGRY SPIRO) [7]
    (GO SPIRO (PLACE TAB1)) [8]

In situation 1:
    (AT SPIRO (PLACE TAB1)) [9]
    (PICK-UP SPIRO BAN1) [10]

In situation 2:
    (HOLD SPIRO BAN1) [11]

In situation 3:
    (EAT SPIRO BAN1) [12]

In situation 4:
    BAN1 ceases to exist [13]

OR

In situation 0:

(WANT SPIRO (HOLD SPIRO BALL1)) [14]
(WANT SPIRO (PLAY SPIRO BALL1)) [15]
(GO SPIRO (PLACE TAB1)) [16]

In situation 5:

(AT SPIRO (PLACE TAB1)) [17]

In situation 6:

(PLAY SPIRO BALL1) [18]
(PICK-UP SPIRO BALL1) [19]

In situation 7:

(HOLD SPIRO BALL1) [20]

Each situation referred to here is a separate state of the world, which represents a state incrementally different from its predecessor as a result of some action. These two sequences of situations are mutually exclusive; situations 1 and 5 are both possible successors to state 0. (Notice that there are two different versions of state 0 which the system keeps in mind.) The sequences are generated by simulating Spiro with the goals corresponding to his respective WANTs. "(HUNGRY SPIRO) in situation 0" or "(WANT SPIRO (PLAY SPIRO BALL1)) in situation 0" are the crucial assumptions shown--they were introduced to explain the monkey's behavior.

The program is now halfway through the sentence, with two completely independent world-views to choose from. The first seems most likely, since only hunger need be assumed, but the program suspends judgment.

Now it accepts the sequential interpretation of "and" and attempts to believe (PICK-UP SPIRO BAN1) (it resolves the reference in favor of a banana it knows rather than a new one) in the state produced by the previous action. It fits best with the supposition that the monkey was interested in the banana, since it confirms the sequence of [5]-[13]. Notice that the program has only been told about events through assumption [11]; the beliefs about situations 3 and 4 are predictions. Beliefs [14]-[20] have been collected as garbage and are no longer accessible.

Now TOPLE informs its interlocutor of its assumptions and awaits the next sentence.

"He eats it."

"He" (treated as "the male referred to") is obviously Spiro. "It" could be several things. The top level program keeps track of referred-to objects, and considers each in reverse chronological order of reference. So, the first object considered is BAN1. (EAT SPIRO BAN1) is so easy to believe (since it was predicted), that the other alternatives are not even considered. States 3 and 4 are accepted as reality. At this point the banana ceases to exist.

This simple monologue suffices to illustrate many of the important features of TOPLE. Its main characteristic is that it uses what it hears in a constructive way to visualize what is being talked about. It even goes so far as to have opinions about what the future will look like, and uses them to disambiguate later parts of the story it is hearing. Thus TOPLE holds many beliefs which are not in any sense deduced from what it has heard, but are guessed, so it must keep track of its reasons for believing what it believes. If later information conflicts with its guesses, it must find new hypotheses which are good at accounting for both the old and the new data.

This function is accomplished by the belief ring device. Assumptions (such as [2]) are bound up with the previous beliefs that made them necessary (in that case, that TAB1 is a physical object), plus a program that is capable of debugging this structure if it is ever challenged. The data that made the assumptions necessary are called primary elements of the ring; the assumptions themselves are secondary elements; and the program is called the ring reconstructor. In the case of [2], the primary element is merely that TAB1 is a physical object; the secondary element is (AT TAB1 (PLACE FLOOR1 ON)); and the reconstructor is a program that is called when someone doubts that the table is on the floor, which asks, "What does support the table, then?" For example, if the next sentence were, "The table is on a platform," the routine in charge of believing such spatial assertions would have to call DOUBT (cf. figure 1) to remove (AT TAB1 (PLACE FLOOR1 ON)). DOUBT would call the ring reconstructor, which would insist that something be assumed to be holding the table up, and would be happy when it saw the newly proposed belief that it be a platform.

PRATT'S LINGUISTICS LANGUAGE

So far in this section we have seen three studies directed at unravelling the issue of how world knowledge interacts with language understanding. Pratt has addressed the relatively applied problem of supplying a general-purpose system for writing natural language "front ends."

NOTATIONS FOR DESCRIBING ENGLISH

At the risk of being animistic, let us think of the problem of writing text-oriented natural language programs as essentially the problem of describing English to the computer. Thanks to such programs as Winograd's SHRDLU, we know in principle how to process English - what remains to be dealt with is the bulk of English. We should therefore take advantage of this progress by factoring out the programming aspects of the problem, which can constitute a fatal distraction in a project of this magnitude, and consider more precisely what English is, with a view to passing this information on to a program such as SHRDLU. With this point of view, we can consider ourselves not computational linguists but just plain linguists, albeit with a somewhat unimaginative and intolerant audience.

Let us now occupy ourselves with the issues an applied linguist might want to address, given that he has to set down his impressions of English. One would like firstly to minimize the effort required to describe phenomena in natural languages, and secondly to avoid having to duplicate in essence what someone else went to a lot of trouble to do. The former is an issue because of the complexity of natural languages, the latter because despite years of research almost every new natural language program starts from scratch, drawing little if any material from previous efforts.

There are at least two approaches one might take to the complexity issue. One might try to reduce the complexity by generalizing special cases as far as is consistent with one's particular goals. This is understandably very popular. In addition, one might try to improve on available notational facilities for describing languages by taking into account those properties of natural languages that are awkward or impossible to describe with the extant facilities. Those taking this approach have been responsible for introducing quite a variety of notation into the field, including phrase-structure grammars, transformational grammars, dependency grammars, systemic grammars, conceptual dependency grammars, transition networks and case grammars, to name some of the more prominent notations.

At least two approaches are possible for the duplication issue too. One may standardize the notation for descriptions, a dictatorial approach meant to allow any contribution to be incorporated into any other program. The danger here is that one might be stuck with an inadequate notation. Alternatively, one may write one's descriptions in a way that makes them readily extensible. This is a relatively poorly understood area, but is well worth pursuing at present. Had SHRDLU been written in a way more conducive to ready extension, a very impressive program might have resulted, considering the effort that others have put into it since Winograd built the original version.

We have been looking at these approaches in the context of "small" language processors, that is, programs that work with a fairly small fragment of English, either question-answering systems or translators into some foreign language. Our intention is to use the experience gained from working with this variety of programs to help understand how best to deal with the notational issues. The examples in the accompanying output from various programs we have written should give the reader a feeling for the sort of programs we have considered and the size of the fragment of English we have been working with:

A "DEEP STRUCTURE" ANALYZER:

```
NONE OF MY FRIENDS WERE EATEN BY A LION .
ASSERTION:
        FALSE
                ACTOR: LION
                ACT: EAT
                OBJECT: FRIENDS
                        PORTION: SOME
                        OWNER: I
```

THE AUTHORS OF NONE OF THESE BOOKS ARE FRIENDS OF PETER .
ASSERTION:
    FALSE
        SUBJECT: AUTHORS
            SPECIFY: THE
            AUTHORS OF: BOOKS
                PORTION: SOME
                PLACE: HERE
        BE: FRIENDS
            FRIENDS OF: PETER

IF A MAN CAN EAT A DOG A HORSE CAN EAT A SMALL CAT .
ASSERTION:
        ACTOR: HORSE
        ACT: EAT
        OBJECT: CAT
            SMALLNESS: SMALL
        IMMEDIACY: POTENTIAL
        CONDITION:
                ACTOR: MAN
                ACT: EAT
                OBJECT: DOG
                IMMEDIACY: POTENTIAL

A MODEST GERMAN TRANSLATOR:

SHE LOVES THE MEN WHO CAN GIVE HER HOUSE FLOWERS.
   SIE LIEBT DIE MAENNER DIE IHREM HAUS BLUMEN GEBEN KOENNEN
THE APPLES THAT THE HORSE EATS WILL BE BETTER TOMORROW.
   DIE AEPFEL DIE DAS PFERD ISST WERDEN BESSER MORGAN SEIN
THAT THE DOG THAT EATS FLOWERS CAN COME IS BAD.
   DASS DER HUND DER BLUMEN ISST KOMMEN KANN IST SCHLECHT
MEN THAT ARE OLD CAN SLEEP BETTER.
   MAENNER DIE ALT SIND KOENNEN BESSER SCHLAFEN
HE SLEEPS BETTER WHILE HIS DOGS EAT.
   ER SCHLAEFT BESSER WAHREND SEINE HUNDE ESSEN

HE LIVES WITH HER BECAUSE WHEN HE EATS APPLES SHE EATS WITH HIM.
ER LEBT MIT IHR WEIL WENN ER AEPFEL ISST SIE MIT IHM ISST
SHE SLEEPS WITH HIM BECAUSE SHE LOVES HIS CAT.
SIE SCHLAEFT MIT IHM WEIL SIE SEINE KATZE LIEBT
HE IS IN THE HOUSE.
ER IST IN DEM HAUS
HE HITS THE BALL TO THE DOG.
ER SCHLAEGT DAS BALL ZU DEM HUND
THE CAT IS IN THE TREE BY THE HOUSE.
DIE KATZE IST IN DEM BAUM NEBEN DEM HAUS
THE STUDENT WILL SLEEP BETTER WHEN HE UNDERSTANDS THE IDEA.
DER STUDENT WIRD BESSER SCHLAFEN WENN ER DIE IDEE VERSTEHT
WHILE THE FLOWER IS OLD THE APPLE IS YOUNG.
WAHREND DIE BLUME ALT IST IST DER APFEL JUNG
SHE WILL GIVE HIM A FLOWER IF HE EATS THE APPLE TODAY.
SIE WIRD IHM EINE BLUME GEBEN WENN ER DEN APFEL HEUTE ISST
THE HORSE THAT EATS BAD APPLES IS A SICK HORSE.
DAS PFERD DAS SCHLECHTE AEPFEL ISST IST EIN KRANKES PFERD

A COMPREHENSION PROGRAM:

JACK BUILT A RED HOUSE. HE LOVED RATS. MORTIMER WAS JACK 'S RAT. HE LIVED IN THE HOUSE THAT JACK BUILT. JILL BUILT A BLUE HOUSE. SHE LOVED CATS. SPOT WAS A CAT WHO ATE MORTIMER. JACK HATED JILL BECAUSE SPOT ATE MORTIMER. FIDO WAS A BROWN DOG. HE CHASED SPOT. SPIKE WAS A BLACK DOG. HE CHASED JILL.

WHAT IS THE COLOR OF THE DOG THAT CHASED THE CAT.
BROWN
DID JACK LOVE MORTIMER.
YES
WHO ATE MORTIMER.
SPOT
WHAT IS THE COLOR OF THE HOUSE IN WHICH THE RAT THAT THE CAT ATE LIVED.
RED
WHY DID JACK HATE JILL.
BECAUSE SPOT EAT MORTIMER
WHO DID THE BLACK DOG CHASE.
JILL
DID JILL LOVE ANY ANIMALS.
YES

TERMINOLOGY

The system assumes two main phases, cognitive and generative. The cognitive phase is parsing, in which the input is preprocessed until it is in a form convenient for operation on by the generative phase, which then produces the translation as output. This dichotomy does not require that one phase run to completion before the other can start.

By syntax is meant all aspects of the source language involved in the cognitive phase, including such things as phrase structure rules and semantic markers. Semantics refers to what is involved in going from the source language (after the syntactic preprocessing) to the target language during the generative phase. Pragmatics means knowledge about the universe of discourse, and the local context, that may be consulted by both the cognitive and generative phases as they make decisions.

DESIGN PHILOSOPHY

There is not one philosophy in LINGOL but three, each tuned to the requirements of the three concepts defined above. In current version of LINGOL, the philosophies are roughly as follows.

SYNTAX

It is necessary to consider syntax since the cognitive phase's output is the generative phase's input. LINGOL is meant to be a practical system suitable for export and immediate use by practising computational linguists. The technology for phrase structure is far advanced over any other technology, and every successful program for the past eight years or so has used it. Also, it is fairly easy to convert a phrase structure system to a syntactic structure system, by tagging each phrase with the corresponding governing word together with pointers to the dependent phrases (and hence words). For these reasons, the decisions was made to use phrase structure as the output of the cognitive phase, leaving the other representations as projects to be experimented with in the future.

Given that LINGOL is based on phrase structure, the next issue is that of the user's language for describing how that phrase-structure is to be built. The two criteria here are expressive power and ease of use. For our first interation of LINGOL, since we were more interested in rapidly developing the semantics technology, we opted to sacrifice expressive power for ease of use if necessary. This corresponds in a way to Woods (1967) and Charniak (TR-266) assuming the existence of some sort of parser and continuing from there.

The user's language for the cognitive component was therefore chosen to be context-free rules, since these are very easy to write. They have exactly the same expressive capacity as Wood's transition networks (1969). Moreover, just as Woods extended the capacity of his networks by allowing the user to specify operations on registers, so do we permit the user to supply code to give hints to the parser whenever it is about to apply a rule. This code has access to the part of the tree built so far by the parser and relevant to the rule in question, and also to the user's data base, or pragmatics. The form of the hint is a shout of approval or disapproval. So far, however, none of the programs written in LINGOL have made more than trivial use of this feature, in sharp contrast to the use made of the features in the semantics stage.

With respect to the actual parser used, the syntax philosophy is that that parser should be transparent to the user, to within there representation of the parts of the tree to which the user's code has access during the cognitive phase. This philosophy has enabled us to run without alteration each of a number of different LINGOL programs in conjunction with various parsing algorithms.

SEMANTICS

In programming his semantics, the user should be able to work without the distracting detail of parsing, tree representation, and ambiguity. The point of identifying the cognitive and generative phases is to isolate these issues logically in order to achieve this division of labor. Whether writing an English-to-French translation program or a question answering system, there are many details to worry about that have absolutely no relevance to the cognitive phase; the myriad idiosyncrasies of French grammar and style, for example. The cognitive phase can ignore most details of style, and many details of grammar.

In taking this point of view, we are following a different philosophy from that of Winograd, who makes use of strong interaction between the syntax and semantics components, which is one of the more notable features of his program. However, the result has been to produce a program whose details are lost in the richness of this interaction, and Winograd himself can barely understand the code he has written after being away from it for a while. For the moment let us be willing to sacrifice whatever additional power this approach has to offer for the sake of being able to write clean, modular, transparent semantic code. However, we do not believe that in order to restore this power we need to restore this interaction. Instead, we plan to rely eventually on strong interaction between syntax and pragmatics, leaving semantics as the cognition-independent arena. This is not just passing the buck; since we see semantics as being more complex than syntax, we are trying to divide the work-load more evenly to keep all modules reasonably small. The interaction of syntax and pragmatics is under study by others in our group, and is reported in previous sections.

The issue now is simply, how does one write programs that operate on trees (the output of LINGOL'S cognitive phase)? This issue has been addressed by many during the past ten years, and the discipline of syntax directed translation has gradually emerged. An early syntax directed translator is that of Warshall and Shapiro (1964). They used the tree-walk paradigm, in which the semantics consists of programs that tell a pointer to move up, down or across the tree and occasionally output information. Floyd (conversation) has commented that the technique was much too clumsy for practical applications when compared with techniques that tied the semantics to the syntax rather than to the output of the syntax.

Some theoretical work has been done on syntax-directed translation, notably by Lewis and Stearns (1968), Knuth (1968), and Aho and Ullman (1972). Knuth's paper is of interest in that it deals with the problem of passing information up and down a tree,

using the notions of inherited (from above) and synthesized (from below) attributes. All of these studies suffer, from the computational linguist's point of view, in that they deal with the microcosm of computer source and target languages, in which the former can be made a compromise between the user's needs and the syntax-directed technology, and the latter is a relatively well-defined, reference-poor language when compared with, say, French.

Knuth's inherited and synthesized attributes come closest to meeting our needs. The problem with these attributes lies with his mechanism for moving them around a tree. Every node through which information is passed must make explicit provision for forwarding it, even if it is irrelevant to that node.

For example, consider:

No mother of such twins has time to relax.

The mother of no such twins has time to relax.

The mother of such twins does not have time to relax.

The mother of such twins has no time to relax.

In each case, what is being negated is the whole sentence, yet the negation marker can be almost anywhere in the sentence. This implies that a large number of rules will have to make provision for passing a negation marker up the tree. This problem can be circumvented by using global variables instead of Knuth's attributes. All that is needed is for the negation marker to set a negation variable, and for the semantics at the syntactic clause level to read it.

However, consider the following:

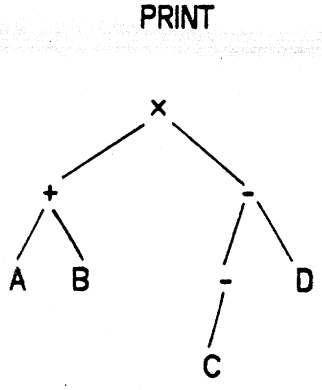
The mother who has no twins has time to relax.

This sentence makes a positive claim (as distinct from the negative one of the previous example) in that it says that there actually are people who do have the time to relax, namely those mothers who have no twins. (Moreover, it does not explicitly say what happens to mothers of twins.) This seems to be a situation where synthesized attributes outperform global variables, since the rule of the relative clause level can simply refuse to pass on the negation marker.

Negation is not the only such troublemaker. Arranging subject-verb, adjective-noun and determiner-noun agreement also requires passing information around the tree, especially when translating into French, where word-for-word translation does not necessarily result in correct agreement. Again, having more than one clause makes difficult the use of global variables, particularly when a plural relative clause is separating a singular subject from its verb. The mechanism we want here is that of the local variable, whose scope is the clause with which it is associated. With many clauses

we will associate many more local variables corresponding to the various markers and other messages that each clause may want. Similarly, we will associate other local variables with noun phrases, to achieve adjective-noun and determiner-noun agreement. In the case of the subject, some of these markers (person and number, but not gender) must be shared with the clause as well, to ensure subject-verb agreement, but we do not want the clause to share the object's variables. Also, a relative clause such as "who sleeps" needs the same information from its governor as does the principal clause. Moreover, we will want to pass not only markers, but also word specific programs written at the dictionary level. (Winograd makes use of this technique for putting the right programs in the right places.) The implementation of local variables must be able to handle these combinations.

All of this is now implemented in the style of the program paradigm. The program paradigm says that the surface structure tree is a program. At each node of the tree there is a function, and the subtrees of that node are the arguments of that function. For example, if we have a tree labelled



This corresponds to the program "(PRINT (A+B) × -(-C+D))".

Since LISP has a mechanism for local variables (two, in fact -- PROG variables and LAMBDA variables), by adopting the program paradigm we automatically get local variables. Moreover, because we can write the code for each function separately, we attain a very high level of modularity, which we have found pays off handsomely when one tries to add new rules to an already operational LINGOL program.

The mechanism we use for running these programs differs slightly from LISP's usual EVAL operator. The main difference is that it evaluates the function at each node first, giving the function the responsibility for evaluating subtrees at its leisure, and controlling the scopes of variables for different subtrees.

PRAGMATICS

The first three studies of this section treat aspects of pragmatics in some detail. LINGOL provides no explicit facilities at present for dealing with world knowledge, although a large LINGOL program is currently under development that may change this situation. There is of course the implicit facility provided by the cognitive component, but the user is really on his own as far as the bulk of the programming effort is concerned. Pratt hopes to be able to draw on the work of those in the group studying representation and inference, to the extent that these studies introduce sufficient regularity into the area to warrant supplying generally useful facilities.

SOME RELEVANT READING

Bobrow, D. G. and J. B. Frazer. "An Augmented State Transition Network Analysis Procedure." Proceedings of IJCAI (1969), 557-568.

Farber, D. J., R. E. Griswold and I. P. Polansky. "SNOBOL, A String Manipulation Language." Journal of the ACM, 11, No. 2 (1964), 21-30.

Fillmore, C. J. "The Case for Case." Universals in Linguistics Theory. Ed. Bach and Harms. Chicago: Holt, Rinehart and Winston, 1968.

Green, P. F., A. K. Wolf, C. Chomsky and K. Laugherty. "BASEBALL: An Automatic Question Answer." Computers and Thought. Ed. E. A. Feldman and J. Feigenbaum. New York: McGraw Hill, 1963.

Hays, D. G. "Dependency Theory: A Formalism and Some Observations." Language, 40, No. 4 (1964), 511-525.

Katz, J. J. and J. A. Fodor. "The Structure of a Semantic Theory." The Structure of Language. Ed. J. A. Fodor and J. J. Katz. Englewood-Cliffs, N.J.: Prentice-Hall, 1964.

Kilma, E. "Negation in English." The Structure of Language. Ed. J. A. Fodor and J. J. Katz. Englewood-Cliffs, N.J.: Prentice-Hall, 1964.

Knuth, D. E. "Semantics of Context-Free Languages." Math Systems Theory, 2 (1968), 127-145.

Lewis, P. M. and R. E. Stearns. "Syntax Directed Transaction." Journal of the ACM, 15, No. 3 (1968), 465-488.

Narasimhan, R. "Computer Simulation of Natural Language Behavior." Invited paper, Conference on Picture Proc. Mach., Canberra, Australia, 1969.

Schan, R, L. Tesler and S. Weber. "Spinoza II -- Conceptual Case Based Natural Language Analysis." Memo AI-109, Publications of Artificial Intelligence Laboratory, Stanford University, 1970.

Simons, R. F., S. Klein and K. McConlogue. "Indexing and Dependency Logic for Answering English Questions." Amer. Doc. 15, No. 3 (1964), 196.

Thorne, J., P. Bratley and H. Dewar. "The Syntactic Analysis of English by Machine." Machine Intelligence 3. Ed. D. Michie. Edinburgh: Edinburgh University Press, 1968.

Warshall, S. and R. M. Shapiro. "A General Purpose Table Driven Compiler." Proc. AFIPS SJCC, 25 (1964), 59-65.

Woods, W. A. "Semantics for a Question Answering System." Report No. NSF-19, Publications of the Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts, 1967.

Woods, W. A. "Augmented Transition Networks for Natural Language Analysis." Report No. CS-1 for the NSF, Publications of the Aiken Computation Laboratory, Harvard University, Cambridge, Masachusetts, 1969.

Yngve, V. H. "COMIT." Communications of the ACM, 6, No. 3 (1963), 83-84.

4 LEARNING AND DEBUGGING

When exposed to the work of artifificial intelligence for the first time, people are often concerned about whether a program can learn. This section describes the capabilities of some programs that do indeed learn new things from experience. Collectively they shed some light on such issues as the following:

1. Is learning a special kind of problem solving?

2. Is learning physical things substantially different from learning how to do things?

3. What knowledge is prerequisite to learning? How should that knowledge be organized into chunks?

4. How much knowledge is involved? How many facts? How many procedures?

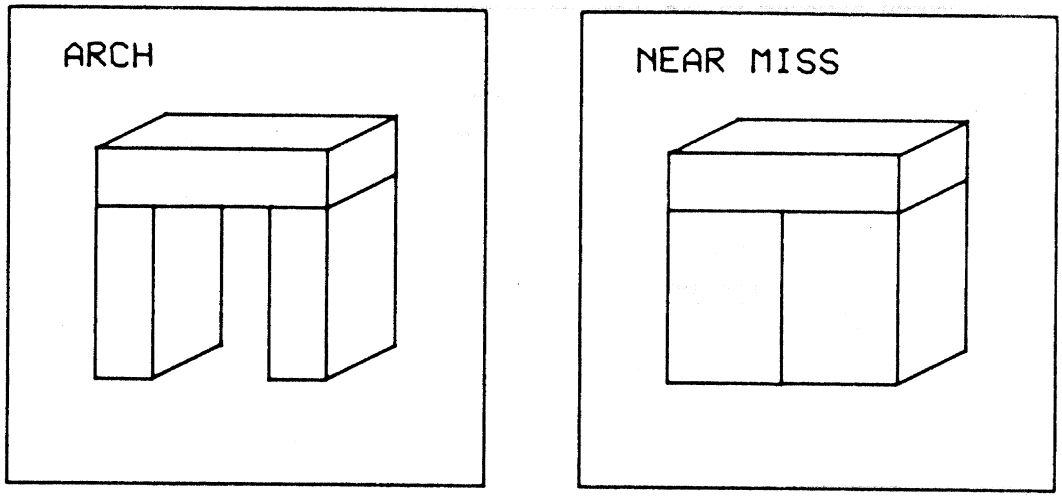
WINSTON'S STRUCTURE LEARNING SYSTEM

To begin we briefly review Winston's work on learning arches and other simple physical structures in the blocks world (TR-231). Figure 1 illustrates a series of samples shown to Winston's system in teaching it the concept of ARCH. The first sample is an example whose description is used as the first in an evolving series of models. The second sample causes refinement of the model by conveying the idea that there must be a hole. The third sample informs the system that the top object may be a wedge as well as a brick. And the last sample brings out the fact that the support relations are essential.

The process by which all this is done consists of the following steps for each sample:

1. The sample is analyzed so as to create a symbolic description of the sort shown in figure 2, consisting of the nodes and relations of a network.

2. The network description of the sample is compared with the network description of the model as evolved so far. Decisions are made about what difference or differences are central to understanding whether the



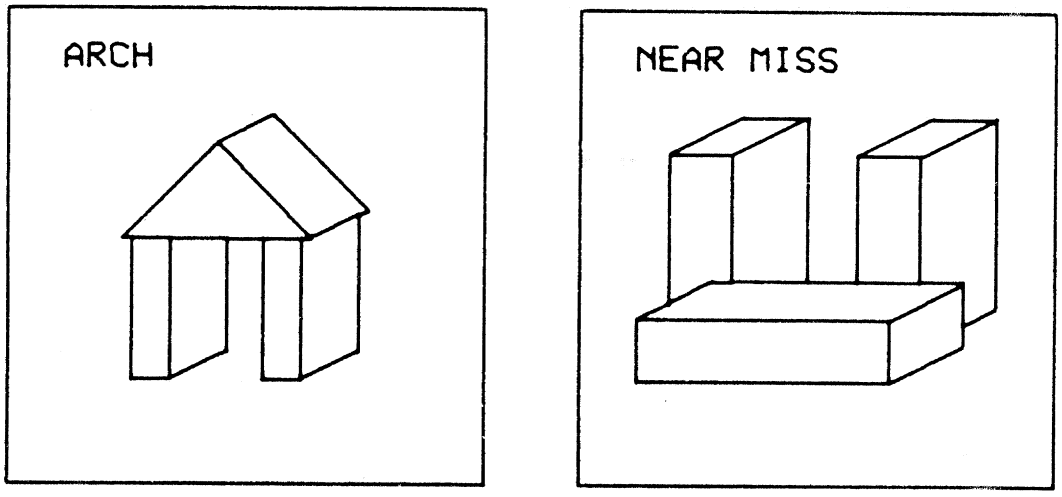


FIGURE 1

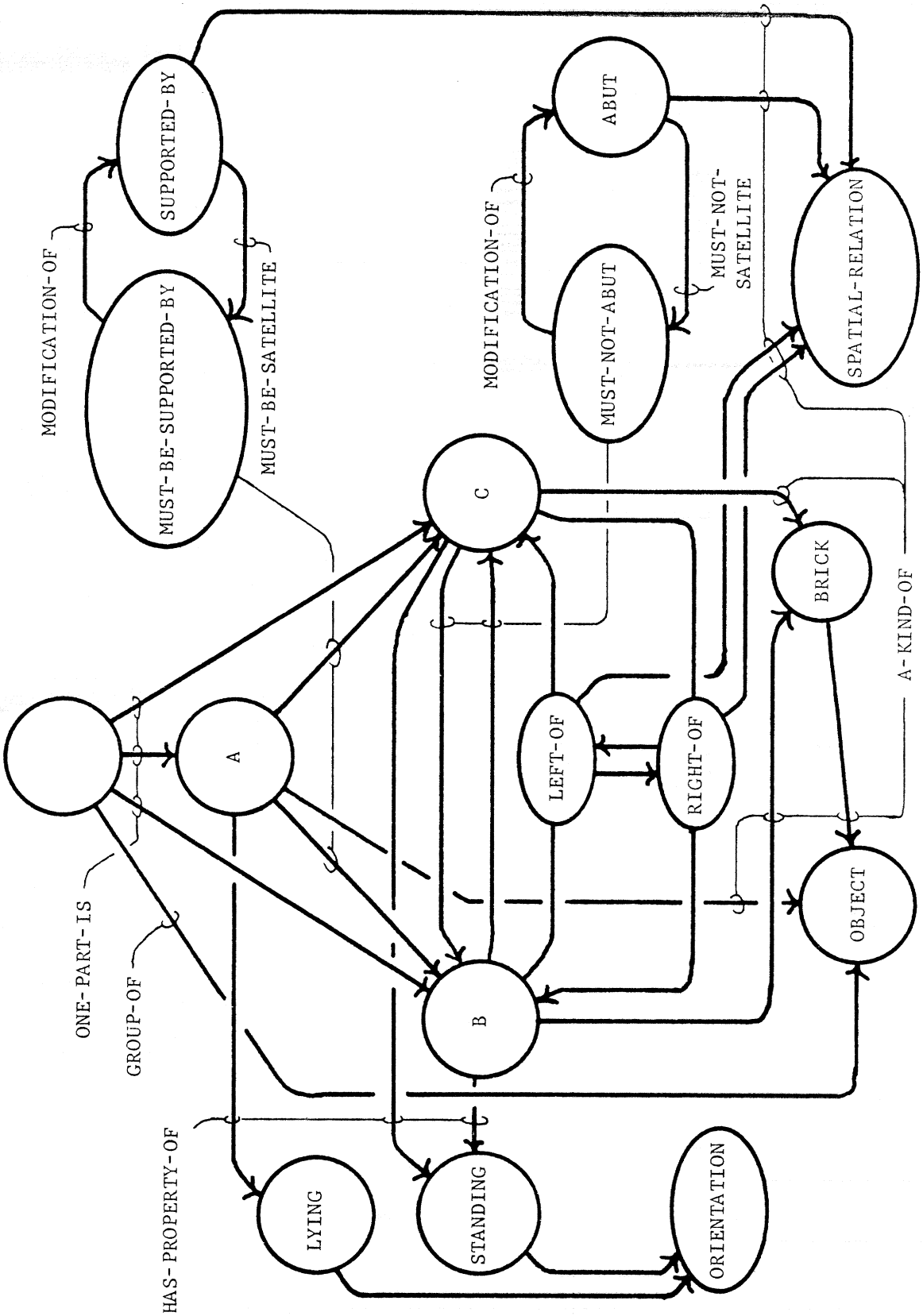

MODIFICATION-OF
SUPPORTED-BY
MUST-BE-SUPPORTED-BY
MUST-BE-SATELLITE
MODIFICATION-OF
ABUT
MUST-NOT-ABUT
MUST-NOT-SATELLITE
SPATIAL-RELATION
C
A
B
LEFT-OF
RIGHT-OF
BRICK
A-KIND-OF
OBJECT
ONE-PART-IS
GROUP-OF
HAS-PROPERTY-OF
LYING
STANDING
ORIENTATION

FIGURE 2

sample is an instance of the concept.

3. The central differences are used to decide what refinement to make in the model's network description. In one case a TOUCH pointer might be changed to MUST-NOT-TOUCH or in another, a SUPPORTS pointer might be changed to MUST-SUPPORT.

The successful results of experiments with programs built around these steps lead to several conclusions:

1. It is important to use samples which are "near-misses" or non-examples that differ only slightly from acceptable examples.

2. Good teaching is important inasmuch as samples which are too different from what has been seen can swamp the system with differences and lead to confusion.

3. Much of the knowledge required for learning structures can be placed in a library relating observed differences to proposed actions on the model network. The size of this library is not large -- it contains only about two dozen entries.

These conclusions in turn open up exciting new questions. Can the analysis and learning of structural form be extended to include description of function? Can similar ideas work both in the learning of structures and in the debugging of procedures? What sort of knowledge libraries are needed to learn procedures? How large are they?

The next sections shed considerable light on these question.

FREILING'S BLOCKS WORLD EPISTEMOLOGY

Winston's system dealt with objects described only in terms of their structure. Such descriptions have many inherent limitations:

(a) They do not provide for recognition of objects on the basis of less concrete criteria, say the uses to which they may be put.

(b) They do not permit generalization to deal with classes which may be different structurally but similar in other respects. For example, the class of possible supports for a television set includes tables and shelves, which share little in the way of common structural properties.

(c) In many practical problems, it is of great advantage to have many ways of describing a particular object. Limitation to one specific mode of description severely restricts the class of problems which may be tackled.

These limitations suggest that we search for other ways to describe objects aside from pure structural form and attempt to understand the relationships between different ways of describing objects.

FUNCTION

Function is the obvious alternative to form. There are several reasons why:

(a) It may be possible to construct simple representations of the functions of objects in the blocks world in terms of simple concepts of motion and areas of unoccupied space.

(b) Many structures in the blocks world have real-world counterparts which are classified in actuality on the basis of function. An arch is principally something we can pass through. A table is principally a structure we can put things on.

(c) There are many relationships between the form and function of objects. Since the possible functions of a class of objects are generally much simpler to enumerate (assuming we have the proper tools!) than the possible structures, functional

description enjoys the advantages of more concise forms of representation with a corresponding increase in our ability to manipulate overall descriptions of objects.

MOTION

In the domain of objects constructed out of blocks, most of the functions one considers seem to be contingent upon the idea of motion or the restriction of motion. The hole is the principal part of the arch because it enables the arch to achieve its function, i.e. one's ability to move through it. The concept of support, or potential support, which may be considered the function of objects like a pedestal or table, may be simply defined as the restriction of motion in a downward direction. A wall may be considered as a structure which prevents one from proceeding in a given direction unless a detour of some sort is taken. It seems clear that if we are going to deal with such functions in a program, we should have a set of primitive concepts with respect to motion available for use. This section will consider a set (which is by no means to be thought of as complete) of such primitives as a basis for the study of functional representation.

The motion we are primarily concerned with will be motion in a straight line, or a sequence of straight lines. We will consider the motion of three types of objects: (a) that of a bird, which may move in any direction; (b) that of a ball, whose motion must have no purely upward components; and (c) that of a creature with legs, for whom the vertical components (e.g. climbing a staircase) must be suitably small.

It should be noted here, that this definition of "suitably small" is of necessity rather vague. There will be many other similarly relative concepts mentioned later, in conjunction with ideas about containment, windows, doors, etc. At the present there does not seem to be a general, systematic method for handling these concepts adequately.

It should be realized that the descriptions of motion and holes given here are not intended to be the basis of a general theory of motion or space. Rather, they should be viewed as gross simplifications of complicated concepts which are intended to permit easy description of more abstract relations. Their chief feature is a large amount of expressive power at a low level of complexity.

The general primitives deal specifically with the relationship of a (potentially) moving object with respect to its environment. Basically, they are:

OBSTRUCTION -- A rolling object is considered obstructed in a specific direction (perpendicular to the vertical) if it meets an obstacle on traveling in that direction and must make a suitably long detour (say greater than or equal to the distance already traveled) before it can continue in that direction. Thus obstruction is a function of the length of the obstructing object and its position relative to the moving object.

In figure 1(a), the moving object is obstructed to the "east" because in order to move east it will be forced to move north from X1 to X2 or south from X1 to X3, a distance greater than it traveled from X0 to X1 in the easterly direction. In figure (b) the object is not obstructed because the detour is relatively short. In figures (c) and (d) we would consider the object obstructed if the perpendicular component of the total path traveled becomes greater than the component in the desired direction. In the case of backward motion (e) in either of the two directions, this may be simply subtracted from the forward component, since one can generally trace an alternative path which does not contain the components backtracked over. As with all the motion primitives, it will be useful to add the qualifier RELATIVELY when an object contains sufficiently sparse (say no more than one fifth the total length) holes which permit the desired motion, but would represent an obstruction if these holes were blocked. (This is to be seen as a tentative answer to the "window" problem where, topologically speaking, a building with an open window does not enclose anything.)

COVERING -- An object will be considered covered if in traveling upwards, the resultant of motions perpendicular to the upward directions ever becomes greater than the upward component of the motion. As with obstruction, backtracking is subtractive, and there is an analogous notion of relatively covered.

SUPPORT -- It is of interest that the existence of gravity dictates that the concept of support be not quite analogous to that of covering. First of all we assume that an object cannot be at rest unless it is supported by another object or group of objects which are at rest. The actual definition of support, however, is likely to give us some trouble. We could define support in such a way that an object is supported by all the objects in contact with its bottom. But this ignores the question of what would happen if we wanted to remove some of the objects beneath the supported one. A simple defintion of support is that an object is supported by any set of points such that one cannot pass a vertical plane through

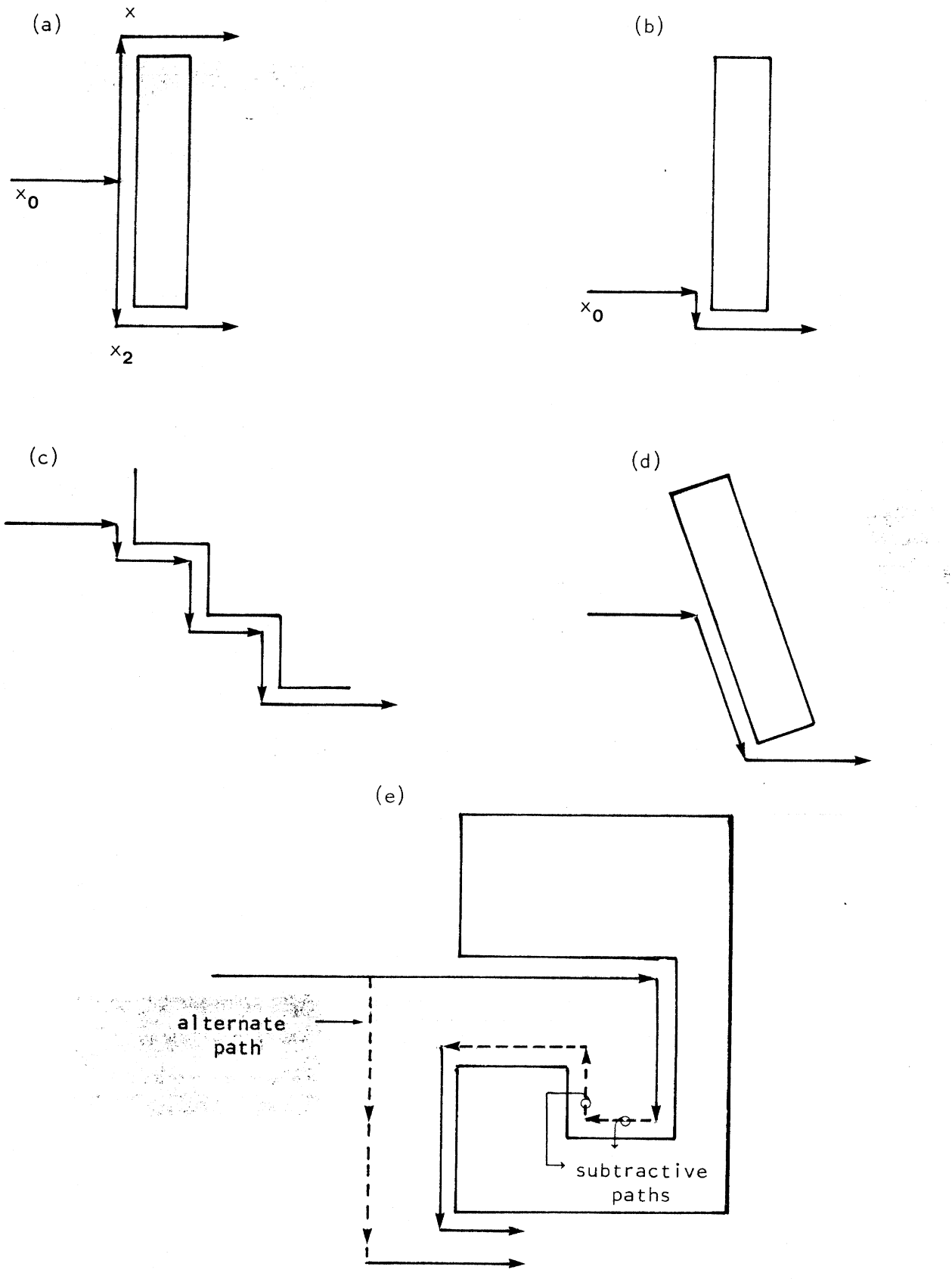


FIGURE 1

the center of gravity of the object which places all members of the set on one side. Such a definition at first glance may seem computationally messy, but there are several alleviating factors. For one we will be dealing with blocks, and in general the supports will be surfaces rather than points. Any points which do appear will of necessity arise from pyramid type objects which prevent their being placed arbitrarily close. In any case, the rule used to determine if an object is supported will not affect the use of the support predicate in higher level computation.

Another concept which is potentially useful is that of SAFE SUPPORT. An object is safely supported if it cannot be rolled to a position where it will drop vertically, i.e. there are no points under it directly in contact with its surface. Since a relatively safe support would not really be very safe, there doesn't seem to be much use for it.

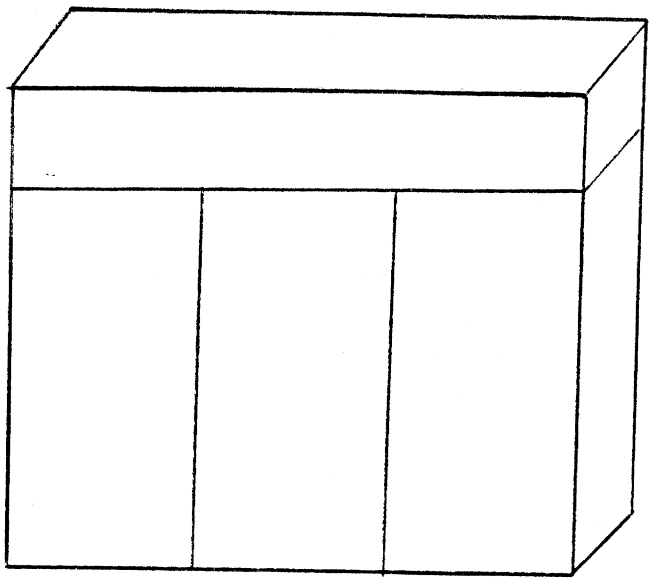
SURROUNDED -- We say an object is surrounded if (a) it is safely supported and (b) it is obstructed in all horizontal directions. In other words, if it were a ball, its motion would be restricted to a fixed horizontal surface. The concept relatively surrounded refers to the relative obstruction in all directions.

CONTAINED -- An object is contained if its movement (in any direction) is restricted to a fixed subspace of the world space. We may make boundary conditions explicit by considering the "world" we observe as a fixed 3-dimensional rectangle with clear walls. All things will be considered contained in the world (unlike Columbus, we do not have to face the prospect of falling off the edge). An object will be considered relatively contained in the usual manner, as long as it is safely supported.

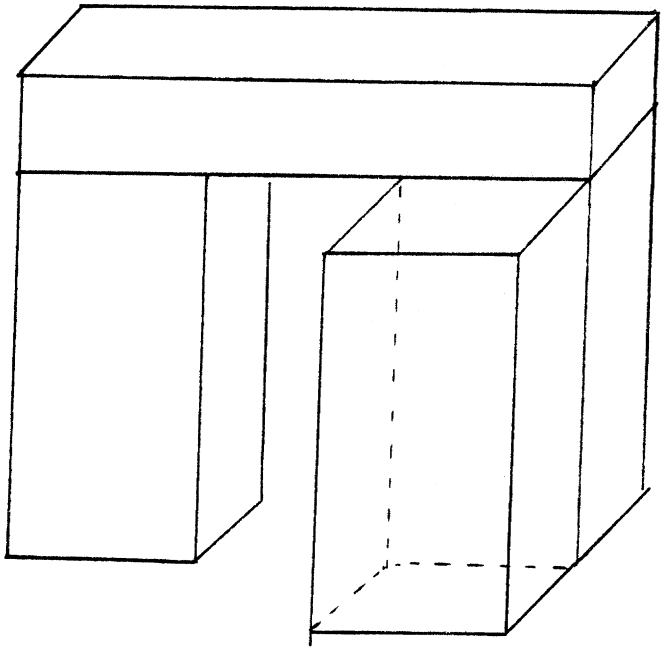
ATTACHMENT -- Objects which are not attached to each other may move independently. There are two types of attachment I feel should be considered -- face attachment and edge attachment. If a face (or suitable subregion thereof) is attached to to the face of another object, the two objects are essentially one object in that they must move together. If an edge of an object is attached to some other object, the former object is said to be edge attached and is free to pivot about that edge with respect to the other object. This will enable us to deal with items such as doors and gates in a structure. We will ignore tolerance problems in door jambs. For example in figure 2 (a) if the marked edge is attached as indicated, we will assume the block A may move freely to the position indicated in 2(b), provided, of course, it is not obstructed as in (c).

FIGURE 2

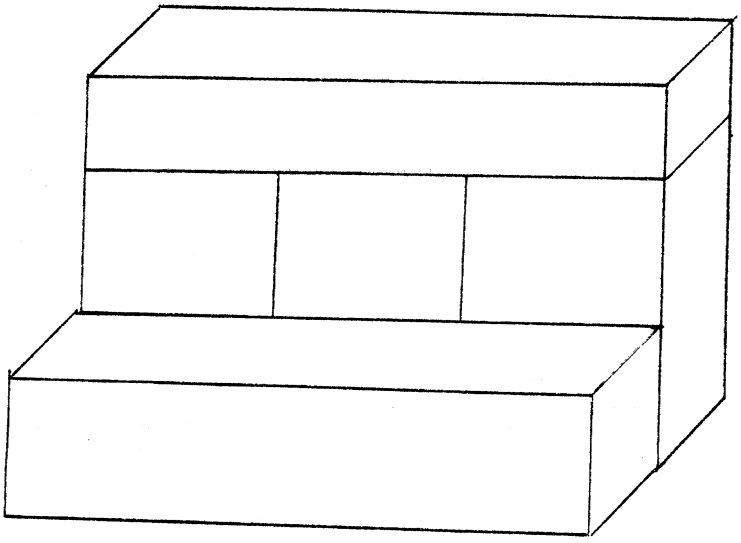
(a)



(b)



(c)



Using these concepts we can provide simple definitions for many common block structures. A box (figure 3) is any structure capable of surrounding an object. A canopy (figure 4) is any structure capable of covering an object. A door (figure 2) is a block which is edge attached to an arch such that in one possible position, the arch and door form a wall. A wall is any group of objects which obstruct motion in some direction for a moving object sufficiently close. Addition of the word "group" is important because otherwise, any object could serve as a wall for a near enough object.

It seems evident that in a system using both functional and structural definitions, we must be careful not to confuse them. Hierarchies formed on the basis of function may differ greatly in their organization from those formed on the basis of structure. Functionally, a table may serve as a pedestal and vice versa, although they differ structurally. Despite the fact that functional criteria may prove a valuable aid in choosing candidates for a class, we will not in general wish to define classes solely in terms of function.

HOLES

Since an object may only move through space which is unoccupied, it is a logical step to desire that freedom and restriction of movement be represented in terms of unoccupied space or holes. In general, holes are not an easy thing to represent. The statement "Holes are the complements of simple objects, and the complements of simple objects are not in general simple" seems to shed some light on the fact. The contour of free space in a given room at a given time may be exceedingly complex. However, for the purposes considered here, it should not be necessary to worry about such complicated questions. Basically, it would be desirable if our representation of holes did not differ too greatly from our representation of the other items in our world. This suggests that we consider holes as composed of blocks of free (or as we shall see later, potentially free) space, generally rectangular in shape. Such a representation also seems advantageous for other reasons. Dividing the free space up into blocks will also give us clues as to which parts of a structure should be grouped together. But perhaps most important, the primitives relating to motion which were discussed in the previous section lend themselves readily to analogy with holes.

It seems advantageous to define holes with respect to a given structure or group of structures. Thus a hole may in part consist of solid objects not attached to the given structure. The reason for this is that any unattached object may be moved independently of the given structure. Suppose we have a box with a block in it (figure

FIGURE 3

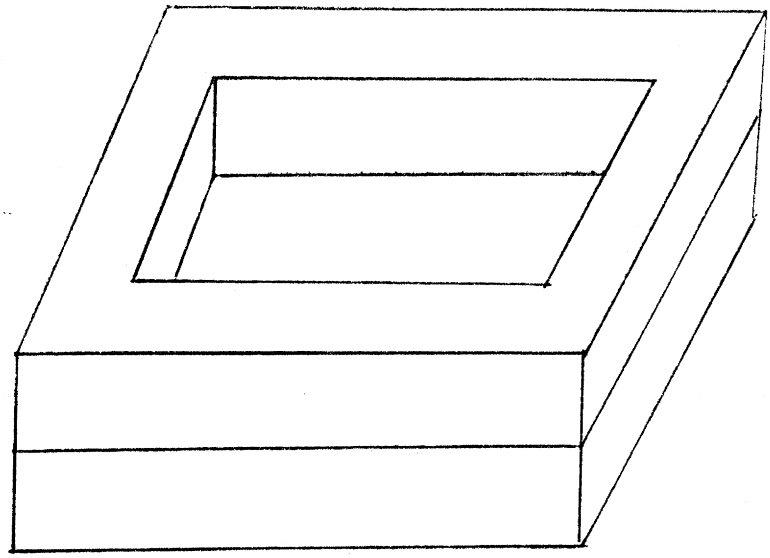
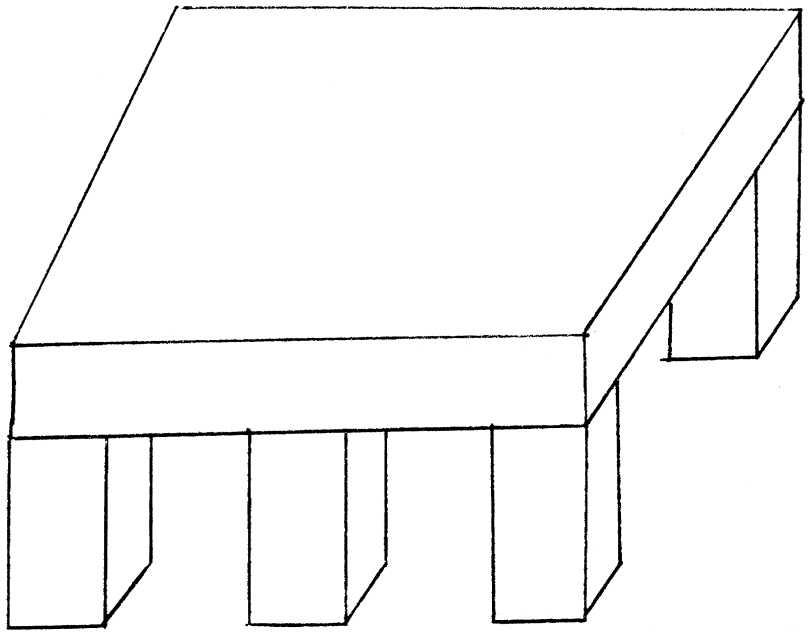
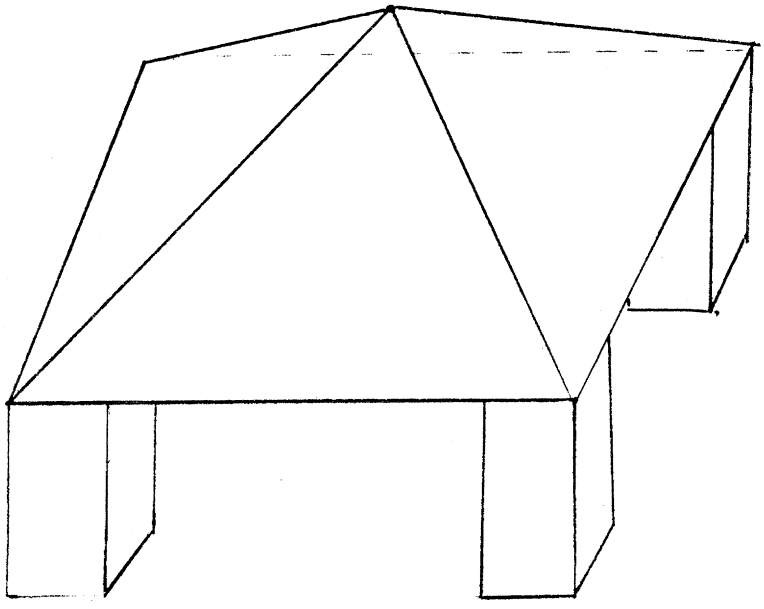


FIGURE 4

(a)



(b)



5), and we move the block to a different position in the box. We do not want to be forced to consider these situations as representing two different holes, so we choose to include the unattached block within the hole which describes the interior of the box. Besides making the task simpler, however, this scheme should give us a certain power in altering descriptions. For instance, suppose we have a ring of blocks surrounding an object (figure 6), all of which are attached except one. Then with respect to the rest of the structure that block represents a hole (albeit a hole which has been temporarily filled) and the presence of that block is not essential. If we are interested in forming an entrance to the area surrounded by the blocks, we know that all we need do is push the block out and we have our hole. (In a sense we will get this information for free if the unattached block is already considered as a hole relative to the attached ring.)

Rectangular holes may be classified according to the number of sides on which they are bounded. Conveniently the number of bounding sides coincides with the general purpose of such a hole.

Passages (ramps) -- Passages are holes which are bounded by three edges, one parallel to the ground and two which are vertical and parallel, above the horizontal edge. The key function of a passage is that it limits non-flying objects to motion along only one line. Though hard to visualize as holes, they are useful in understanding the functions of roads and bridges.

Ports -- A port is a space bounded by four edges, all perpendicular to a given plane. They are similar to passages, in that they restrict motion to a line. A port may be long, e.g. a tunnel; or short, e.g. an arch and its supporting surface. Generally, their purpose is to provide for motion from one region to another. Unlike passages ports restrict the motion of any object.

Niches -- A niche is a space bounded by five edges, the lowest of which must be parallel to the ground. Niches generally provide places for objects to rest or be contained. Boxes (figure 3) and wall indentations are both examples of niches. Any niche supports an object, while a niche with only one edge parallel to the ground will safely support any object inside.

Rooms -- Rooms (for want of a better word) are considered to be areas of space bounded on all sides -- i.e. completely enclosed. They represent the idea of containment.

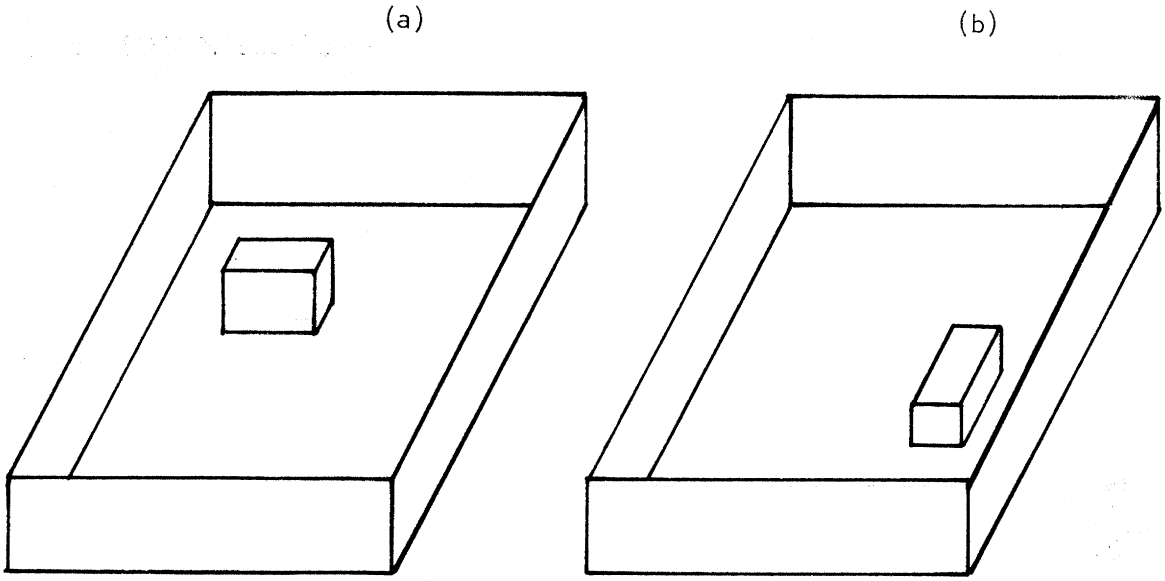


FIGURE 5

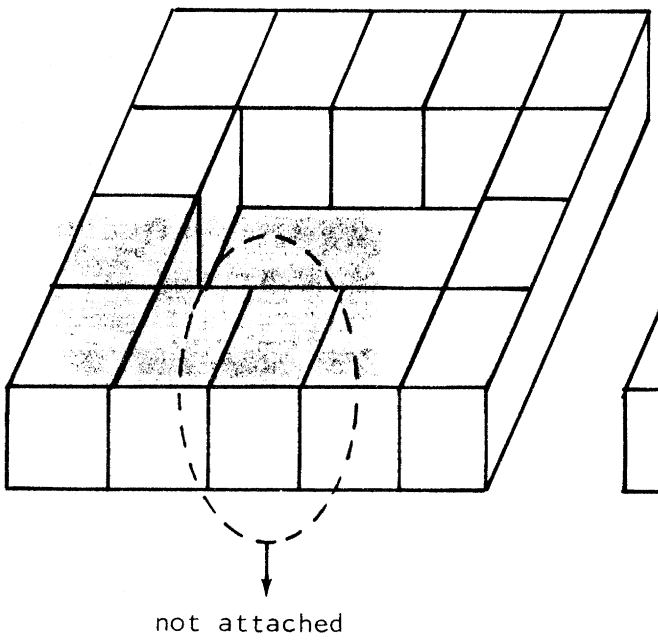


FIGURE 6

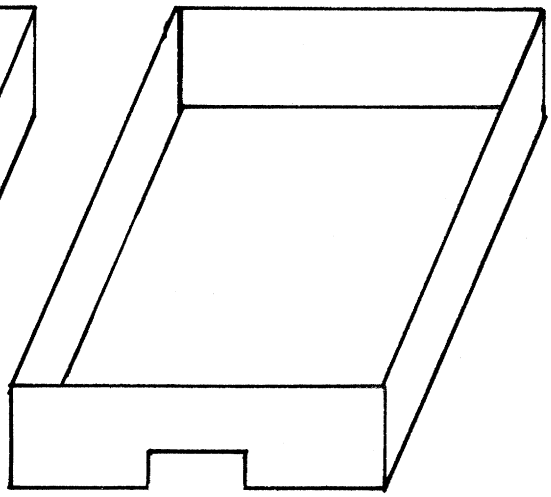


FIGURE 7

It should be mentioned that the edges bounding holes do not have to be solid. They may contain reasonably small (say not more than one sixth total surface area) holes, with the general provision that the floor be solid. For example (figure 7), a box with a port in one side is still considered a niche, or a tunnel with a window would still be considered a tunnel. Thus most of the motion concepts to which holes correspond are the "relative" counterparts of those concepts.

It is interesting to note that some of the motion concepts discussed in the previous section have direct representation in terms of holes. The notion of constraint to one direction of motion and safe support may be achieved by either a port or a passage. The notion of surrounded may be directly represented by a niche with only one edge parallel to the ground. The notion of containment translates directly into a room.

For the sake of simplicity, some perfectly natural conditions have been neglected. Consider for example a room with a sunken area in the middle. One would like to consider this still to be a room, but it might conflict with the notion of containment since containment implies safe support, and a large enough niche in the center might cause us to consider an object in this particular room not safely supported. The bug here is probably with our notion of "safe-support." We probably want to permit "sufficiently small" drops. This would allow us the liberty of considering structures like staircases to provide safe support.

A FORMALISM FOR FUNCTION

Any system which plans to provide some representation for function must also provide a formalism for such representation. The preliminary formalism described here is exceedingly simple (no doubt reflecting the simplicity of the domain) but some aspects suggest generalization to more complicated areas. Syntactically, ?X represents a pattern match which binds X to any item occurring in that positon, much like the pattern matching rules of Planner or Conniver. The symbol "$" represents "self", i.e. "$" is considered a reference to the object whose function we are describing. For example, (SUPPORTS $ X) indicates that the object we are describing supports an object named X. The formalism basically consists of a predicate, POSSIBLE, the logical connectives AND, OR, NOT, and CHOICE, which corresponds to exclusive or, e.g.

(CHOICE (HAVE ?X CAKE) (EAT ?X CAKE))

and some functions and predicates (IN ?X ?Y), (IS ?X ?Y) <true if X is a member of class Y>, (SUITABLE-OBJECT ?X ?MODE), and PASS, SURROUNDED-BY, SUPPORTED-BY, CONTAINED-IN, OBSTRUCTED-BY, and COVERED-BY.

(IN ?X ?Y) returns T if X is located in some hole which is part of the description of structure Y. (SUITABLE-OBJECT ?X ?MODE) generates a structure representing a movable object whose size is reasonable with respect to structure X. MODE is optional. Where specified, it refers to FLY, WALK, or ROLL depending on which motion abilities we desire the generated objects to posess. The others are all predicates of the form (PREDICATE ?X ?Y ?MODIFICATIONS) where the possibilities for MODIFICATIONS vary with the predicate. For COVERED-BY, SURROUNDED-BY and CONTAINED-IN, MODIFICATIONS may be RELATIVELY or NIL. For SUPPORTED-BY it may be SAFELY or NIL. For OBSTRUCTED-BY, MODIFICATIONS is a list of the form (DIRECTION, MOD2) where DIRECTION represents a direction and MOD2 represents RELATIVELY or NIL. For PASS, MODIFICATIONS may be ON or THRU or BETWEEN. ON may apply only to passages, THRU applies to holes in general and BETWEEN to a list of two objects.

POSSIBLE is a general predicate which operates on the motion primitives. It asserts that there is currently no condition which prevents the relation on which it operates from taking place. If it can make the predicate true it returns T, otherwise NIL. CHOICE is a predicate operating on a list (L1 L2 ....LN) of predicates, and can best be understood in terms of a predicate CAN-MAKE (similar to Planner's THGOAL) which succeeds if it proves its argument can be realized through limited manipulation of the structures involved. (CHOICE L1 L2 . . . LN) is equivalent to:

```
(AND (CAN-MAKE L1)
     (CAN-MAKE L2)
          .
          .
          .
     (CAN-MAKE LN)
     (NOT (CAN-MAKE (AND L1 (OR L2 ....LN))))
                    .
                    .
                    .
     (NOT (CAN-MAKE (AND LN (OR L1 ....LN-1))))) .
```

Examples:

ARCH

```
(POSSIBLE (PASS (SUITABLE-OBJECT $ WALK) PORT1 THRU)
```

DOOR

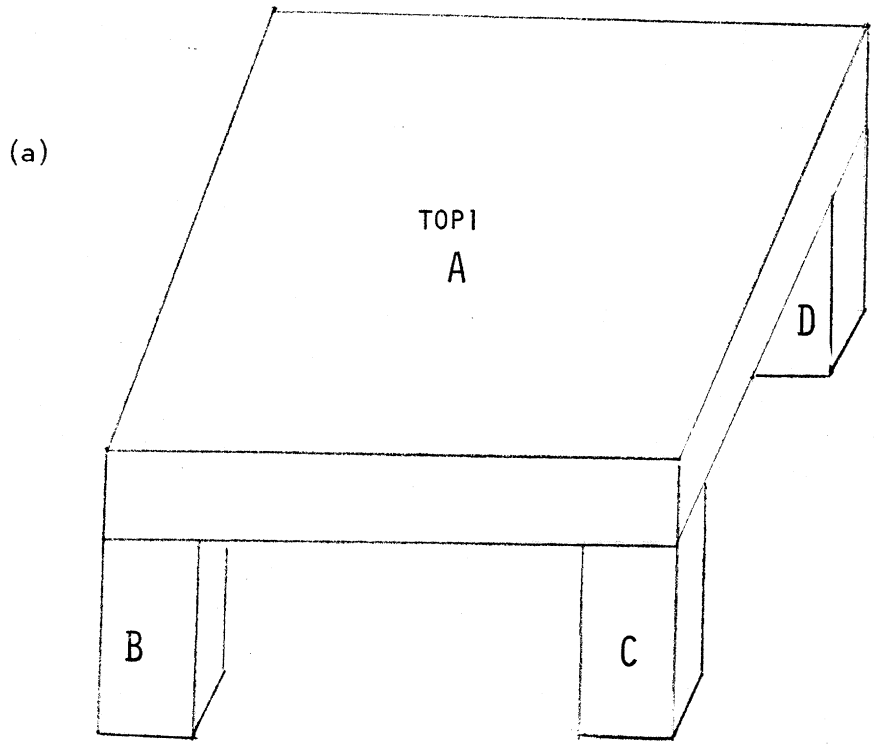
```
(CHOICE
  (POSSIBLE (PASS (SUITABLE-OBJECT $ WALK) (B1 B2) BETWEEN))
  (NOT POSSIBLE (PASS (SUITABLE-OBJECT $ WALK) (B1 B2) BETWEEN)
```

ROAD

```
(AND
  (POSSIBLE (PASS (SUITABLE-OBJECT $ WALK) $ ON))
  (POSSIBLE (SUPPORTED-BY (SUITABLE-OBJECT $ WALK) $ SAFELY)
```

USE OF FUNCTIONAL REPRESENTATION

Assuming we have a program which takes a structure and interprets it in such a way as to discover all the pertinent holes, we may use the holes and other information to construct a list of possible functions for the structure. Let us look for example at a table (figure 8). Our hypothetical hole finder will find the four ports shown in figure 8. Furthermore the large square area of the top suggests that it will support something. Consequently the list of possible functions will be:



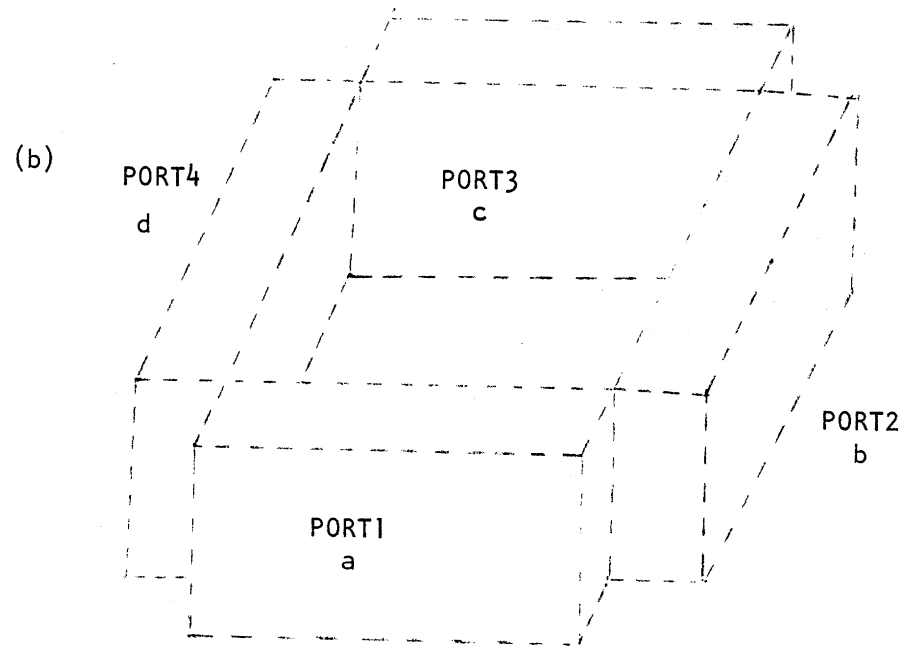


FIGURE 8

```
(POSSIBLE (PASS (SUITABLE-OBJECT $) PORT1 THRU))
    "       "          "            PORT2   "
    "       "          "            PORT3   "
    "       "          "            PORT4   "
(POSSIBLE (SUPPORTED-BY (SUITABLE-OBJECT $ ROLL) TOP1 ))
(POSSIBLE (COVERED-BY (SUITABLE-OBJECT $ FLY) TOP1))
```

If we are now searching this structure for a table, and the functional representation of table is

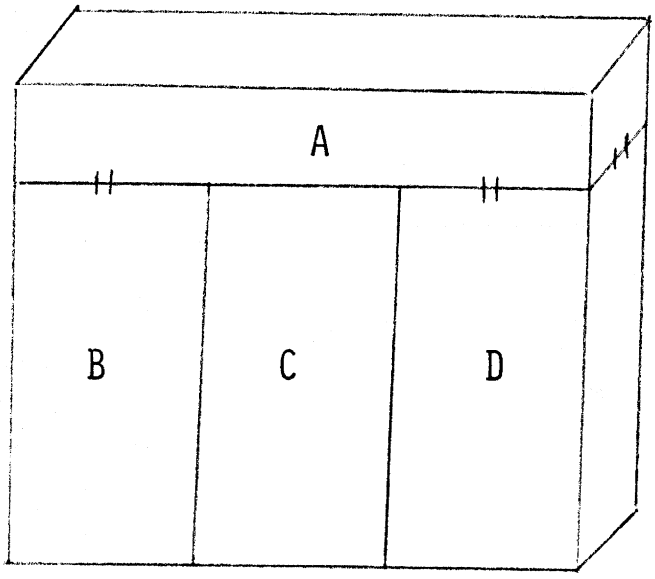
(POSSIBLE (SUPPORTED-BY (SUITABLE-OBJECT $) T4)

where T4 points to the table top in an internal description of the table, we immediately have an anchor with which to begin our comparison with the table description. Winston's program would have to search the entire description before deciding to link the two table tops. Furthermore, if we were looking for a house in figure 8, and assuming the house had a functional representation

(POSSIBLE (CONTAINED-IN (SUITABLE-OBJECT $) R1 RELATIVELY))

we would see that this does not occur in the description and would not even have to bother trying to match the two descriptions. It is interesting to note that if we are actually looking for a pedestal (which will have essentially the same functional description as a table) we will succeed on the functional description but then end up with a bad match. But we haven't lost entirely, because the machine has discovered an important thing. While not strictly speaking a pedestal, the object of figure 8 may be used for a pedestal if one is needed and none are around. If at some later point the machine wishes to build something, this information may prove invaluable. Consequently, we see that certain advantages will accrue from keeping our knowledge of functional properties separate (in some sense) from our knowledge of structural form.

Another interesting action is that of finding an arch in a scene like figure 9, where the hole is blocked. Assuming an initial grouping by attachment, and that block C is not attached to the others, C will be represented as a hole with respect to substructure A-B-D. The arch will be easily found. Winston's program would have to explore possibilities A-B-C and B-C-D as well as A-B-D in determining the arch. Suppose now that C were edge-attached to B or D. Then by considering the extremes of its motion (figure 2B), our function-finding program should know enough to construct:



‖ → attached blocks

FIGURE 9

```
(CHOICE
  (POSSIBLE (PASS (SUITABLE-OBJECT $) (B D) BETWEEN))
  (NOT (POSSIBLE (PASS (SUITABLE-OBJECT $) (B D) BETWEEN))))
```

matching the description of door.

It is appropriate to wonder if the domain discussed here is not too simple to afford effective extension to other areas. This is a difficult question. Certainly at some level of structural complexity much more sophisticated theories of physical laws may be needed to adequately describe function. Nevertheless, it is quite striking that the functional concepts discussed find an easy and direct representation in terms of specific structural properties and that certain structural properties may immediately be singled out to provide clues as to functional use. Perhaps this is an artifact of our simple blocks world. But perhaps not.

SUSSMAN'S SKILL LEARNING SYSTEM

In this section we remain in the blocks world, but we turn completely away from considerations of form and function. Instead we ask about how a program can learn to do new things by debugging primitive procedures. In particular we study Sussman's HACKER system for skill acquisition (TR-297). It sheds light on several important issues, including the relationship of problem-solving to learning, the relationship between imperative and declarative aspects of knowledge, the nature of plans and their teleological structure, and the role of bugs and debugging in the refinement of plans.

A THEORY OF PROBLEM SOLVING

A human problem-solver first tries to classify his problem into a subclass for which he knows a solution method. If he can, he applies that method. If he cannot, he must construct a new method by applying some more general problem-solving strategies to his knowledge of the domain. In constructing the new method, he is careful to avoid certain pitfalls he has previously encountered and he may use methods he has previously constructed to solve subproblems of the given problem. The new method is committed to memory for future use. If any method, new or old, fails on a problem for which it is expected to work, the failure is examined and analyzed. As a result the method may be modified to accommodate the new problem. Often the analysis of the failure can also be classified and abstracted to be remembered as a pitfall to avoid in the future when constructing new methods.

HOW HACKER WORKS

HACKER's purpose is to cope with construction planning in the BLOCKS world. HACKER's structure is outlined in figure 1. When given a problem, HACKER first checks to see if it has a program in his Answer Library whose pattern of applicability matches the problem statement. If so, it runs that program. If not, it must write a new program, using some general knowledge of programming techniques applied to his knowledge of the Blocks World. Any proposed program is criticized to avoid certain bugs it has previously encountered. HACKER may use subroutines (in the Answer Library) it has previously constructed to solve subproblems of the given problem. After criticism, the proposed solution program is tried out. The new program is stored in the Answer Library, indexed by an applicability pattern derived from the statement of the problem

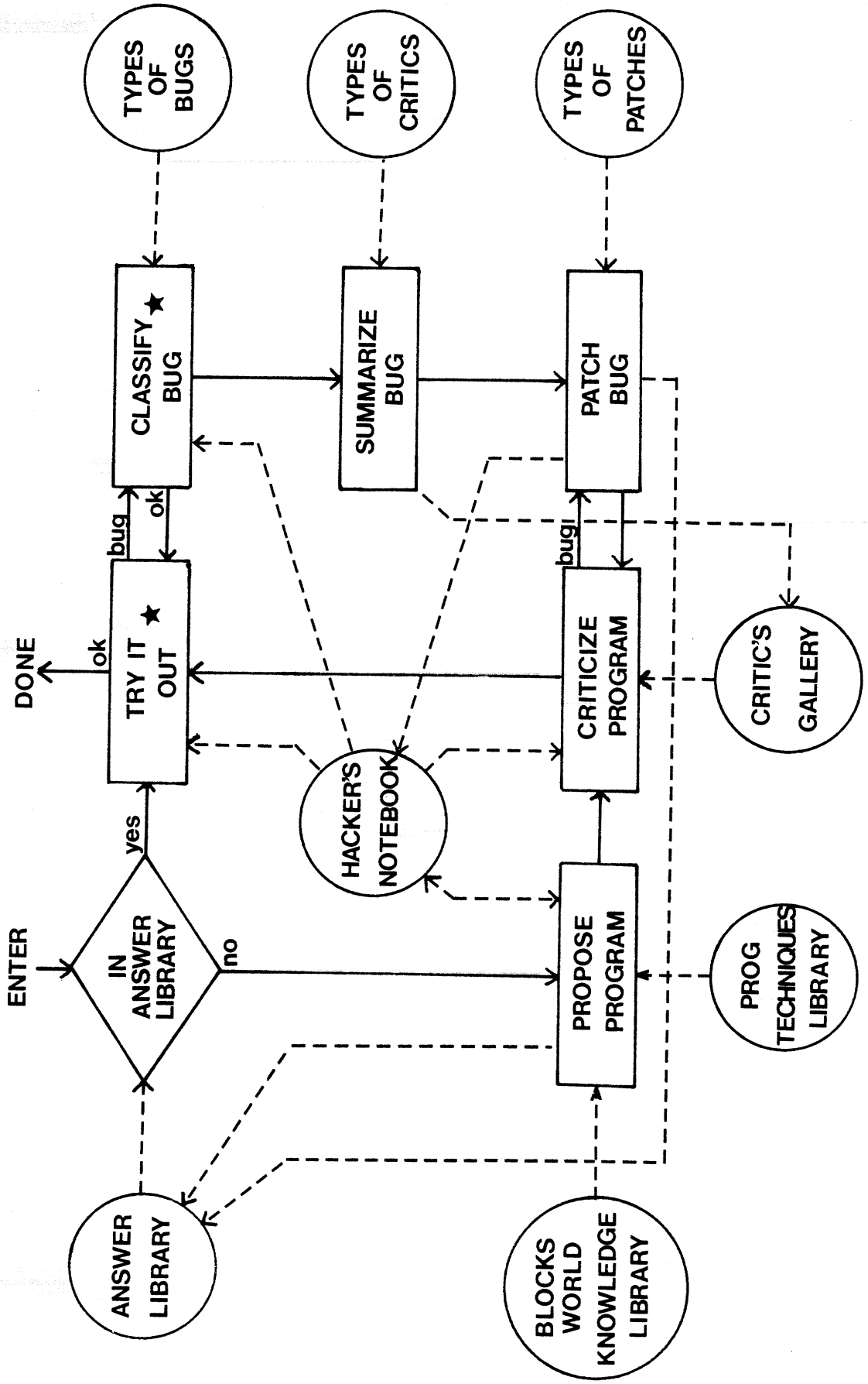

TYPES OF BUGS
TYPES OF CRITICS
TYPES OF PATCHES
CLASSIFY BUG ★
SUMMARIZE BUG
PATCH BUG
bug
ok
bug
TRY IT OUT ★
DONE
ok
CRITICIZE PROGRAM
CRITIC'S GALLERY
HACKER'S NOTEBOOK
ENTER
IN ANSWER LIBRARY
yes
no
PROPOSE PROGRAM
PROG TECHNIQUES LIBRARY
ANSWER LIBRARY
BLOCKS WORLD KNOWLEDGE LIBRARY

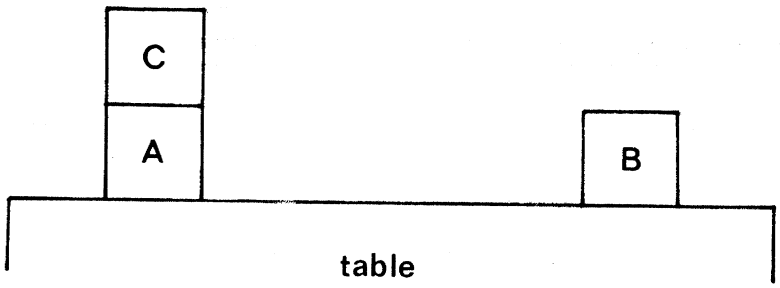
FIGURE 1

for which it was written, so that it can be used to solve similar problems in the future. If any program, new or old, manifests a bug when it is applied to a problem which matches its pattern of applicability, general debugging knowledge is used to classify the mode of failure. Often, the nature of the bug can be summarized and remembered as a critic. The program is patched to fix the bug and tried again.

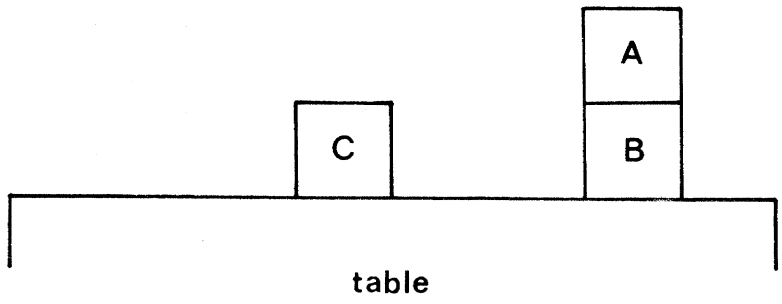
PROBLEM STATEMENTS AND THE ANSWER LIBRARY

Problems are posed to HACKER in the form of simple patterns such as (ACHIEVE (ON A B)). These patterns do not completely specify the final state, they only specify a kernel of the problem to be solved, leaving the problem to be further specified by "reasonable" completion. For example, suppose the initial situation is as pictured below and we pose the problem:

(ACHIEVE (ON A B))



We have certainly not told HACKER what to do with block C. But the following is a reasonable completion of the kernel goal state:



The most convenient completion of a problem statement depends upon just what method in the Answer Library is retrieved to compute the answer, or what methods are retrieved to build and debug an answer if there is not one immediately available.

The Answer Library is a set of programs indexed by patterns which describe the problems for which they are considered relevant. The patterns are simple s-expressions which contain variables and which are matched against the problem posed. Thus, the Answer Library may contain a method which matches the problem:

```
(TO (ACHIEVE (ON x y))
    (ACHIEVE (CLEARTOP x))
    (ACHIEVE (PLACE-FOR x y))
    (PUTON x y))
```

The pattern language is rather poor in expressivity. It is difficult to encode complex requirements elegantly.

PROPOSAL OF PROGRAMS

If there is no procedure in the Answer Library which matches the problem posed, a program must be proposed to solve the problem. There are two ways by which a new program can be proposed by HACKER's program proposer.

1) By generalization of a piece of code previously written to solve a different but "similar" problem.

2) By application of a general plan whose pattern of applicability matches the problem statement.

In the first case, HACKER's notebook is examined to determine if HACKER has ever before written code to satisfy a similar goal. If it is possible to variabilize such a program, and the goal for which it was written so that it matches the new problem, the old code is extracted and subroutinized (inserted in the Answer Library with the generalized pattern) so that it is applicable in the new situation as well as the old one.

In the second case HACKER examines his Programming Techniques Library and his Blocks World Knowledge Library (collectively, his "Bag of Tricks") for a pattern-directed displacement macro to expand and replace the problem statement. There are three types of such macros:

1) There are macros which change the representation of the problem. For example, if the problem is (ACHIEVE (CLEARTOP A)) it is necessary to be able to substitute some more expanded version of the problem if we do not recognize it directly. Thus, the Blocks World contains:

(MEANING-OF (CLEARTOP x) (NOT (EXISTS (y) (ON y x))))

Expansion of this macro on the given problem yields the changed problem:

(ACHIEVE (NOT (EXISTS (y) (ON y A))))

2) There are also macros which expand problems of a particular form into code. Such a macro represents a possible imperative semantics for statements of the form they match. For example:

```
(CODE-FOR (ACHIEVE (NOT (EXISTS vars pattern)))
        (FOR-EACH vars pattern
                (ACHIEVE (NOT pattern))))
```

compiles the following program for the given problem:

```
(FOR-EACH (y) (ON y A)
        (ACHIEVE (NOT (ON y A))))
```

where FOR-EACH is a canned loop often used in CONNIVER programs for iterating through data items.

This kind of macro can easily compile a bug. It may not be possible, for example, to make (NOT (EXISTS (y) (P y))) true by independently making (P a) false for each "a" for which it is now true. This is most clearly true for the particular macro which expands conjunctions into a "linear theory" plan.

3) Finally, there are "tricks" concerned with the details of the Blocks World. One of these (very simplified) is:

```
(CODE-FOR (ACHIEVE (NOT (ON x y)))
      (ACHIEVE (ON x TABLE)))
```

Thus when the code for (CLEARTOP A) is run this macro further transforms the program into:

```
(FOR-EACH (y) (ON y A)
      (ACHIEVE (ON y TABLE)))
```

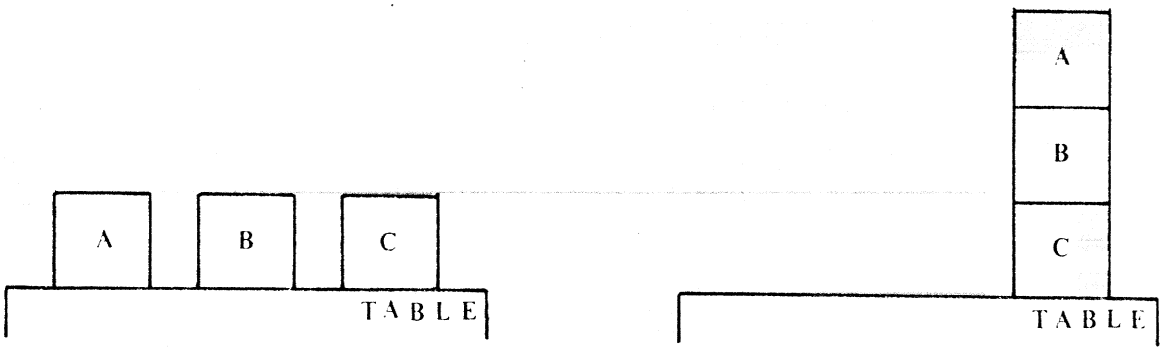
THE ORIGINS OF BUGS

HACKER has ways of repairing bugs when they come up, but how do bugs come up? There are several important sources of bugs. Sometimes, because of generalizations made when a new program is inserted in the Answer Library, a program is applied to a kind of situation which was not anticipated when the program was written. Other bugs result from unanticipated interactions between the steps of a proposed solution. Let us examine the genesis and repair of a bug of this latter kind.

Suppose that one is confronted with a composite goal in which the problem is to achieve the conjunction of two conditions. In the absence of any further knowledge about the structure of the problem, what is a rational strategy to follow in attempting to solve the problem? The simplest approach, which has had great success in the history of science, is to begin with a linear theory -- to assume that the two subgoals can be achieved by independent processes. Thus, the linear theory plan is to break up the conjunction into its components, and then achieve each component independently, with the hope that there will be no interference between the subproblem solutions. Of course, this assumption is often false and leads to a bug, but it is a place to start. Understanding the nature of the resulting bug will often point out the correct patch to make and may lead to a more fundamental understanding of the problem domain.

Consider, for example, HACKER's behavior on the following problem: Suppose that there are 3 blocks on the table, A, B and C, and we ask HACKER to build a 3-block-high tower:

(ACHIEVE (AND (ON A B) (ON B C)))



Before After

HACKER has already written a program to (ACHIEVE (ON x y)) for any bricks x,y. But

HACKER cannot find any program in his Answer Library which matches the given conjunction problem. HACKER then goes into program proposal mode. It fishes about for a strategy which matches the problem posed. The linear theory for achieving conjunctions is retrieved. It suggests the plan:

```
(TO AND2 (ACHIEVE (AND (ON A B) (ON B C)))
  L3: (ACHIEVE (ON A B))
  L4: (ACHIEVE (ON B C)))
```

That is, in simplified HACKER syntax: first try to get A on B, then try to get B on C. If the subgoals are independent, their order doesn't matter, so the arbitrary order from the problem statement is used. The proposal is then passed by the criticizer and tried out; the criticizer does not yet know anything about this kind of problem.

Of course, it has a bug. The program, AND2, first puts A on B. Next it tries to put B on C, but that means it must grasp B. It cannot move B with A on it (a physical restriction of the robot's hand), so it removes A from B and puts it on the table. (This is part of that Answer Library subroutine which HACKER has constructed to solve some earlier problem of the form (ACHIEVE (ON x y)) and which is being used here.) Next, it puts B on C and is done. But it failed to achieve its overall purpose -- A is no longer on B!

Actually, in HACKER, the program would never get this far. Besides proposing the plan, the linear theory also placed the following teleological commentary for that plan into HACKER's Notebook (figure 1):

```
(PURPOSE L3 (TRUE (ON A B)) AND2)
(PURPOSE L4 (TRUE (ON B C)) AND2)
```

These state that the author of the plan expected that A would be on B starting after line L3 and remain there at least until the program AND2 was done (the fourth position could have contained a line number in a more complex plan where L3 was a prerequisite step rather than a main step) and B would be on C starting after line L4 and remain there until AND2 was done. When a program is executed for the first time, it is executed in CAREFUL mode. In CAREFUL mode these comments are interpreted along with the lines to which they are attached. A demon was set after L3 to protect the truth of (ON A B) until AND2 is done. This demon interrupted the execution of L4 at the moment A was lifted off of B. The bug is thus manifest as a PROTECTION-VIOLATION and

caught. Control now passes from the interrupted process to the bug classifier.

TYPES OF BUGS

We have seen how a bug can be constructed when a powerful but imperfect method of plausible inference is invoked. What do we do when such a bug comes up? Until recently, it was thought that a very good idea would have been to include a combinatorial search mechanism (e.g. backtracking) to unwind the problem solver back to some earlier point where the next most plausible proposal could be selected and tried out. The hitch with this idea is that this kind of search rapidly leads to a combinatorial explosion -- just what is this "next most plausible" proposal? It might be that the next most plausible proposal will fail in precisely the way that the current one does and that only the one-hundredth most plausible will succeed. Perhaps the program should re-evaluate its plausibilities on the basis of this failure. That is, the program should be able to learn from its mistakes, not only so as not to make the same error again, but to be positively guided by analysis of the structure of the mistake.

If this conclusion is to be taken seriously it becomes important to better understand the nature of bugs; to classify and name the bugs and repair strategies. The idea of thinking of bugs as important concepts and BUG as a "powerful idea" may seem surprising; but we suspect that isolating and systematizing them may become as important in the study of intelligence as classifying interactions has become in physics!

Now let's see how HACKER understands the above mentioned bug, which has manifested itself as a protection violation. What is its underlying cause? The basic strategy of HACKER in debugging a bug manifestation is to compare (a model of) the behavior of the misbehaving program with various prototypical bug patterns. If a match is found, the program is said to be suffering from a bug which is an instance of the prototype.

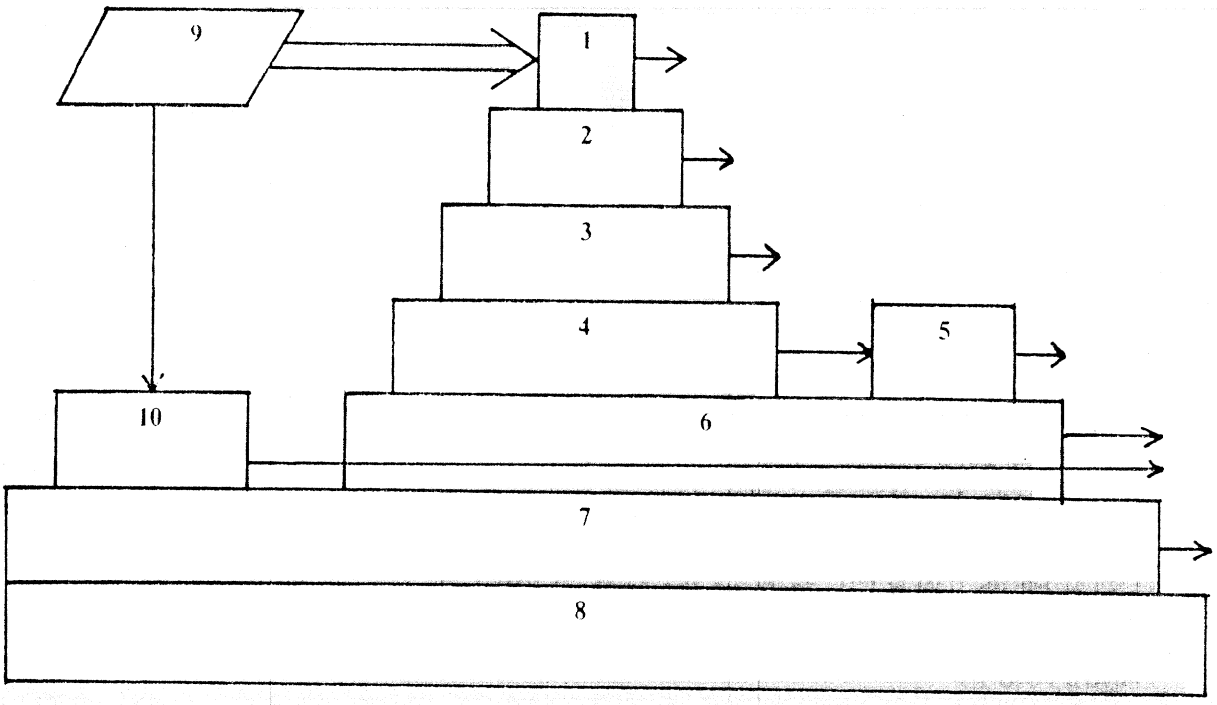
What constitutes a model of the behavior of the misbehaving program and how is it constructed? Here is one scheme: at the time of the PROTECTION-VIOLATION interrupt, the bug classifier has access to an essentially complete chronological history of the problem-solving process which was interrupted. (A human debugger often uses a "tracer" to help him construct such a history, but special features of CONNIVER provide this and more in CAREFUL mode.) HACKER also has access to a complete teleological commentary of the proposed solution and access to variable bindings and other relevant data.

The bug classifier begins by noting two pointers: the current control point and

the origin of the protection comment whose scope was violated. These pointers are then traced with the help of the relevant teleological commentary and history as follows:

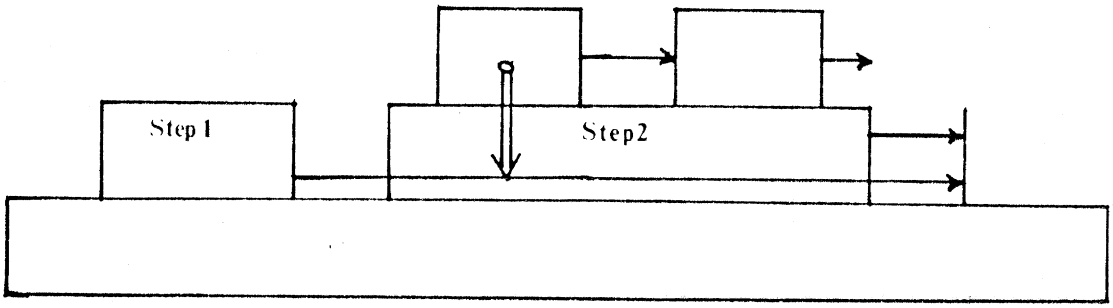
Where was I?	In 1: (PUTON A TABLE)
Why?	Main Step in 2: (ACHIEVE (ON A TABLE))
Why?	Main Step in 3: (ACHIEVE (NOT (ON A B)))
Why?	Main Step Generic in 4: (ACHIEVE (CLEARTOP B))
Why?	Prerequisite Step for 5: (PUTON B C)
Why?	Main Step in 6: (ACHIEVE (ON B C))
Why?	Main Step in 7: (ACHIEVE (AND (ON A B) (ON B C)))
Why?	8: COMMAND
Who complained?	9: Protect (TRUE (ON A B))
Why?	Result of 10: (ACHIEVE (ON A B))
Why?	Main Step in 7: (ACHIEVE (AND (ON A B) (ON B C)))

A Main Step is a step in a program whose purpose is to achieve a result which contributes to the overall goal of the program. Its purpose comment states that the result achieved by that step is needed until the program returns to its caller. A Prerequisite Step is one whose purpose is to set up for the execution of some other step. The result of this trace can be summarized in the following schematic diagram of the buggy process:



Each box in this diagram is a stack frame of the process. The horizontal dimension is its extent in time; the vertical dimension is the depth of functional application. Thus, the blocks labeled 7 and 8 (the AND2 frame and command level frame respectively) exist from the time the command is typed until it returns. Frame number 10 is the frame of line L3:(ACHIEVE (ON A B)) and frame number 6 is the frame of line L4:(ACHIEVE (ON B C)). Frame number 9 is special -- it is the protection demon on the result of L3. It points at the accused violator. The horizontal arrows indicate the scopes of the purposes of the steps. Arrows which terminate on boxes are prerequisite step scopes. (In this trace there is only one prerequisite scope, from 4 to 5.) Other arrows are main step scopes.

This structure matches a particular prototype bug called PREREQUISITE-CLOBBERS-BROTHER-GOAL (PCBG):



By this we mean a bug which is due to an interaction between two program steps whose purpose scopes terminate at the same time. A prerequisite step) for a main step in the code for step 2 clobbered the result of step 1. In this case, the process of achieving (CLEARTOP B), a prerequisite of (PUTON B C), which is a main step in L4:(ACHIEVE (ON B C)), destroys the truth of (ON A B), the result of L3:(ACHIEVE (ON A B)). Since both L3 and L4 are main steps in AND2 their purpose scopes terminate when AND2 returns.

Just how much generality is there in the concept PCBG? Perhaps it is just peculiar to the Blocks World? In fact, PCBG is a very common form of non-linearity.

If, for example, one wants to paint the ceiling, it is simultaneously necessary that the paint be on the platform and that the painter be on the ladder. The linear strategy is to achieve each subgoal independently. The painter can either first lift the can to the ladder platform, and then climb the ladder (which works); or he can first climb the ladder and then lift the can (which doesn't work). Once he is on the ladder, he has no access to the can on the ground. He must first come down to get the paint (clobbering the previously achieved subgoal of being on the ladder). Climbing down -- to achieve the prerequisite to lifting the paint can -- has clobbered the brother goal of being on the ladder.

In programming too, one often runs into PCBG's. Consider the problem of compiling the LISP expression (F 3 (G 4)). If the argument passing convention is to load the arguments into successive argument registers and then call the function, we see that the call to function F requires that 3 be in register 1 and the result of (G 4) be in register 2. If we try the obvious order -- first put 3 in register 1, then calculate (G 4) and put it in register 2 -- we find that we must load 1 with 4 to call G, thus clobbering the brother goal of having 3 in register 1.

FIXING THE BUG

Now that the bug is classified, can we come up with a modification to the plan (program) which eliminates the bug? The offending prerequisite must, in any case, be accomplished before its target step. Its scope must extend until that step. But since the first and second conjuncts are brothers (they are both for the same target), their scopes must overlap. Thus, since the scope of the first conjunct and the scope of the prerequisite of a main step for the second step are incompatible, the only way to prevent the overlap is to move the step for the second conjunct ahead of the step for the first. We must assign an order to the plan. Thus, the patcher changes the plan as follows:

```
(TO AND2 (ACHIEVE (AND (ON A B) (ON B C)))
  L4: (ACHIEVE (ON B C))
  L3: (ACHIEVE (ON A B)))
```

A new comment is added to HACKER's notebook summarizing this ordering constraint (BEFORE L4 L3). The program is patched and the result works. In this case a critic is compiled which summarizes what has been learned. If for any blocks A, B, and C we are proposing a program which has lines with the purposes of getting A on B and B on C, we must compile the line which puts B on C before the one which puts A on B. Applied recursively, this advice is sufficient to ensure that any program which piles up bricks will do it in the correct order -- from the bottom-up.

OTHER BUGS

Of course, not every bug is a PCBG -- not even every bug which manifests as a protection violation. If, for example, we try to build an arch -- (ACHIEVE (AND (ON A B) (ON A C))) -- with a linear theory plan, the bug will manifest as a protection violation but no interchange or other simple modification of the linear theory plan can succeed. This kind of bug is a DIRECT-CONFLICT-BROTHERS (DCB) which can only be resolved using more Blocks World knowledge.

CONCLUSIONS

We can draw the conclusion that to be effective, a problem-solver need not know the precise way to solve each kind of problem. It can attempt to break a hard problem up into subproblems. Sometimes these subproblems can be solved independently, in which case the linear theory plan will work. Sometimes the steps of the plan will interact and debugging will be necessary. And sometimes, because of prior experience, we may know that a particular kind of problem may require a particular kind of nonlinear plan, such as the ordered plan required for the problem discussed here.

The appearance of a few bugs need not be seen as evidence of a limitation of problem solving ability, but rather as a step in the effective use of a powerful problem solving strategy -- approximation of the solution of a problem with an almost-right plan. This strategy becomes powerful if the bug manifestation that results from the failure of such an almost-right plan can be used to focus the problem-solver on the source of the difficulty. A problem-solver based on debugging need not thrash blindly for an alternate plan but can be led by the analysis of the failure -- provided that adequate bug classifying and repairing knowledge is available.

Thus effective problem solving depends as much on how well one understands one's errors as on how carefully and knowledgeably one makes one's initial choices at decision points. The key to understanding errors is in understanding how intentions and purposes relate to plans and actions. This indicates that an important part of the knowledge of a problem-solver is in teleological commentary about how the subparts of the performance knowledge relate to each other so as to achieve the overall goals of the system. It also indicates the need for knowledge about how to trace out bugs and about the kinds of bugs that might be met in applying a given kind of plausible plan.

SOME RELEVANT READING

Bobrow, D. G. and B. Wegbreit. A Model and Stack Implementation of Multiple Environments. Report No. 2334. Publications of Bolt, Beranek and Newman, Inc., Cambridge, Masachusetts, 1972.

GOLDSTEIN'S DRAWING PROGRAM DEBUGGER

In the previous section we saw that Sussman tried to concentrate on domain independant knowledge about programming, avoiding factors specific to the blocks world insofar as possible. In this respect, Goldstein's work on the debugging of simple children's display programs complements Sussman's. Goldstein gives a very thorough account of the epistemology of the domain and how it interacts with the job of correcting the drawing procedures (AIM-305).

In this his Ph. D. work has had the following goals:

1. to provide a more precise understanding of the fundamentals of programming;

2. to facilitate the development of machines capable of debugging and expanding upon the programs given them by humans;

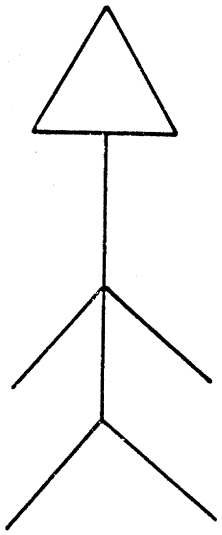
3. to produce insight into the problem solving process so that it can be described more constructively to students.

MYCROFT is intended to supply occasional advice to a student to aid in the debugging of programs that go awry. (Just as the system's namesake, Mycroft Holmes, occasionally supplied advice to his younger brother Sherlock on particularly difficult cases.) In this interaction, the user supplies statements that describe aspects of the intended picture and plan, and the system fills in details of this commentary, diagnoses bugs and suggests corrections.

In this brief glance, our primary purpose will be to describe MYCROFT as a model of the debugging process. This is reasonable since MYCROFT's utility as an adviser stems directly from its understanding of debugging skill.

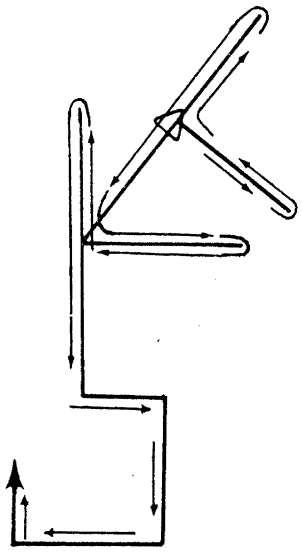
MYCROFT is able to correct the programs responsible for the bugged pictures shown in figures 2, 3, 4 and 5 so that the intended pictures are achieved. The debugging of figure 2, a typical example, will be thoroughly explained. Figures 3, 4 and 5 are corrected in analogous ways.

The pictures are drawn by program manipulation of a graphics device called the turtle which has a pen that can leave a track along the turtle's path. Turtles play an important role in the LOGO environment where children learn problem solving and mathematics by programming display turtles, physical turtles with various sensors, and



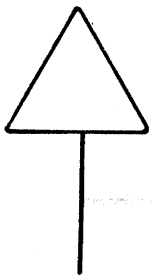
Intended MAN

FIGURE 1

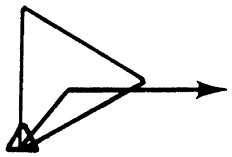


Picture drawn by NAPOLEON

FIGURE 2

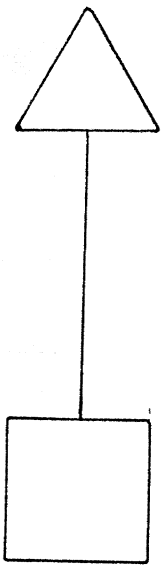


INTENDED TREE

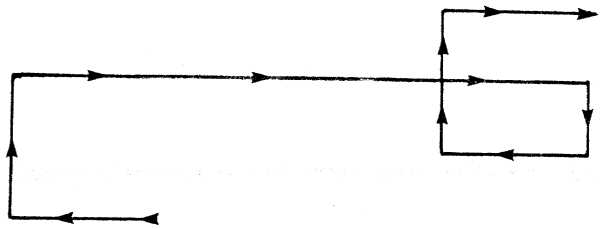


Picture drawn by
bugged TREE program

FIGURE 3

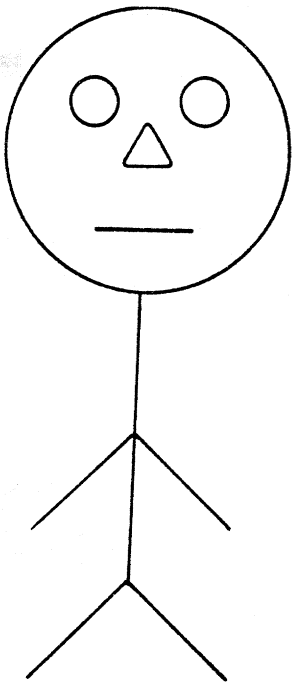


Intended WISHINGWELL

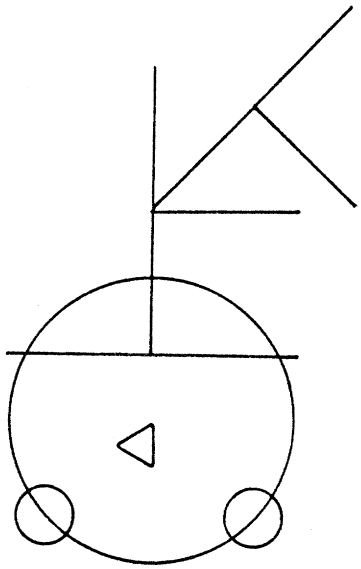


Picture drawn by bugged WISHINGWELL program

FIGURE 4



Intended FACEMAN



Picture drawn by bugged FACEMAN program

FIGURE 5

music boxes (Papert 1972). Turtle programs have proven to be an excellent starting point for teaching programming to children of all ages, and therefore provide a reasonable initial problem domain for building a program understanding system.

The context of MYCROFT's activity is the interaction of three kinds of description: graphic (i.e. the picture actually drawn), procedural (the turtle program used to generate the picture) and predicative (the collection of statements used to describe the desired scene). For MYCROFT, debugging is making the procedural description produce a graphical result that satisfies the set of predicates describing intent. Thus, debugging here is a process that mediates between different representations of the same object.

FLOW CHART OF THE SYSTEM

The organization of the monitor system is illustrated in figure 6. Input to MYCROFT consists of the user's programs and a model of the intended outcome. For the graphics world, the model is a conjunction of geometric predicates describing important properties of the intended picture. MYCROFT then analyzes the program, building both a Cartesian annotation of the picture that is actually drawn and a plan explaining the relationship between the program and model. (Any or all of the plan can be supplied directly by the user thereby simplifying MYCROFT's task.)

The next step is for the system to interpret the program's performance in terms of the model and produce a description of the discrepancies. These discrepancies are expressed as a list of the violated model statements. The task is then for the debugger to repair each violation. The final output is an edited turtle program (with copious commentary) which satisfies the model. (Occasionally, the plan that MYCROFT hypothesizes requires implausible repairs -- for example, major deletions of user code -- resulting in the debugger asking the plan-finder for a new plan.)

We now introduce MYCROFT by describing the debugging of NAPOLEON (figure 2) and discussing some important ideas about the nature of plans. For a discussion of the other modules shown in the flowchart, see (AIM-305).

PICTURE MODELS

To judge the success of a program, MYCROFT requires as input from the user a description of intent. A declarative language has been designed to define picture models. These models specify important properties of the desired final outcome without indicating the details of the drawing process. The primitives of the model language are

FLOWCHART OF MYCROFT

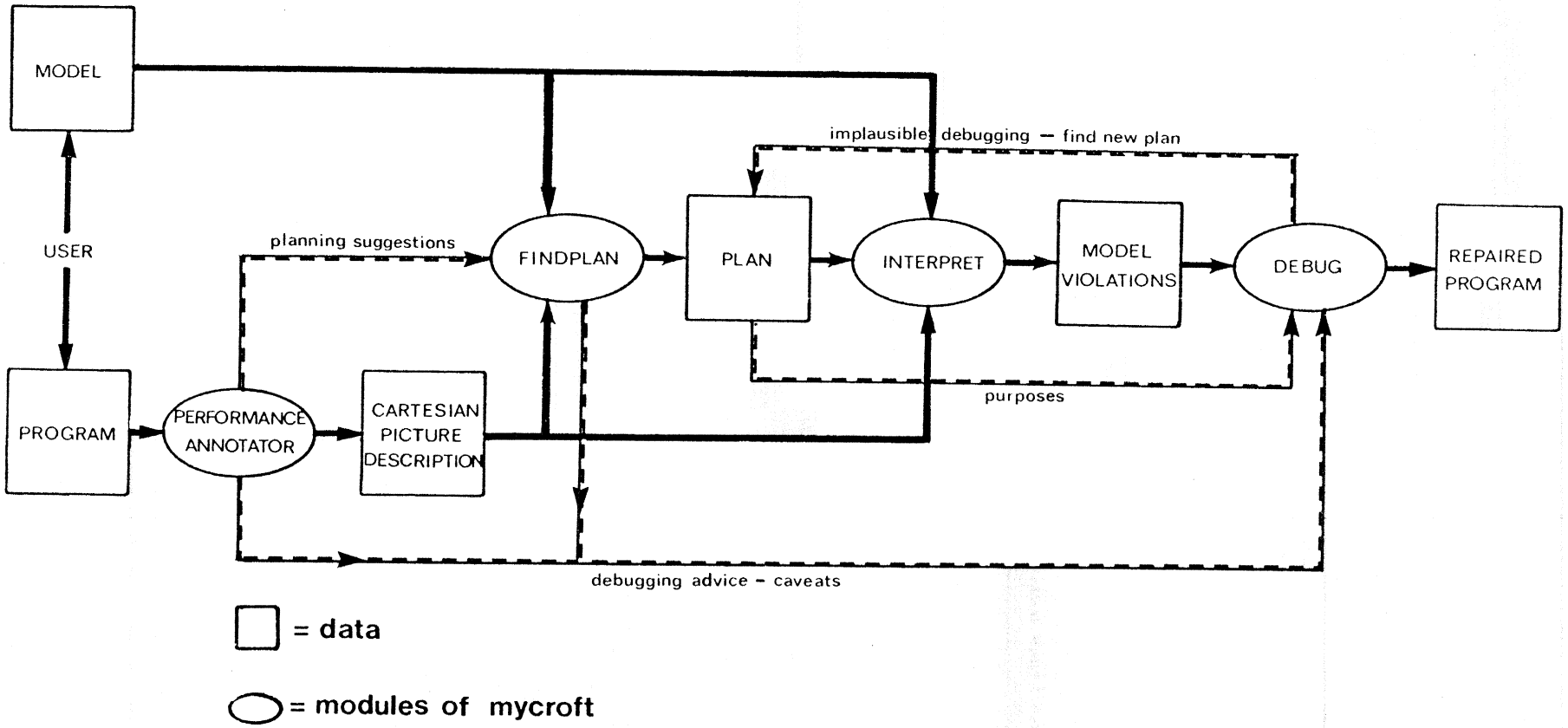


FIGURE 6

geometric predicates for such properties as connectivity, relative position, length and location. The following models are typical of those that the user might provide to describe figure 1.

```
MODEL MAN
M1 PARTS HEAD BODY ARMS LEGS
M2 EQUITRI HEAD
M3 LINE BODY
M4 V ARMS, V LEGS
M5 CONNECTED HEAD BODY, CONNECTED BODY ARMS, CONNECTED BODY LEGS
M6 BELOW LEGS ARMS, BELOW ARMS HEAD
END

MODEL V
M1 PARTS L1 L2
M2 LINE L1, LINE L2
M3 CONNECTED L1 L2 (VIA ENDPOINTS)
END

MODEL EQUITRI
M1 PARTS (SIDE 3) (ROTATION 3)
M2 FOR-EACH SIDE (= (LENGTH SIDE) 100)
M3 FOR-EACH ROTATION (= (DEGREES ROTATION) 120)
M4 RING CONNECTED SIDE
END
```

The MAN and V models are underdetermined: they do not describe, for example, the actual size of the pictures. The user has latitude in his description of intent because MYCROFT is designed only to debug programs that are almost correct. Therefore, not only the model, but also the picture drawn by the program and the definition of the procedure provide clues to the purpose of the program.

THE NAPOLEON EXAMPLE

MYCROFT is designed to repair a simple class of procedures called Fixed-Instruction Programs. These are procedures in which the primitives are restricted to constant inputs. Subprocedures are allowed; however, no conditionals, variables, recursions or iterations are permitted. Given below are the three programs which drew figure 2 -- NAPOLEON, VEE, and TRICORN. The "<-" commentary is called the plan and was generated by MYCROFT to link the picture models -- MAN, V and EQUITRI -- to the programs.

```
TO NAPOLEON              <- (accomplish man)
10 VEE                   <- (accomplish legs)
20 FORWARD 100           <- (accomplish (piece 1 body))
30 VEE                   <- (insert arms body)
40 FORWARD 100           <- (accomplish (piece 2 body))
50 LEFT 90               <- (setup heading (for head))
60 TRICORN               <- (accomplish head)
END
```

```
TO VEE                   <- (accomplish v)
10 RIGHT 45              <- (setup heading for 11)
20 BACK 100              <- (accomplish 11)
30 FORWARD 100           <- (retrace 11)
40 LEFT 90               <- (setup heading for 12)
50 BACK 100              <- (accomplish 12)
60 FORWARD 100           <- (retrace 12)
END
```

```
TO TRICORN              <- (accomplish equitri)
10 FORWARD 50           <- (accomplish (piece 1 (side 1)))
20 RIGHT 90             <- (accomplish (rotation 1))
30 FORWARD 100          <- (accomplish (side 2))
40 RIGHT 90             <- (accomplish (rotation 2))
50 FORWARD 100          <- (accomplish (side 3))
60 RIGHT 90             <- (accomplish (rotation 3))
70 FORWARD 50           <- (accomplish (piece 2 (side 1)))
END
```

The turtle command FORWARD moves the turtle in the direction that it is currently pointed: RIGHT rotates the turtle clockwise around its axis. (A complete description of LOGO is available from the laboratory but is not needed here.)

A Cartesian representation of the picture is generated by an annotator that describes the performance of turtle programs. The plan is used to bind subpictures to model parts. This allows MYCROFT to interpret programs with respect to their models and produce lists of violated model statements. MYCROFT produces the following list of discrepancies for NAPOLEON:

```
(NOT (LINE BODY))           ;The body is not a line.
(NOT (BELOW LEGS ARMS))     ;The legs are not below the arms.
(NOT (BELOW ARMS HEAD))     ;The arms are not below the head.
(NOT (EQUITRI TRICORN))     ;The head is not an equilateral
```

MYCROFT is able to correct these bugs and achieve the intended picture using both planning and debugging knowledge.

PLANS

This section introduces a vocabulary for talking about the structure of a procedure which is useful for understanding both the design and debugging of programs. A main-step is defined as the code required to achieve a particular subgoal (subpicture). A preparatory-step consists of code needed to setup, cleanup or interface between main-steps. Thus, from this point of view, a program is understood as a sequence of main-

steps and preparatory-steps. A similar point of view is found in (Sussman TR-297). The plan consists of the purposes linking main- and preparatory-steps to the model: in the turtle world, the purpose of main-steps is to accomplish (draw) parts of the model; and the purpose of preparatory-steps is to properly setup or cleanup the turtle state between main-steps or, perhaps, to retrace over some previous vector.

A modular main-step is a sequence of contiguous code intended to accomplish a particular goal. This is as opposed to an interrupted main-step whose code is scattered in pieces throughout the program. In NAPOLEON, the main-steps for the legs, arms and head are modular; however, the code for the body is interrupted by the insertion of the code for arms. The utility of making this distinction is that modular main-steps can often be debugged in private (i.e. by being run independently of the remainder of the procedure) while interrupted main-steps commonly fail because of unforseen interactions with the interleaved code associated with other steps of the plan.

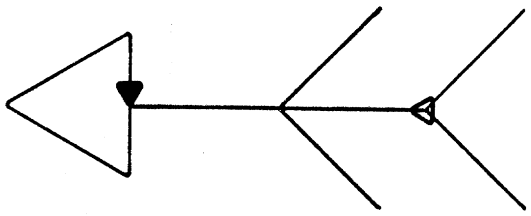
Linearity is an important design strategy for creating programs. It has two stages. The first is to break the task into independent subgoals and design solutions (main-steps) for each. The second is then to combine these main-steps into a single procedure by concatenating them into some sequence, adding (where necessary) preparatory-steps to provide proper interfacing. The virtue of this approach is that it divides the problem into manageable subproblems. A disadvantage is that occasionally there may be constraints on the design of some main-step which are not recognized when that step is designed independently of the remainder of the problem. Another disadvantage is that linear design can fail to recognize opportunities for subroutinizing a segment of code useful for accomplishing more than one main-step. A linear plan will be defined as a plan consisting only of modular main-steps and preparatory steps: a non-linear plan may include interrupted main-steps.

LINEAR DEBUGGING

Linearity is a powerful concept for debugging as well as for designing programs. MYCROFT pursues the following linear approach to correcting turtle programs: the debugger's first goal is to fix each main-step independently so that the code satisfies all intended properties of the model part being accomplished. Following this, the main-steps are treated as inviolate and relations between model parts are fixed by debugging preparatory-steps. This is not the only debugging technique available to the system, but

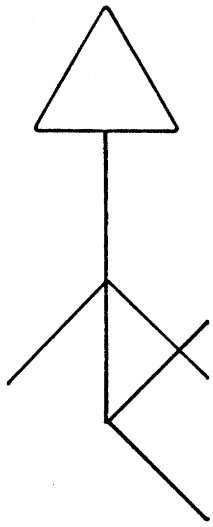
it is a valuable one because it embodies important heuristics (1) concerning the order in which violations should be repaired and (2) for selecting the repair-point (location in the program) at which the edit for each violation should be attempted.

Following this linear approach, MYCROFT repairs the crooked body and the open head of NAPOLEON before correcting the BELOW relations. Repairing these parts is done on the basis of knowledge described in the next two sections. Let us assume for the remainder of this section that these property repairs have been made -- NAPOLEON appears as in figure 7 -- and concentrate on the debugging of the violated relations.



NAPOLEON with parts corrected

FIGURE 7



NAPOLEON with statement 15 as RIGHT 135

FIGURE 8

Treating main-steps as inviolate and fixing relations by modifying setup steps limits the repair of (BELOW LEGS ARMS) to three possible repair-points: (1) before the legs as statement 5, (2) before the first piece of the body as statement 15 and (3) before accomplishing the arms as statement 25. MYCROFT understands enough about causality to know that there is no point in considering edits following the execution of statement 30 to affect the arms or legs. The exact changes to be made are determined by imperative semantics for the model primitives. This is procedural knowledge that generates, for a given predicate and location in the program, some possible edits that

would make true the violated predicate. MYCROFT generally considers alternative strategies for correcting a given violation: it prefers those edits which produce the most beneficial side effects, make minimal changes to the user's code or most closely satisfy the abstract form of the plan.

For BELOW, the imperative semantics direct DEBUG to place the legs below the arms by adding rotations at the setup steps. More drastic modifications to the user's code are possible such as the addition of position setups which alter the topology of the picture; however, MYCROFT tries to be gentle to the turtle program (using the heuristic that the user's code is probably almost correct) and considers larger changes to the program only if the simpler edits do not succeed. The first setup location considered is the one immediately prior to accomplishing the arms. Inserting a rotation as statement 25, however, does not correct the violation and is therefore rejected. The next possible edit point is as statement 15. Here, the addition of RIGHT 135 makes the legs PARTLY-BELOW the arms and produces figure 8. This edit is possible but is not preferred both because the legs and arms now overlap and because the legs are not COMPLETELY-BELOW the arms. MYCROFT is cautious, being primarily a repairman rather than a designer, and is reluctant to introduce new connections not described in the model. Also, given a choice, MYCROFT prefers the most constrained meaning of the model predicate. If the user had intended figure 8, then one would expect the model description to include additional declarations such as (CONNECTED LEGS ARMS) and (PARTLY-BELOW LEGS ARMS).

Adding RIGHT 90 as statement 5 achieves (COMPLETELY-BELOW LEGS ARMS) and the NAPOLEON program now produces the intended picture (figure 1). This correction has beneficial side effects in also establishing the proper relationship between the head and arms, confirming for MYCROFT that the edit is reasonable, since a particular underlying cause is often responsible for many bugs. Thus the result of (DEBUG (BELOW LEGS ARMS)) is:

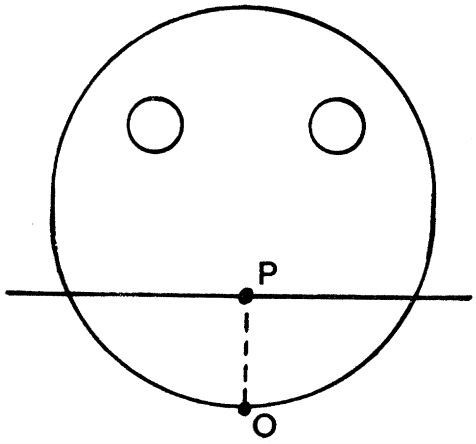
```
5 RIGHT 90 <- (setup heading such-that (below legs arms)
                                       (below arms head))
              (assume (= (entry heading) 270))
```

The assume comment records the entry state with respect to which the edit was made. If the program is run at a future time in a new environment, then debugging is simplified. The cause of a BELOW violation will now immediately be seen to be an incorrect assumption, and the corresponding repair is obvious -- insert code to satisfy

the entry requirements described by the assumption. This illustrates the existence of levels of commentary between the model and the program, each layer being more specific, but also more closely tied to the particular code and runtime environment of the program.

Linear debugging greatly restricts the possibilities that must be considered to repair a violation. It is often successful and constitutes a powerful first attack on the problem of finding the proper edit; however, it is not infallible. Non-linear bugs due to unexpected interactions between main-steps would not be caught by this technique.

Figure 9 illustrates a non-linear bug. (INSIDE MOUTH HEAD) is violated but it cannot



be repaired by adjusting the interface between these two parts (indicated in figure 9 by the dotted line OP) since the mouth is longer than the diameter of the head. The imperative semantics for fixing INSIDE recognize this. Consequently, MYCROFT resorts to the non-linear technique of modifying main-steps to repair a relation between parts. The imperative semantics suggest changing the size of one of the parts because this

transformation does not affect the shape of the part and consequently will probably not introduce new violations in properties describing the part. Advice is required from the user to know whether shrinking the mouth is to be preferred to expanding the head. Two more non-linear debugging techniques are discussed next: one is based upon knowing the abstract form of plans, and the other uses domain-dependent theorems about global effects.

INSERTIONS

In programming, an interrupt is a break in normal processing for the purpose of servicing a surprise. Interrupts represent an important type of plan: they are a necessary problem solving strategy when a process must deal with unpredictable events. Typical situations where interrupts prove useful include servicing a dynamic display, and arbitrating the conflicting demands of a time sharing system. In the real world, biological creatures must use an interrupt style of processing to deal with dangers of their environment such as predators.

A very simple type of interrupt is one in which the program associated with the interrupt is performed for its side effects and is state-transparent, i.e. the machine is restored to its pre-interrupt state before ordinary processing is resumed. As a result, the main process never notices the interruption. In the turtle world, an analogous type of organization is that of an inserted main-step (insertion). It naturally arises when the turtle, while accomplishing one part of a model (the interrupted main-step), assumes an appropriate entry state for another part (the insertion). An obvious planning strategy is to insert a subprocedure at such a point in the execution of the interrupted-step. Often, the insertion will be state-transparent: for turtles, this is achieved by restoring the heading, position and pen state. The insertion of the arms into the body by statement 30 of NAPOLEON is an example of a position- and pen- but not heading- transparent insertion.

Insertions do not share all of the properties of interrupts. For example, the insertion always occurs at a fixed point in the program rather than at some arbitrary and unpredictable point in time. Nor does the insertion alter the state of the main process as happens in an error handler. However, if one focuses on the planning process by which the user's code was written, then the insertion as an intervention in accomplishing a main-step does have the flavor of an interrupt.

The FINDPLAN module aids the debugger in a second way beyond just the generation of the plan. This is through the creation of caveat comments to warn the debugger of suspicious code that fails to satisfy expectations based on the abstract form of the plan. In particular, if FINDPLAN observes an insertion that is not transparent, then the following caveat is generated:

30 VEE <- (caveat findplan (not (rotation-transparent insert))).

The non-transparent insertion may have been intentional, e.g. the preparation for the next piece of the interrupted main-step may have been placed within the insertion. The user's program may have prepared for the next main-step within the insertion. Hence, FINDPLAN does not immediately attempt to correct the anomalous code. Only if subsequent debugging of some model violation confirms the caveat is the code corrected. There will often be many possible corrections for a particular model violation. The caveat is used to increase the plausibility of those edits that eliminate FINDPLAN's complaint. In this way, the abstract form of the plan helps to guide the debugging.

For NAPOLEON, analysis of (NOT (LINE BODY)) leads MYCROFT to consider (1) adding a rotation as statement 35 to align the second piece of the body with the first or (2) placing this rotation into VEE as the final statement. Ordinarily, linear debugging would prevent the latter as it does not respect the inviolability of main-steps. However, it is chosen here because of the corroborating complaint of FINDPLAN. The underlying cause of the bug is a main-step error (non-transparent insertion) rather than a preparatory-step failure. Thus, (DEBUG (LINE BODY)) produces:

70 RIGHT 45 <- (setup heading such-that (transparent vee))

GEOMETRIC KNOWLEDGE

Linearity, preparation and interrupts are general problem-solving strategies for organizing goals into programs. However, it is important to remember that domain-dependent knowledge must be available to a debugging system. The system must know the semantics of the primitives if it is to describe their effects.

The debugger must also have access to domain-dependent information to repair main-steps in which the subparts must satisfy certain global relationships. For example, TRICORN has the bug that the triangle is not closed. Each main-step independently achieves a side but the sides do not have the proper global relationship. Debugging is

simplified by the explicit statement in the model that:

(FOR-EACH ROTATION (= (DEGREES ROTATION) 120)).

But suppose the model imposed no constraints on the rotations. Then the design of the rotations would have to be deduced from such geometric knowledge as the fact that N equal vectors form a regular polygon if each rotation equals 360/N degrees.

The pieces of an interrupted-step such as the first side of TRICORN are not always separated by a state-transparent insert. (This would be a local interruption.) Instead, it is possible that more global knowledge is needed to understand the properties of the intervening code which justifies the expectation that the pieces will properly fit together. In TRICORN, the second piece (drawn by statement 70) must be colinear with the first (drawn by statement 10). The global property of the code which justifies this is that equal sides and 120 degree rotations results in closure. Thus, debugging violations of globally interrupted-steps requires domain-dependent knowledge.

CONCLUSIONS

The design of MYCROFT required an investigation of fundamental problem solving issues including description, simplification, linearity, planning, debugging and annotation. MYCROFT, however, is only a first step in understanding these ideas. Further investigation of more complex programs, and of the semantics of different problem domains is necessary. It is also essential to analyze additional planning concepts such as ordering, repetition and recursion as well as the corresponding debugging techniques. Ultimately, such research will surely clarify the learning process in both men and machines by providing an understanding of how they correct their own procedures.

SOME RELEVANT READING

Papert, S. "Teaching Children Thinking." Programmed Learning and Educational Technology, 9, No. 5 (Sept. 1972).

Polya, George. How to Solve It. Princeton: Princeton University Press, 1945.

Polya, George. Mathematical Discovery. 2 vols. New York: John Wiley and Sons, 1962.

5 UNDERSTANDING VISION

A system written by Winston, Horn, and Freuder succeeded in copying a block's world structure some time ago. Since that early milestone, progress has been steady in understanding both scene analysis and image analysis, both of which are essential to a complete understanding of vision and the successful implementation of seeing systems. To summarize this work, it seems appropriate to divide the section into two parts, one of which concentrates on what we call the heterarchical approach to system building and one of which concentrates on understanding how the physical world's constraints can be exploited in both scene and image analysis.

THE HETERARCHICAL SCENE COPYING SYSTEM

This first section on heterarchy begins with a description of the heterarchical scene analysis stucture of the Winston-Horn-Freuder copying system after generalization and improvement by Finin, Lerman, and others. That system's objectives were, in part, to explore some of the characteristics that define the heterarchical approach:

1. A complex system should be goal oriented. Procedures at all levels should be short and associated with some definite goal. Goals should normally be satisfied by invoking a small number of subgoals for other procedures or by directly calling a few primitives.

2. The executive control should be distributed throughout the system. In a heterarchical system, the modules interact not like a master and slaves but more like a community of experts.

3. Programmers should make as few assumptions as possible about the state in which the system will be when a procedure is called. The procedure itself should contain the necessary machinery to set up whatever conditions are required before it can do its job. This is obviously of prime importance when many authors contribute to the system, for they should be able to add knowledge via new code without completely understanding the rest of the system.

4. The system should contain some knowledge of itself. It is not enough to think of

executives and primitives. There should be modules that act as critics and complain when something looks suspicious. Others must know how and when the primitives are likely to fail. Communication among these modules should be more colorful than mere flow of data and command. It should include what in human discourse would be called advice, suggestions, remarks, complaints, criticism, questions, answers, mistakes, lies, and conjectures.

5. A system should have facilities for tentative conclusions. The system will detect mistakes as it operates. A conjectured configuration may be found to be unstable or the hand may be led to grasp air. When this happens, we need to know what facts in the data base are most problematical; we need to know how to try to fix things; and we need to know how far-ranging the consequences of a change are likely to be.

COPYING TOY BLOCK STRUCTURES

We give here a brief description of the higher level functions of a heterarchical vision system built to copy toy block structures. We also provide a scenario giving the interaction of those functions in a very simple situation. The main purpose is to illustrate the top down, goal oriented and environment dependent flavor of the system.

Figure 1 shows the possible call paths between some of the programs. Note in particular the network quality that distinguishes the system from earlier pass-oriented systems.

Clarity requires that only a portion of the system be described. In particular, the diagram and the discussion omits the following:

1. A large number of MICROPLANNER antecedant and erasing programs which keep the blocks world model up to date.
2. A large network of programs which track lines.
3. A large network of programs that uses the group-hypothesize-criticize idea to find otherwise inaccessible properties of hidden objects.
4. A network of programs that jiggles an object if the arm errs too much when placing it.

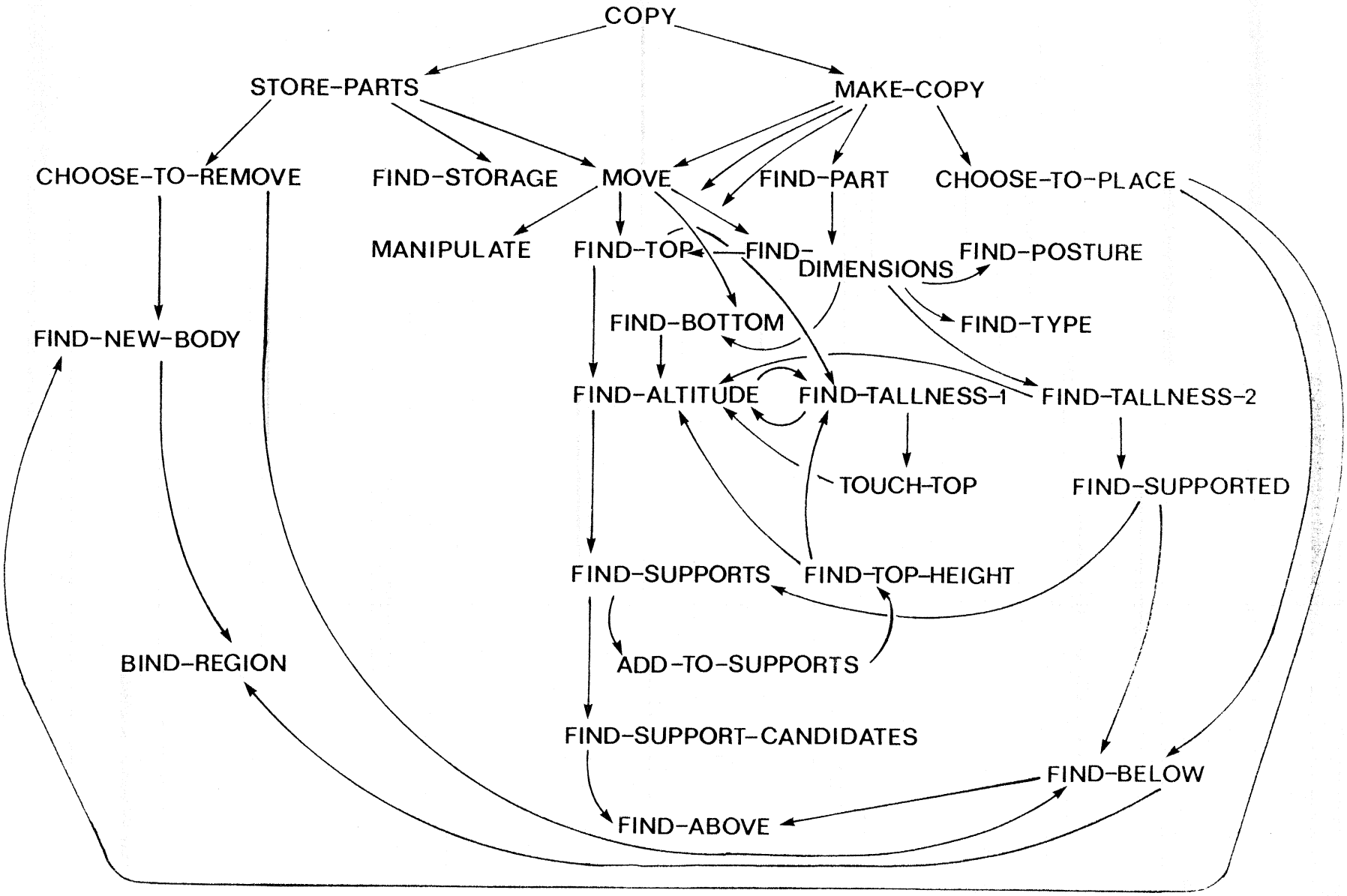


FIGURE 1 The Vision System

THE FUNCTIONS

COPY

As figure 1 shows, COPY simply activates programs that handle the two phases of a copying problem; namely, it calls for the spare parts to be found and put away into the spare parts warehouse area, and it initiates the replication of the new scene.

STORE-PARTS

To disassemble a scene and store it, STORE-PARTS loops through a series of operations. It calls appropriate routines for selecting an object, finding a place for it, and for enacting the movement to storage.

CHOOSE-TO-REMOVE

The first body examined by CHOOSE-TO-REMOVE comes directly from a successful effort to amalgamate some regions into a body using FIND-NEW-BODY. After some body is created, CHOOSE-TO-REMOVE uses FIND-BELOW to make sure it is not underneath something. Frequently, some of the regions surrounding a newly found body are not yet connected to bodies, so FIND-BELOW has a request link to BIND-REGION. (The bodies so found, of course, are placed in the data base and are later selected by CHOOSE-TO-REMOVE without appeal to FIND-NEW-BODY.)

FIND-NEW-BODY

FIND-NEW-BODY locates some unattached region and sets BIND-REGION to work on it. BIND-REGION then calls a collection of programs by Eugene Freuder which do a local parse and make assertions of the form:

(R17 IS-A-FACE-OF B2)
(B2 IS-A BODY)

These programs appeal to a complicated network of subroutines that drive line finding and vertex finding primitives around the scene looking for complete regions.

FIND-BELOW

As mentioned, some regions may need parsing before it makes sense to ask if a given object is below something. After assuring itself that an adjacent region is attached to a body, FIND-BELOW calls the FIND-ABOVE programs to do the work of determining if the body originally in question lies below the object owning that adjacent region.

MOVE

To move an object to its spare parts position, the locations and dimensions are gathered up. Then MANIPULATE interfaces to the machine language programs driving the arm. After MOVE succeeds, STORE-PARTS makes an assertion of the form:

(B12 IS-A SPAREPART)

FIND-TOP

The first task in making the location calculations is to identify line-drawing coordinates of a block's top. Then FIND-TALLNESS and FIND-ALTITUDE supply other information needed to properly supply the routine that transforms line-drawing coordinates to X Y Z coordinates.

FIND-DIMENSIONS

This program uses FIND-TOP to get the information necessary to convert drawing coordinates to three-dimensional coordinates. If the top is totally obscured, then it appeals instead to FIND-BOTTOM and FIND-TALLNESS-2.

SKELETON

SKELETON identifies connected sets of 3 lines which define the dimensions of a brick (Finin WP-19, Finin WP-26). It and the programs under it are frequently called to find instances of various types of lines.

FIND-ALTITUDE

FIND-ALTITUDE determines the height of an object's base primarily by finding its supporting object or objects. If necessary, it will use the arm to try to touch the object's top and then subtract its tallness.

FIND-STORAGE

Once an object is chosen for removal, FIND-STORAGE checks the warehouse area for an appropriate place to put it.

MAKE-COPY

To make the copy, MAKE-COPY, CHOOSE-TO-PLACE, and FIND-PART replace STORE-PARTS, CHOOSE-TO-REMOVE and FIND-STORAGE. Assertions of the form:

(B12 IS-A SPAREPART)

(B2 IS-A-PART-OF COPY)

(B2 IS-ABOVE B1)
are kept up to date throughout by appropriate routines.

CHOOSE-TO-PLACE

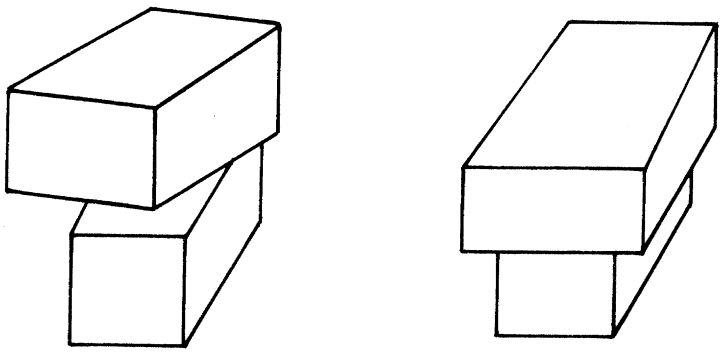
Objects are placed after it is insured that their supports are already placed.

FIND-PART

The part to be used from the warehouse is selected so as to minimize the difference in dimensions of the matched objects.

A SCENERIO

In what follows the scene in figure 2A provides the spare parts which first must be put away in the warehouse. The scene to be copied is that of Figure 2B.



a b

A source of spare parts and a scene to be copied

FIGURE 2

COPY

COPY begins the activities.

STORE-PARTS

STORE-PARTS begins supervision of disassembly.

CHOOSE-TO-REMOVE
FIND-NEW-BODY
BIND-REGION

CHOOSE-TO-REMOVE parses a few regions together into a body, B1. A great deal of work goes into finding these regions by intelligent driving of low level line and vertex finding primitives.

FIND-BELOW
BIND-REGION
FIND-ABOVE

A check is made to insure that the body is not below anything. Note that B2 is parsed during this phase as required for the FIND-ABOVE routines. Unfortunately B1 is below B2 and therefore CHOOSE-TO-REMOVE must select an alternative for removal.

FIND-BELOW
FIND-ABOVE

B2 was found while checking out B1. CHOOSE-TO-REMOVE now notices it in the data base and confirms that it is not below anything.

FIND-STORAGE

FIND-STORAGE finds an empty spot in the warehouse.

MOVE

MOVE initiates the work of finding the location and dimensions of B2.

```
FIND-TOP
  FIND-ALTITUDE
    FIND-SUPPORTS
      FIND-SUPPORT-CANDIDATES
        FIND-TOP-HEIGHT
          FIND-ALTITUDE
            FIND-SUPPORTS
              FIND-SUPPORT-CANDIDATES
                FIND-TOP-HEIGHT
          FIND-TALLNESS-1
      FIND-TALLNESS-1
```

FIND-BOTTOM proceeds to nail down location parameters for B2. As indicated by the depth of call, this requires something of a detour as one must first know B2's altitude, which in turn requires some facts about B1. Note that no calls are made to FIND-ABOVE routines during this sequence as those programs previously were used on both B1 and B2 in determining their suitability for removal.

FIND-DIMENSIONS

A call to FIND-DIMENSIONS succeeds immediately as the necessary facts for finding dimensions were already found in the course of finding location. Routines establish that B2 is a lying brick.

MANIPULATE

MANIPULATE executes the necessary motion.

CHOOSE-TO-REMOVE
FIND-BELOW
FIND-STORAGE

B2 is established as appropriate for transfer to the warehouse. A place is found for it there.

MOVE
FIND-TOP
FIND-DIMENSIONS
MANIPULATE

The move goes off straightforwardly, as essential facts are in the data base as side effects of previous calculations.

CHOOSE-TO-REMOVE
FIND-NEW-BODY

No more objects are located in the scene. At this point the scene to be copied, figure 2, is placed in front of the eye and analysis proceeds on it.

MAKE-COPY
CHOOSE-TO-PLACE
FIND-NEW-BODY
BIND-REGION

B3 is found.

FIND-BELOW
BIND-REGION
FIND-ABOVE

B3 is established as ready to be copied with a spare part.

FIND-PART
FIND-DIMENSIONS
FIND-TOP

Before a part can be found, B3's dimensions must be found. The first program, FIND-TOP, fails.

FIND-BOTTOM
FIND-ALTITUDE
FIND-SUPPORTS
FIND-SUPPORT-CANDIDATES
FIND-TOP-HEIGHT

FIND-DIMENSIONS tries an alternative for calculating dimensions. It starts by finding the altitude of the bottom.

FIND-TALLNESS-2
FIND-SUPPORTED
FIND-BELOW
FIND-ABOVE
FIND-SUPPORTS
FIND-SUPPORT-CANDIDATES

FIND-TALLNESS-2 discovers B4 is above B3.

FIND-ALTITUDE
TOUCH-TOP
FIND-TALLNESS-1

FIND-ALTITUDE finds B4's altitude by using the hand to touch its top subtracting its tallness. B3's height is found by subtracting B3's altitude from that of B4.

```
MOVE
  MANIPULATE
```

Moving in a spare part for B3 is now easy. B3's location was found while dealing with its dimensions.

```
CHOOSE-TO-PLACE
  FIND-BELOW
FIND-PART
  FIND-DIMENSIONS
    FIND-TOP
MOVE
  MANIPULATE
```

Placing a part for B4 is easy as the essential facts are now already in the data base.

```
CHOOSE-TO-REMOVE
  FIND-NEW-BODY
```

No other parts are found in the scene to be copied. Success.

SOME RELEVANT READING

Clowes, M. "On Seeing Things." Artificial Intelligence, 2, No. 1 (1971).

Huffman, D. "Impossible Objects as Nonsense Sentences." Machine Intelligence 6. Ed. D. Michie and B. Meltzer. Edinburgh: Edinburgh University Press, 1971.

SHIRAI'S HETERARCHICAL LINE DRAWING SYSTEM

The heterarchical scene copying system was heterarchical only down to the level of work with a line drawing produced by some other program. Shirai's line-drawing system is therefore another step toward heterarchy in vision inasmuch as it attempts to being high level knowledge about what can possibly be seen in the blocks world down to the level of understanding picture intensity arrays.

The system's high level knowledge largely consists of knowledge about where lines are likely to be. Shirai cites these heuristics:

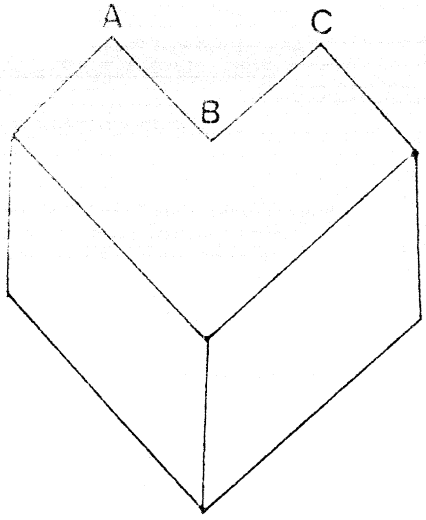
1. If two boundary lines make a concave point (such as point B in figure 1), try to find colinear extensions of them. If only one extension is found, track along this line. Most of such cases are like the one in figure 1B where one body hides the other. It is easy to see to which body this line belongs.

2. If no extensions of two concave lines are found, try to find another line which starts from the concave point. If only one line is found, track along this line. Most of these cases are as in figure 1C where it is not clear locally to which body this line belongs. In figure 1C, line BD belongs to the upper body, but this is not always true. That is, the lines AB, BC, BD are not sufficient to decide.

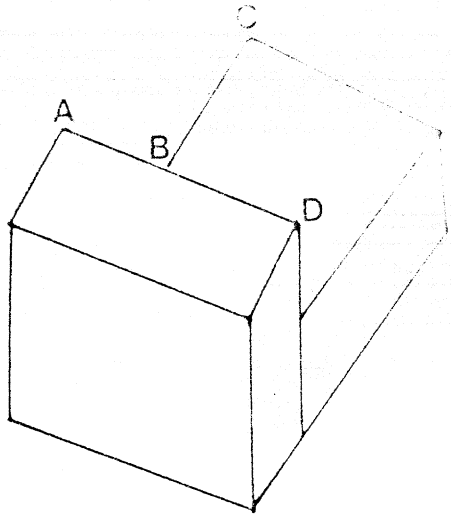
3. If both extensions of the two lines are found at a concave point as in figure 1D, try to find a third one. If only one line is found, track along this line.

Whenever tracking terminates, an attempt is always made to connect the new line to the other lines that were already found. If more than one line segment is found in (1), (2) or (3), the tracking of those lines is put off hopefully to be clarified by the results obtained in simpler cases. Figure 2 illustrates two extensions found at concave point P. The interpretation of the two lines is put off to treat simpler cases first. That is, one would continue examining the contour and lines AB and CD might be found next; then, by a circular search at point B (which is explained later), line BP would be found. At this stage it is easier to interpret lines AB and BP as boundary lines which separate two bodies. Then line DP would be found similarly and interpreted correctly.

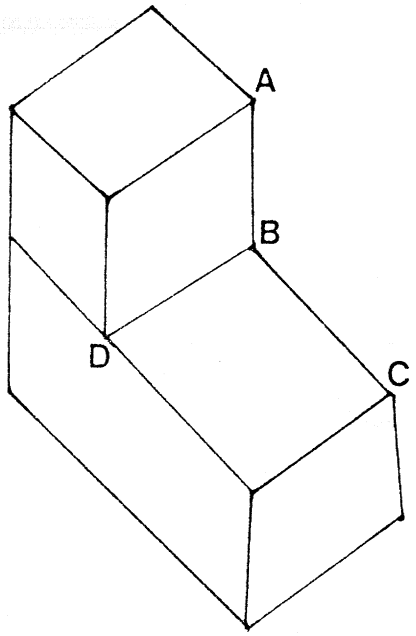
4. If an end of a boundary line is left unconnected as PQ in figure 3, try to find the line starting from the end point (Q in this example) by circular search. If multiple



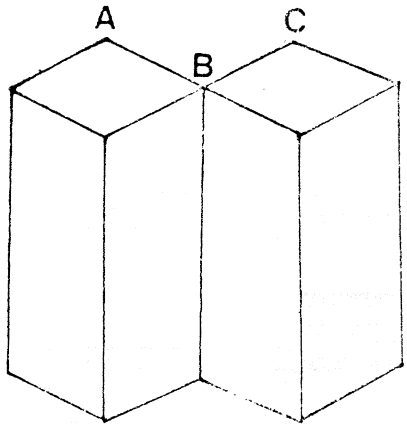
(a)



(b)



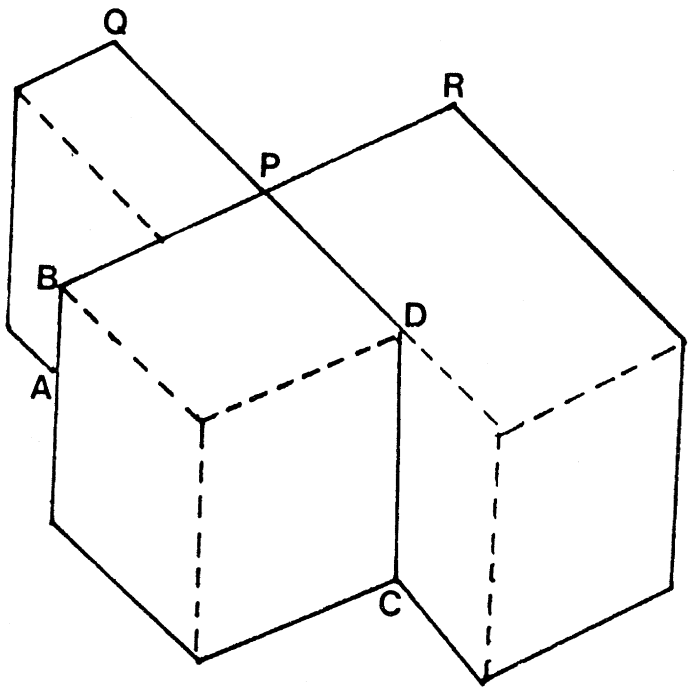
(c)



(d)

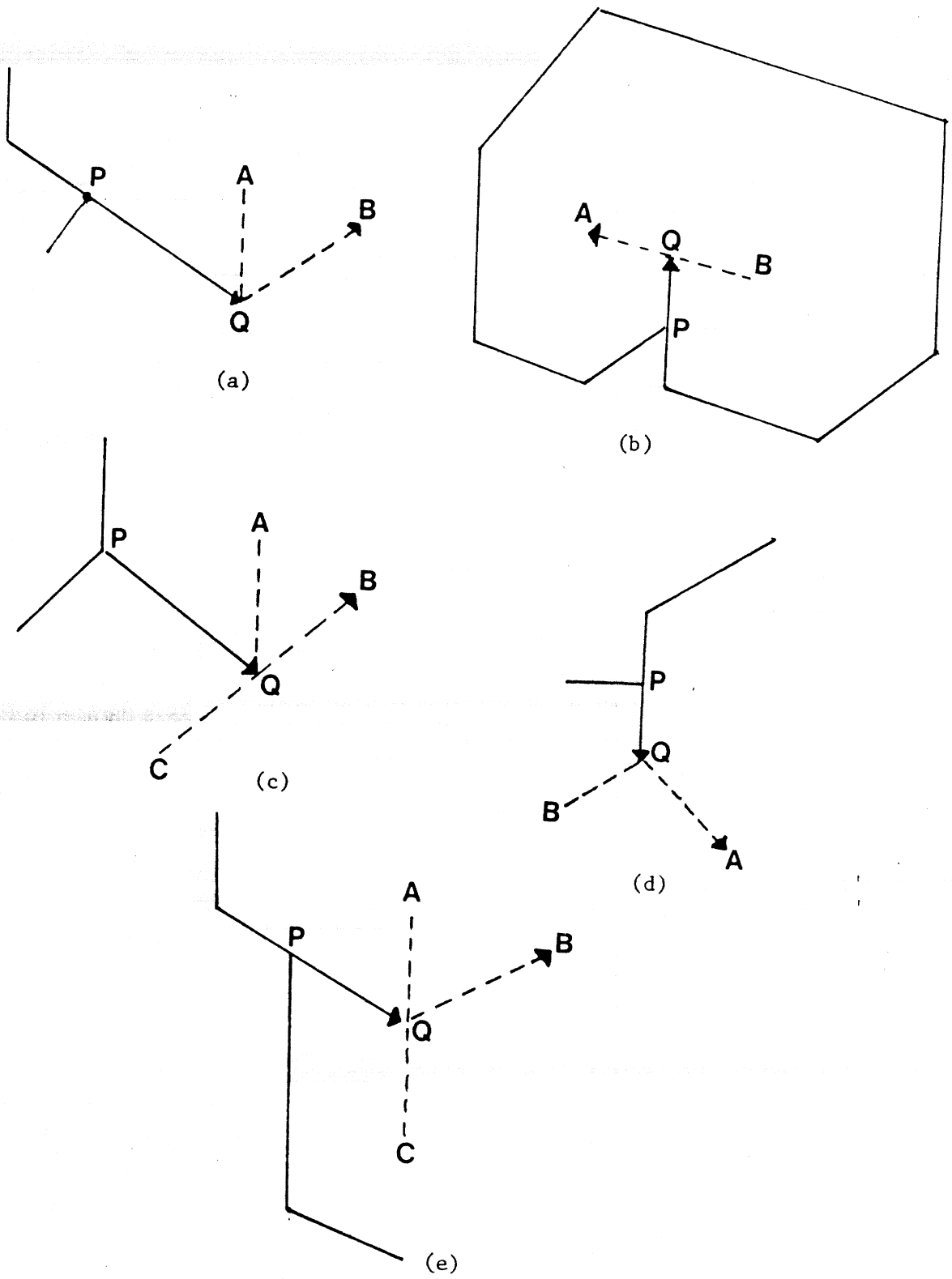
EXAMPLES OF CONCAVE BOUNDARY LINES

FIGURE 1



ILLUSTRATES TWO EXTENSION LINES AT CONCAVE POINT P

FIGURE 2



EXAMPLES OF LINE CONFIGURATION FOUND BY CIRCULAR SEARCH

FIGURE 3

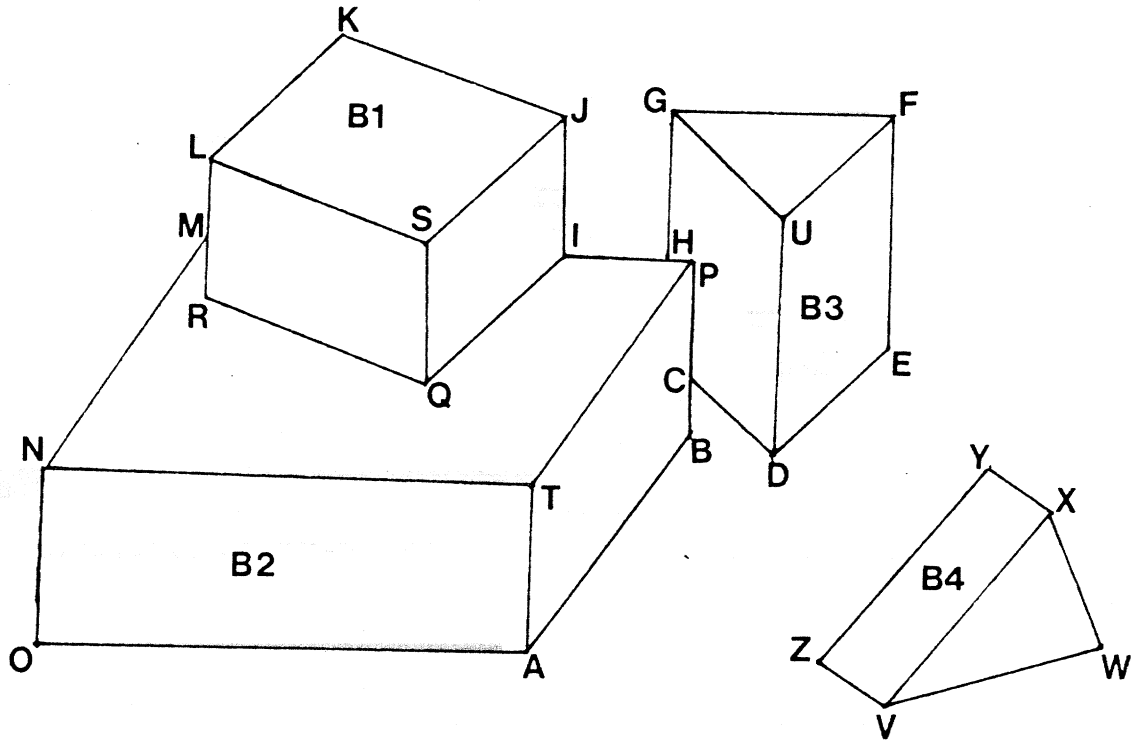
lines are found, try to decide which line is the boundary. If a boundary line is determined, track along it. In this figure, dotted lines are found by circular search and the arrows show the boundary lines to be tracked.

5. If no line is found by the circular search, extend the line (PQ in this example) by a certain length and test if the line is connected to other lines. If not, then apply circular search again as in (4). This is necessary because the termination point of the tracking is not always precise. Note that this process can be repeated until successful (that is either the line is connected to other lines or line segments are found by circular search).

6. If the boundary lines of a body are known, select the vertices of the boundary that might have internal lines starting from them. The selection of vertices is based on heuristics such as selecting upper right vertex rather than lower right vertex. At each vertex, try to find an internal line which is nearly parallel to other boundary lines. If one line is found, track along it. In figure 4, for example, internal line JS is parallel to the boundary lines KL and IQ, and QS is parallel to RL and IJ. Line FU is parallel to ED and XV is parallel to YZ. Thus it is often useful to search for internal lines parallel to boundary lines of the same body. Note that this search for parallels covers only a small area.

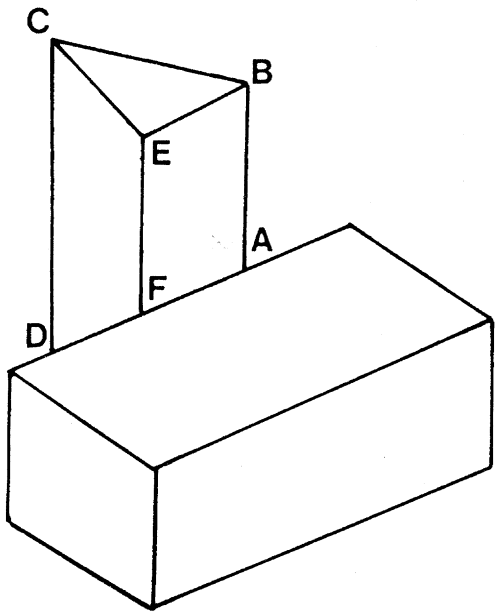
7. If no line is found in (6), try to find one by circular search between adjacent boundary lines. When one line is found, track along it. In figure 5, circular search between BA and BC is necessary to find the internal line BE.

8. If two internal lines meet at a vertex, try to find another internal line starting at the vertex. This process is useful in two cases. One is where no internal line was found in (7) because of little difference in brightness between adjacent faces. Suppose in figure 4, that the internal line SJ was not found at vertex J, but that LS and QS were found. Then try to find an internal line starting at S toward J. If there is enough contrast near S, a line segment is found. The other case is where a body is partly hidden by other bodies. In figure 5, the triangular prism is partly hidden. After BE and CE are found, EF is searched for. Sometimes the direction of the line is predictable and sometimes it is not. If it is predictable, then try that direction. If it is unpredictable or if the predicted direction failed, then apply circular search between the two internal lines. If one line is found, track along it.



EXAMPLE OF SCENE

FIGURE 4



EXAMPLE OF CIRCULAR SEARCH FOR INTERNAL LINES

FIGURE 5

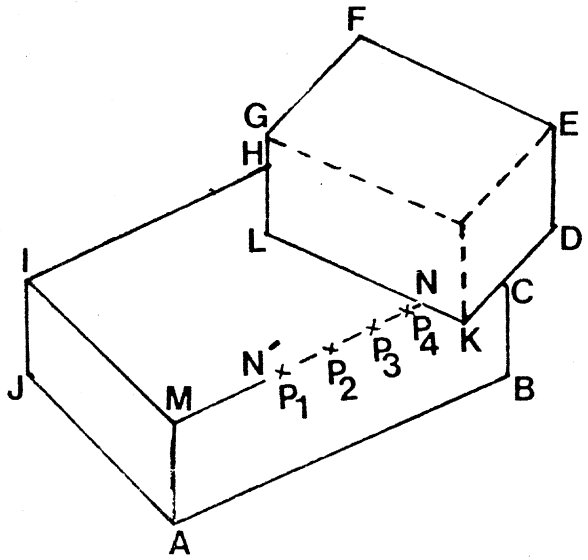
9. If an end of an internal line is not connected to any line, try to find lines starting from the end by circular search. If lines are found, track along them one by one.

10. If no line is found in (9), extend the line by a certain length as in (5) and test if it is connected to other lines. If not connected, try circular search again as in (9). This process can also be repeated until successful. Figure 6 illustrates this process. In figure 6A, line MN' is not connected to others at N', thus step (9) is tried at N' and fails. The line is extended to P1 and step (9) is again applied. This process is repeated until the line is connected to line KL at N. Figure 6B shows that line HI is extended by this process to P2 where a new line is found by circular search. Similarly line CG is extended to P4. This helps to find bodies sitting on obscured edges.

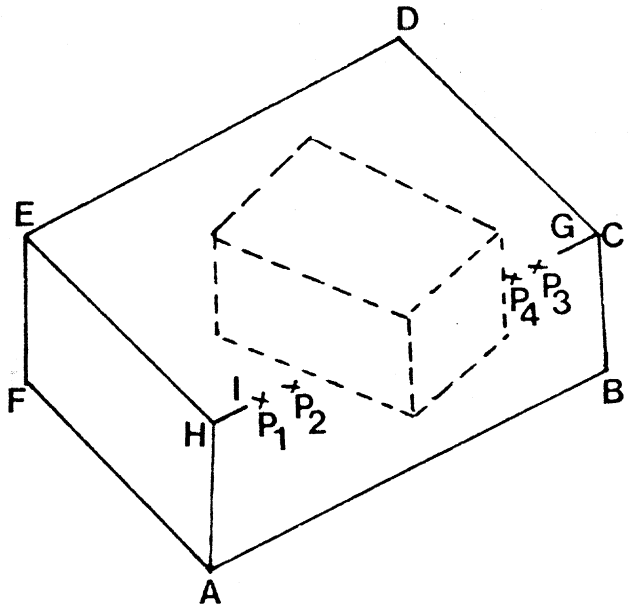
Whenever one of the above steps is finished, considerable information is stored. It is then available to help guide further search. For instance, if tracking along a line terminates, a test is made to see if the line is an extension of other known lines or if the line is connected to a known vertex. If a new boundary line connects two known boundary lines, the body is split into two. In figure 7, for example, line N'P' is obtained by tracking from point N. This line is interpreted as an extension of HN, and HN and N'P' are merged into one straight line using the equations of these two lines. It is then connected to CO and figure 7B is obtained. Before the line was connected to CO, there were two bodies B1 and B2 as in figure 7A. Now body B2 is split into two bodies, B2 and B3. We can interpret line NO as the boundary of B3 which hides a part of B2. Other properties of lines and vertices are obtained similarly at this stage.

AN EXAMPLE OF IMAGE ANALYSIS

Figure 8 illustrates the entire line-finding procedure. At first, the contour lines AB, BC, CD, DE, EF, FG, GH, HI, IJ, JK and KA are obtained as shown in figure 8A. Step (1) described in the previous section is tried for the concave points G and J. In this example, the position of G is not precise enough to find the extension of FG. On the other hand, a line segment is found as an extension of the line KJ. KJ is extended by tracking as far as L. Because there is no other point to which step (1) is applicable, step (2) is tried for point G. One line segment is found and extended until tracking terminates. Thus a line G'M' is obtained as in figure 8B. This line is interpreted as an extension of FG and connected to JL. Then the position of point F, G, L are adjusted to as shown in



(a)



(b)

(DOTTED LINES ARE NOT YET FOUND IN THIS STAGE)

EXAMPLES OF LINE VERIFYING BY CIRCULAR SEARCH

FIGURE 6

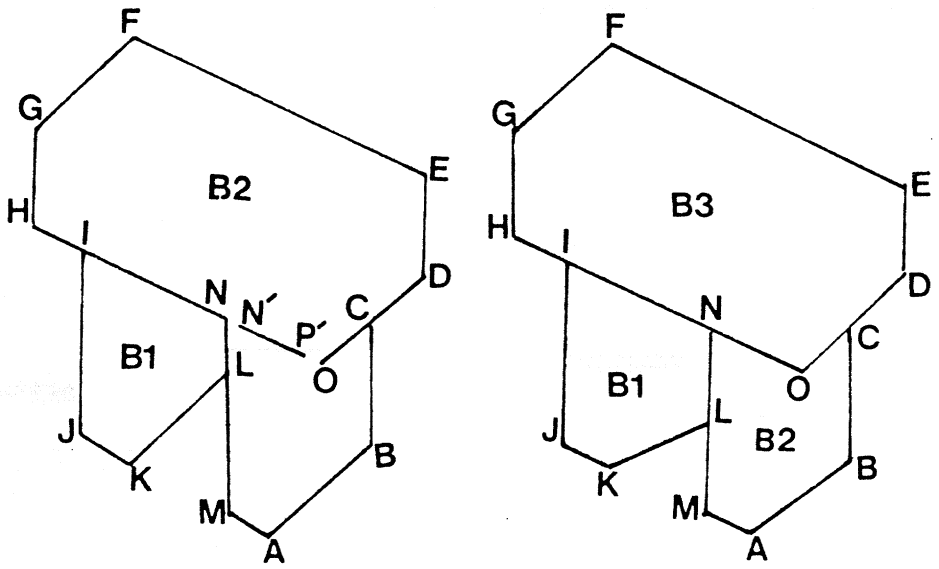


ILLUSTRATION OF THE PROCESS AFTER TRACKING

FIGURE 7

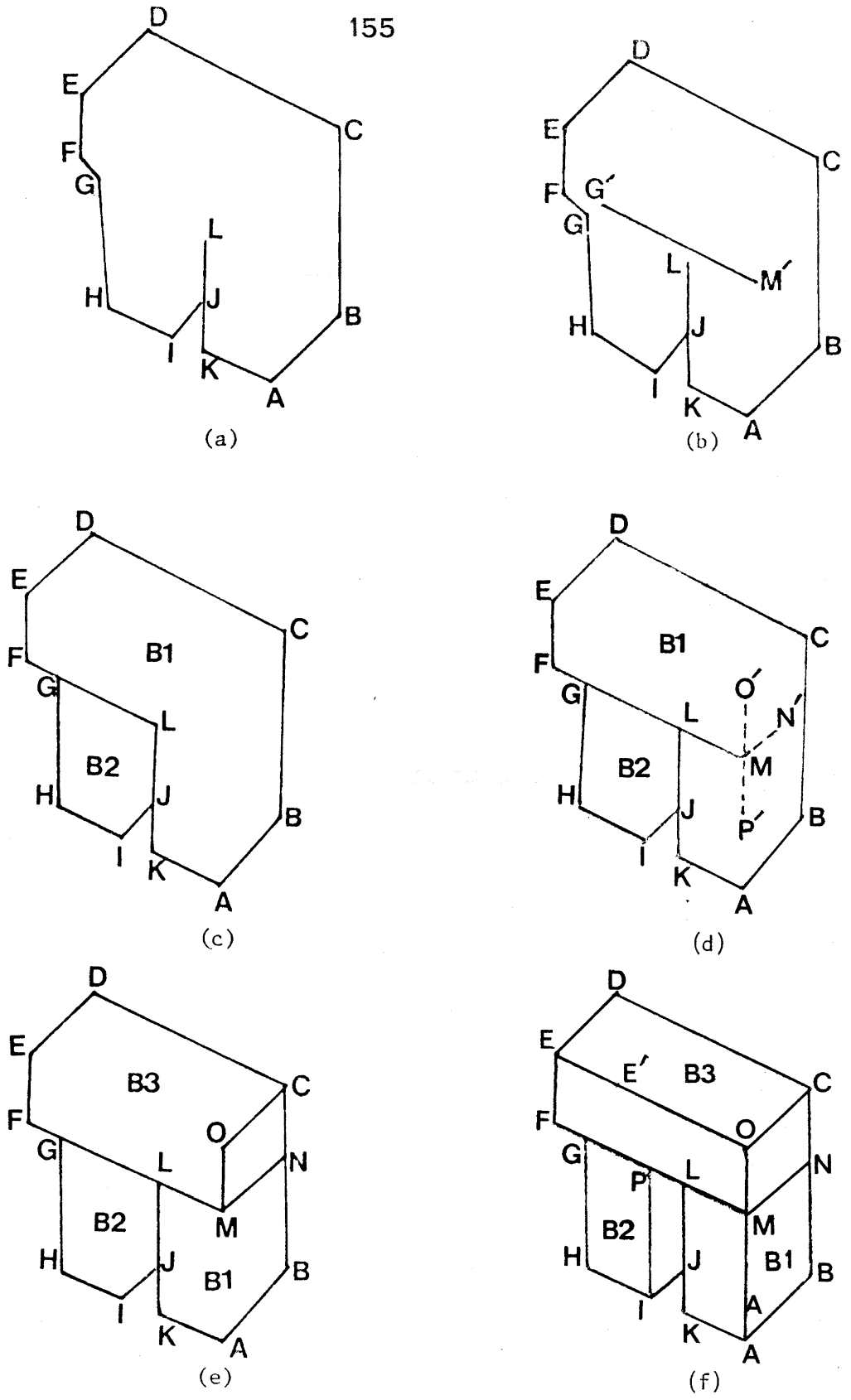


ILLUSTRATION OF THE PROCEDURE TO FIND LINES

FIGURE 8

figure 8C. Now two bodies B1 and B2 are created by the boundary lines GL and JL. It is important to notice this, for it means that step (1) is again applicable at this stage to point L. Thus line FL is extended as far as M in figure 8D (note that line MN' has not yet been found). LM is interpreted as an extension of FL but the end point M is not connected to any other lines. Thus vertices F, G, L and end point M are adjusted considering the new line LM. Here neither step (1), (2) nor (3) is applicable, so that (4) is now applied to M. Three lines are found by circular search as figure 8D. MN' is determined as a boundary line and extended by tracking. When it terminates, the line is connected to boundary line BC at vertex N as in figure 8E. Body B1 splits into bodies, B1 and B3. It is known at this stage that B1 is hidden by B3, and B2 is hidden partly by B3 and partly by B1. Next, step (6) is applied to each body sequentially, selecting first the body for which it is easiest to propose the internal lines. In this example, the order is B3, B1, B2 because B3 hides B1 which hides B2. Internal lines CO and MO are found and connected at vertex O, but no line segment is found using step (6) and (7) applied to vertex E (this stage is shown in figure 8E). Step (8) is applied to vertex O and a line segment toward E is found. This is extended by tracking as far as E' as in figure 8F. Line OE' fails to be connected to any other lines which activates step (10). After a few trials, OE' is extended to connect to vertex E. Similarly, internal line AM is obtained for body B1 and line IP is obtained for B2. When every step has finished, the three bodies and the relationships between them have been discovered.

LERMAN AND WOODHAM'S TRACKING PRIMITIVES

It is clear from the previous section that Shirai's heterarchical system depends on three basic vision primitives:

(i) given an initial point and an estimated initial direction, TRACK a line in that direction until the line terminates.

(ii) given two points, VERIFY the existence of a line joining those two points.

(iii) given the location of a vertex, FIND-SUSPECT directions for possible lines emanating from that vertex.

Lerman and Woodham have improved Shirai's early versions of these programs. This section describes their results in some detail in order to give an impression of the detail and complexity involved at this level of vision processing.

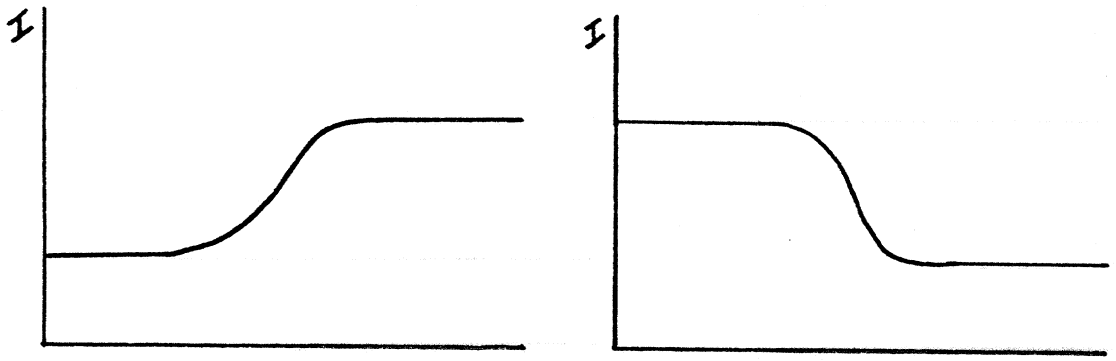
CHARACTERIZING FEATURE POINTS

A feature point is by definition a localized property of the image intensity array. Rather than consider the various 2-dimensional detection predicates one might apply to image points, we simplify the problem by characterizing feature points according to a profile of intensity values along a linear band perpendicular to the direction of the line for which this point is a feature point.

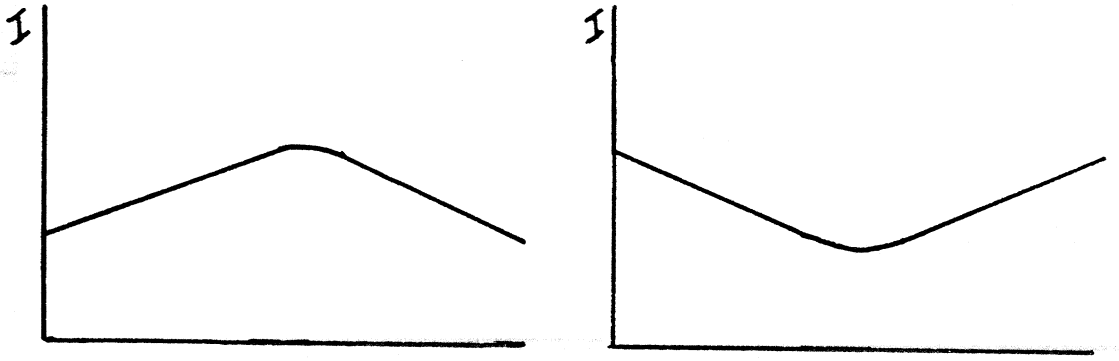
Not all real edges in a scene have similar cross-sectional intensity profiles. Herskovits and Binford classified edges into three types according to the three characteristic intensity profiles depicted in figure 1. Step-type edges occur at contrast boundaries between regions of relatively homogeneous intensity. In Blocks World scenes, the light source can often induce a varying intensity across a region. Roof-type edges occur at boundaries between regions whose intensity profiles vary almost symmetrically across the boundary. Finally, edge-effects occur at boundaries representing a sharp highlight (or, in the inverse sense, at boundaries representing a sharp 'lowlight' -- typically at a crack where one object is stacked upon another).

If the actual intensity profiles were as clean and smooth as in figure 1, we could immediately accept the above characterization of feature points and concern ourselves with implementing a simple-minded recognition algorithm based upon that

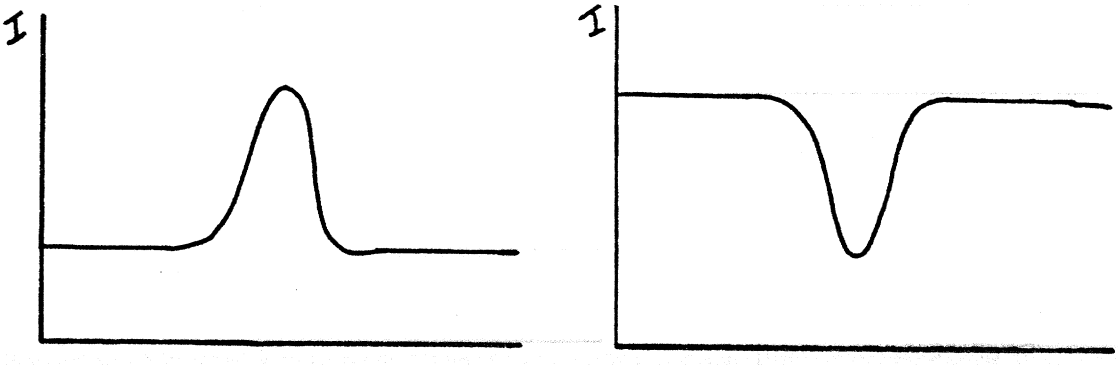
HERSKOVITS-BINFORD INTENSITY PROFILE
CHARACTERIZATIONS



(a) Step



(b) Roof



(c) Edge-Effect

FIGURE 1

characterization. However, as one might expect, intensity profiles are generally quite noisy.

In our effort to characterize feature points, we seek some filter (predicate) to apply to the actual intensity profile which has the following attributes:

(i) the variance in filter values due to noise is minimized.

(ii) the variance in filter values due to the actual edge is maximized.

(iii) the filter (predicate) is computationally simple to apply.

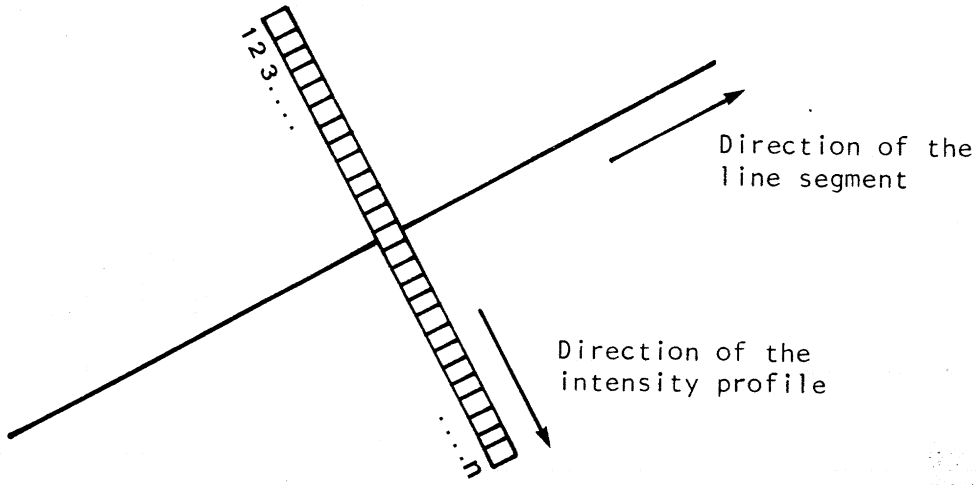
Shirai's contrast function for intensity profiles is remarkably successful according to the above three criteria. The Shirai filter is conceptually quite simple. Consider an intensity profile of n points taken along a band perpendicular to the direction of the line as shown in figure 2A. We define the filter-value of the i profile point, as in figure 2B, to be the difference between the sum of the m subsequent intensity points and the sum of the m preceding intensity points, where m is some parameter. (Thus, there are (n-2m) points for which the contrast function is well defined.)

In figure 3, we show the filter-value profiles corresponding to the various edge-types depicted in figure 1. For the purposes of the TRACK program, it has not been necessary to differentiate between edge-effects and roof-type edges. The same detection process works well for both types of lines. Thus, in the TRACK program, we characterize a feature point into one of four classes:

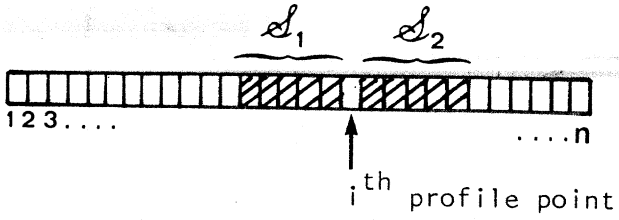
(i) a MAXIMUM is a feature point representing a step-type edge where the intensity profile crosses from a region of relative brightness to a region of relative darkness.

(ii) a MINIMUM is a feature point representing a step-type edge where the intensity profile crosses from a region of relative darkness to a region of relative brightness.

(iii) a HILITE is a feature point representing an edge-effect (or a roof-type edge) where the intensity profile crosses a highlight (ie. a boundary of relative brightness).



(a) Taking an intensity profile



$$\mathscr{I}_1 = \sum_{r=i-m}^{i-1} I_r \qquad \mathscr{I}_2 = \sum_{r=i+1}^{i+m} I_r$$

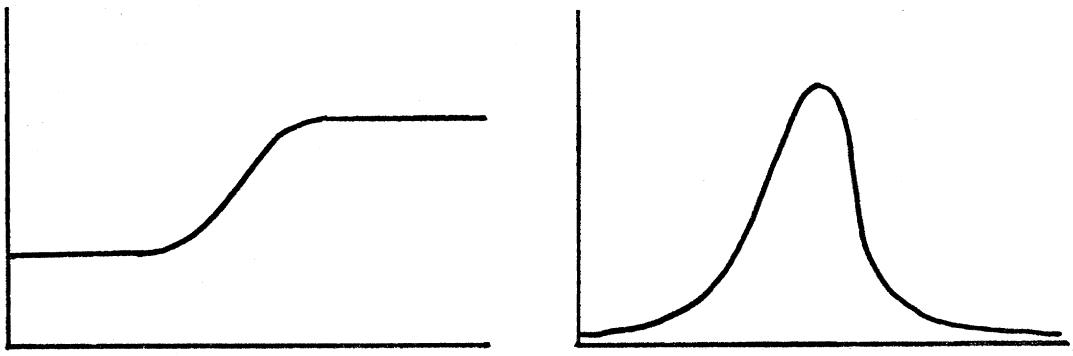
The Shirai contrast function at the i^{th} profile point is F(m) where

$$Fm(i) = \mathscr{I}_1 - \mathscr{I}_2$$

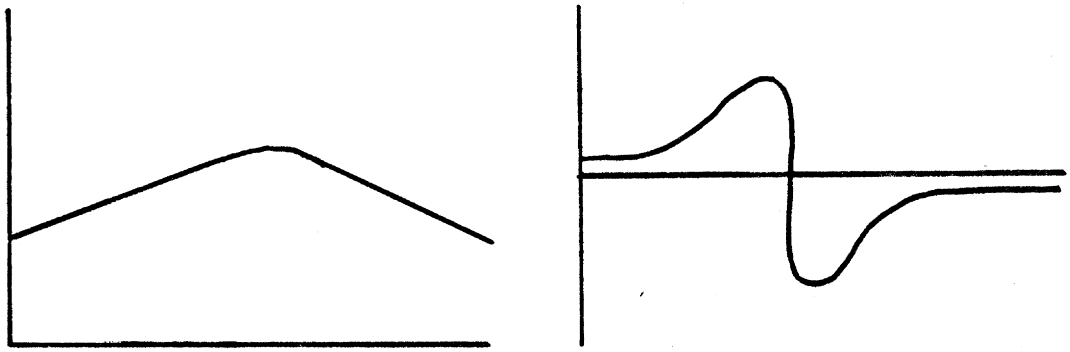
(b) Defining the Shirai contrast function

FIGURE 2

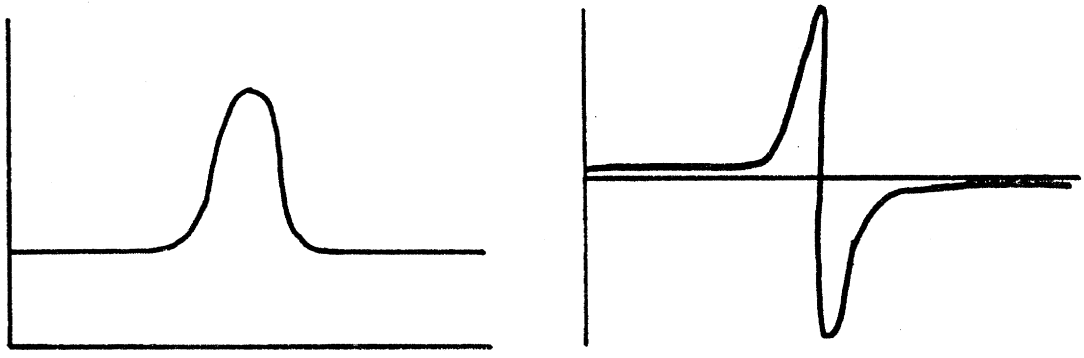
SHIRAI FILTER-VALUE PROFILES



(a) Step



(b) Roof



(c) Edge-Effect

FIGURE 3

(iv) a CRACK is a feature point representing an edge-effect (or a roof-type edge) where the intensity profile crosses a "lowlight" (ie. a boundary of relative darkness).

We characterize a particular line segment according to the feature points which are found to lie on that line segment. Thus, a line segment is of type MAXIMUM, MINIMUM, HILITE or CRACK respectively as its feature points are of type MAXIMUM, MINIMUM, HILITE or CRACK. We note in passing that reversing the direction in which a particular line segment is tracked causes a MAXIMUM line segment to become a MINIMUM (and vice-versa).

In figure 4 and 5 we show the intensity profiles and filter-value profiles across two edges from a typical stored picture.

RECOGNIZING FEATURE POINTS

Applying the Shirai contrast function to an intensity profile gives us a filter-values profile which is relatively free from noise and whose shape is characteristic of the type of feature point being considered. However, we are still faced with the following questions:

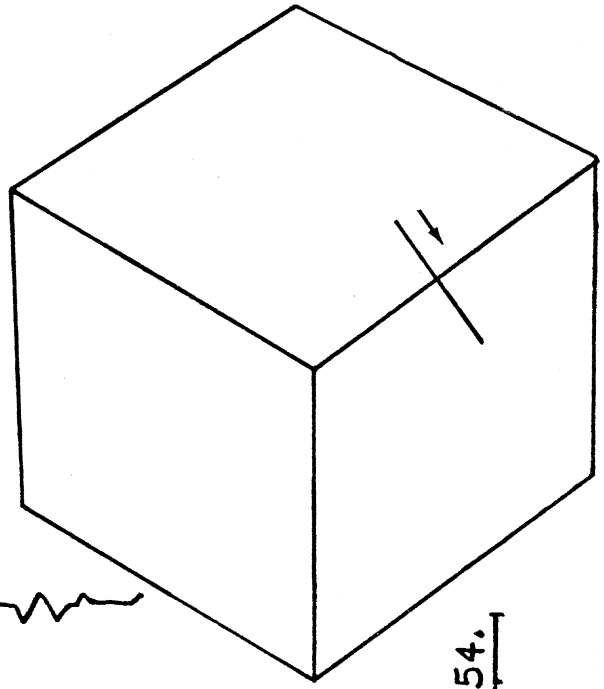
Given an arbitrary filter-values profile, how do we decide whether or not it represents the profile of a feature point?

Given that a filter-values profile represents that of a feature point, how do we decide the type of the feature point and how do we locate the feature point in the profile?

In this section we present the algorithm used in the TRACK program package to analyze the filter-value profiles.

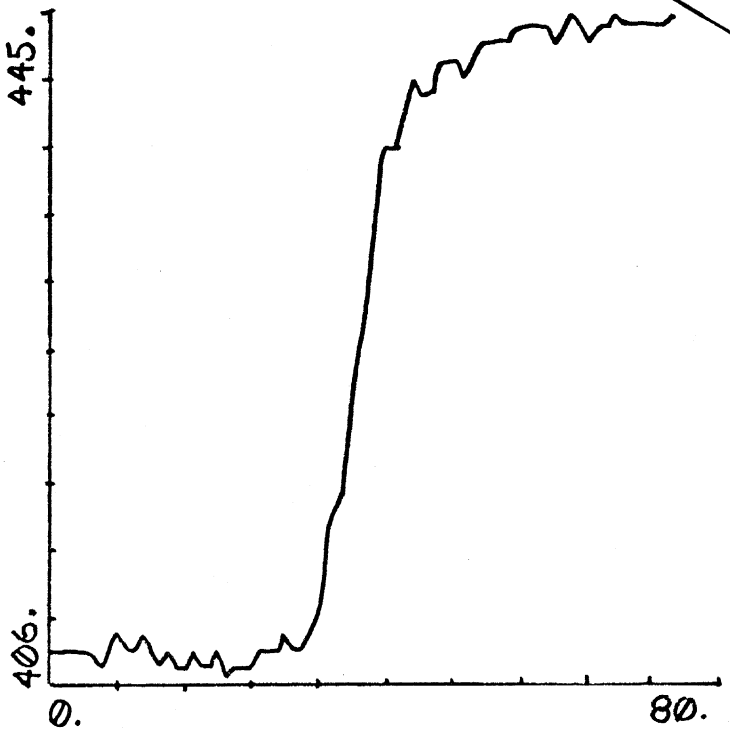
The general algorithm for finding peaks in the filter-values profile is summarized as follows:

(1) Beginning at the leftmost end of the filter-values profile, move to the right until we find an entry whose absolute value is m units greater than the minimum absolute value encountered along the way. Call the position of this entry START



A MAXIMUM FEATURE POINT

VIDI VALUES



FILTER VALUES

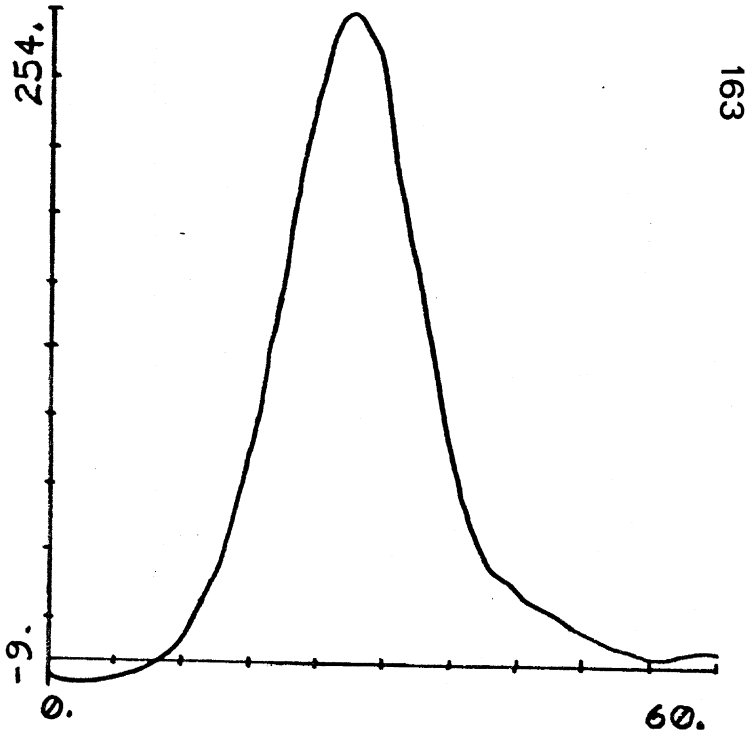


FIGURE 4

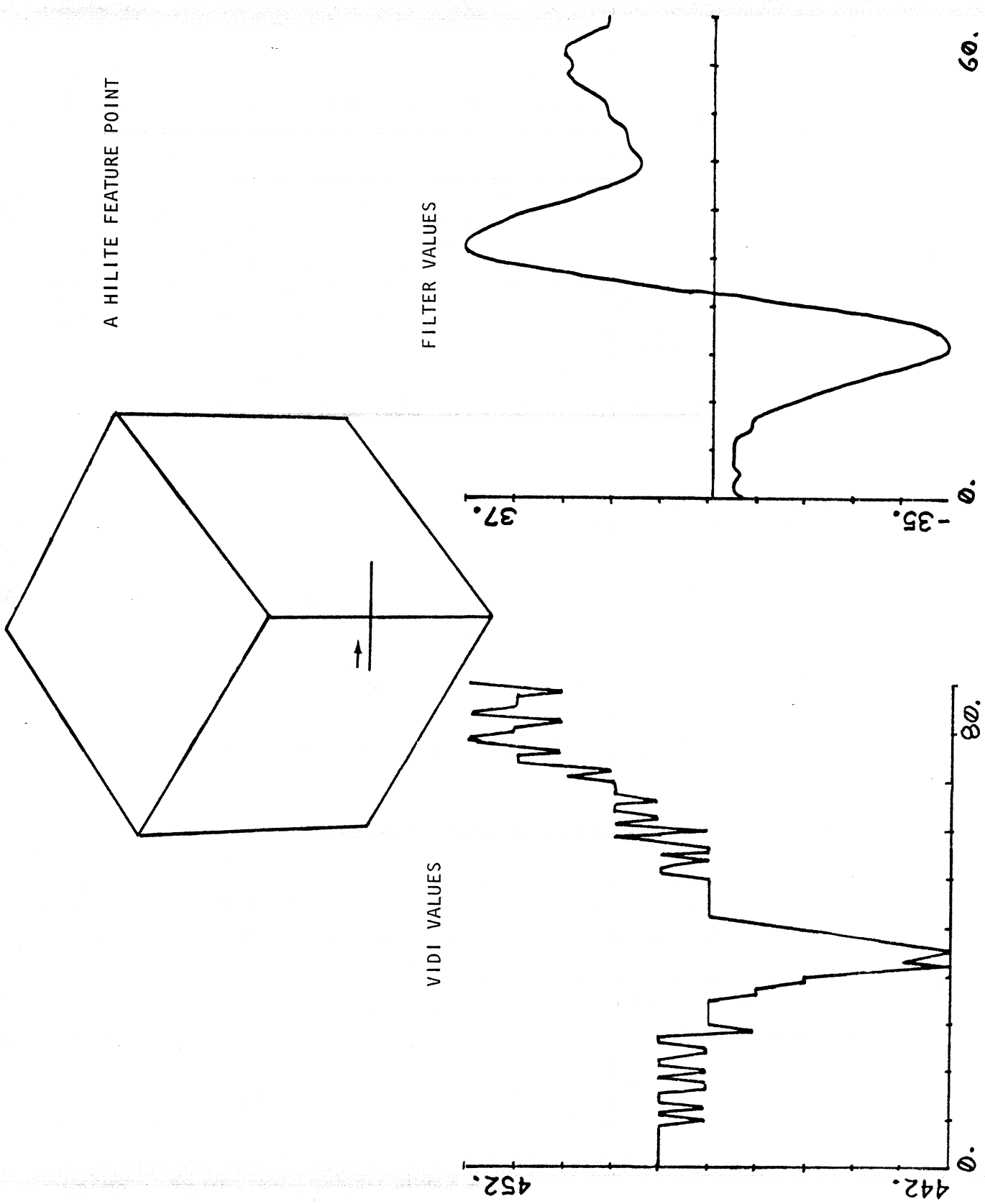

A HILITE FEATURE POINT
FILTER VALUES
37.
-35.
0.
60.
VIDI VALUES
452.
442.
0.
80.

FIGURE 5

and call its value STARTV. If we reach the rightmost end of the filter-values profile before finding such a START, return NIL indicating that there is no feature point along this profile.

(2) Beginning at the rightmost end of the filter-values profile, move to the left until we find an entry whose absolute value is m units greater than the minimum absolute value encountered along the way. Call the position of this entry END and call its value ENDV. If we reach START before finding such an END, return NIL to indicate that there is no feature point along this profile.

(3) Find the maximum and minimum filter-values between START and END. Call these values respectively MAXV and MINV.

(4) If MAXV < m, reject the possibility of there being a MAXIMUM type feature point in this intensity profile.

If both STARTV and ENDV are not less than 90% of MAXV, reject the possibility of there being a MAXIMUM type feature point in this intensity profile.

Otherwise, consider all portions of the filter-values profile between START and END lying above a height 80% of MAXV. Call the midpoint of that portion closest to the line being tracked a MAXIMUM type feature point.

(5) Similarly, if MINV > m, reject the possibility of there being a MINIMUM type feature point in this intensity profile.

If both STARTV and ENDV are not greater than 90% of MINV, reject the possibility of there being a MINIMUM type feature point in this intensity profile.

Otherwise, consider all portions of the filter-values profile lying below a height 80% of MINV. Call the midpoint of that portion closest to the line being tracked a MINIMUM type feature point.

(6) If both MAXIMUM and MINIMUM type feature points have been found in (4) and (5) above, and, if the difference in the height of the two respective peaks is no greater than 75% of the largest peak, call the feature point a HILITE or CRACK

according to whether we have crossed a MINIMUM-MAXIMUM pair or a MAXIMUM-MINIMUM pair. Otherwise, call the feature point a MAXIMUM or MINIMUM according to which of the MAXIMUM or MINIMUM peaks is closest to the line being tracked.

In figure 6 we show a typical filter-values profile analysis. Steps (1) and (2) above

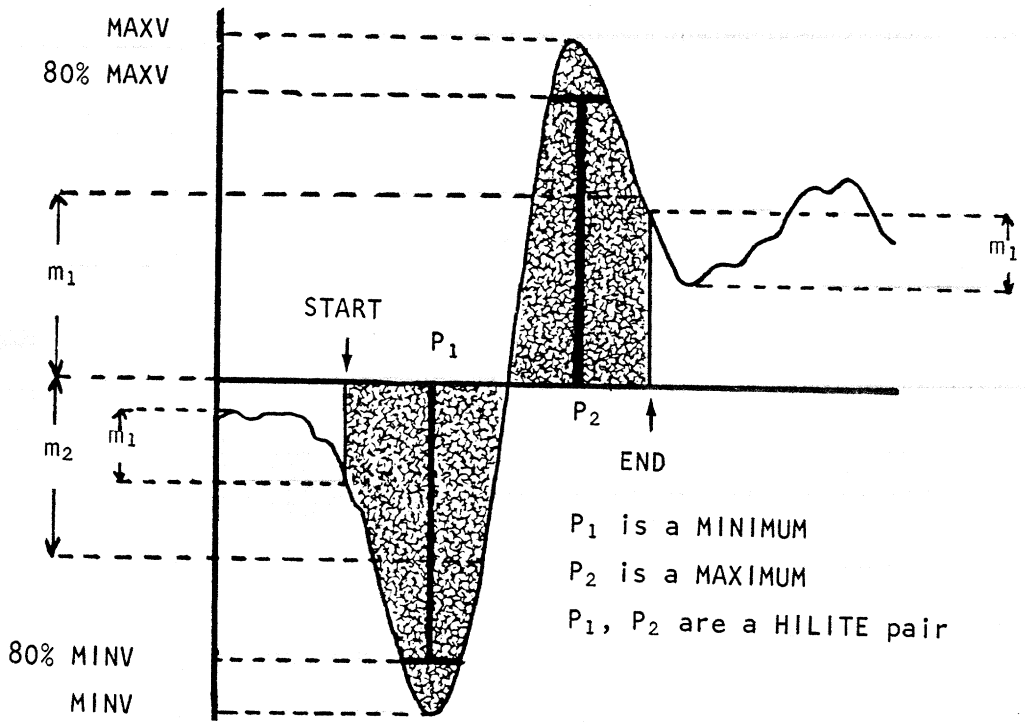


FIGURE 6

are designed to frame the filter-values profile so that the values located between START and END actually represent peaks and gradients due to edge effects. We can think of START and END as defining the 'shoulders' of the peak. In steps (4) and (5), we see that a particular intensity profile can be rejected as representing that of a feature point either because its MAXV/MINV values are not large enough or because the shoulders of that portion of the peak we have captured in our profile are not low enough. (The parameters m and m are estimated from the height of the peaks already

encountered in tracking the current line segment.)

TRACKING A LINE

TRACK-LINE is defined such that, given an initial point and an estimated initial direction, it will track a line in that direction until the line terminates. TRACK-LINE is called by:

(TRACK-LINE X Y DIRN TYPE)

where

X is the x-coordinate of the initial point for the track (0. < X < 1024.)

Y is the y-coordinate of the initial point for the track (0. < Y < 1024.)

DIRN is the direction of the line in degrees where direction is interpreted to be the directed angle the line makes with respect to the positive x-axis (0. < DIRN < 2 π)

TYPE is an optional argument which, if specified, is one of 'MAXIMUM, 'MINIMUM, 'HILITE or 'CRACK indicating the specific type of line to be tracked. (If no TYPE argument is specified, TRACK-LINE will track any type of line it encounters.)

TRACK-LINE returns:

NIL if no line was found (or if no line of the specified type was found).

((x-end-pt y-end-pt) EQUN TYPE)
where x-end-pt and y-end-pt are respectively the x and y coordinates of the end-point of the line segment detected, where EQUN is the three element list representing the equation of the line segment detected and, finally, where TYPE is one of 'MAXIMUM, 'MINIMUM, 'HILITE or 'CRACK indicating the type of the line segment detected.

The TRACK-LINE function consists of three separate parts:

(i) begin a track in the estimated direction to determine if there is, in fact, a line segment to be tracked. At this stage we determine:
    a) the line type (if it is not already specified)
    b) an updated estimate of the line's direction

c) an estimate of the average peak height to be expected in the filter-value profiles across this line.

(ii) follow the line segment until the track is lost.

(iii) find, as accurately as possible, the end-point of the line segment.

BEGINNING A TRACK

The beginning of a track is handled slightly differently depending upon whether or not a line type is specified in the call to TRACK-LINE.

In beginning a track of unknown type, TRACK-LINE examines 5. intensity profiles across points spaced between 15. and 30. vidissector units along the estimated path of the line segment. The analysis of these 5. profiles proceeds as described before. However, rather than attempting to decide a line type associated with each profile, the decision is postponed until all 5. profiles have been examined.

All the MAXIMUM feature points encountered are entered onto a list called MAXLIST. Similarly, the MINIMUM feature points are entered onto a list called MINLIST. In addition, all the MAXIMUM-MINIMUM pairs that qualify as a CRACK are entered onto a list called CRACKLIST. Finally, MINUMUM-MAXIMUM pairs that qualify as a HILITE are entered onto a list called HILIST.

If either CRACKLIST or HILIST has 3. or more entries, the line type is said, respectively, to be a CRACK or a HILITE. Otherwise, if either MAXLIST or MINLIST has 3. or more entries, the line type is said, respectively, to be a MAXIMUM or a MINIMUM. Finally, if none of the feature point lists has 3. or more entries, TRACK-LINE returns NIL indicating that no line segment has been found.

TRACK-LINE now knows the type of the line segment to be tracked. In addition, the position of the feature points encountered provides an updated estimate of the direction of the line segment while the average peak height of the feature points encountered allows us to "tune" the feature point detection process for the particular line segment in question.

FOLLOWING A LINE SEGMENT

Following a line segment can be seen as consisting of two basic operations:

(i) Predicting where a feature point should lie along the line segment.

(ii) Determining whether or not there is an appropriate feature point sufficiently close to the predicted location.

The prediction operation for (i) above is simply stated. Prediction consists of extrapolating the line segment some distance beyond its current end-point using a least-mean-square equation of the line as determined from the feature points already found to lie on the line.

For (ii) above, we again will be applying the feature point recognition algorithm described before. At this point, however, we are left with a certain vagueness to the notions of "appropriate" and "sufficiently close" stated above. Most of the 'hair' associated with the line following algorithm given below is concerned with making precise just what we mean by "appropriate" and "sufficiently close."

Before detailing the algorithm, we begin with a brief discussion of the parameters involved. Suppose that d is the distance in the filter-values profile between the actual feature point and its predicted location. Two special variables D1 and D2 determine our interpretation of the feature point. If d < D1 then we say the feature point is a GOOD POINT (ie. the point definitely lies on the line). If d > D2 then we say the feature point does not lie on the line. If D1 < d < D2 then we say the feature point is an AMBIGUOUS POINT and we defer a decision until subsequent analysis. D2 is constant throughout and its default is D2 = 8.5. D1 varies during the line following algorithm but it is initially set to 3.0.

A number of parameters are associated with terminating conditions for the line following algorithm. The special variables M1 and M2 are used to record the current status of line following. One can think of M1 as counting the number of times we have failed to find a feature point on the line since the last GOOD POINT and of M2 as counting the number of AMBIGUOUS POINTs since the last GOOD POINT. However, the above is only a first approximation since, in fact, the interpretation of M1 and M2 is affected by special variable MA (default value is MA = 1.).

The variable MA determines how willing we are to accept AMBIGUOUS POINTs as GOOD POINTs after having failed, at some previous stage, to find any feature point at all on the line. (Refer to the algorithm below for a precise interpretation.) The follow algorithm will terminate if either M1 > MN or (+ M1 M2) > MT. (The defaults for MN and MT are MN = 3 and MT = 7).

Finally, two special variables serve as parameters for the prediction operation.

STEP (default value is STEP = 10.0) is the basic step size in vidissector units used for extrapolating beyond the current feature point. M is an "enthusiasm factor" used to scale the basic step size according to the number of successive GOOD POINTs we have previously encountered.

We are now ready to summarize the line following algorthm:

(1) Initialize for following a line by SETQ'ing M = 1.0, M1 = 0., M2 = 0. and D1 = D0. Also, initialize XY to be the last feature point found by begin-track and EQ to be the updated equation of the line as determined by begin-track.

(2) SETQ XY to be the point (* M STEP) units past the current XY along the line given by EQ. Obtain an intensity profile centered at XY.

(3) If no feature point of the required type is found, SETQ M1 to (1+ M1) and M to 1.0
Go to (7).

(4) If d < D1, consider the current XY to be a GOOD POINT. If M1 > MA SETQ M1 to (1- M1). Otherwise, consider as GOOD POINTs all feature points previously considered to be AMBIGUOUS POINTs and SETQ M1 to 0., M2 to 0. and D1 to D0. Update EQ to be the least-mean-square equation based upon all feature points now considered as GOOD POINTs. SETQ M to (+ M 0.5).
Go to (7)

(5) If D1 < d < D2, consider the current XY to be an AMBIGUOUS POINT. SETQ M to 1.0, M2 to (1+ M2) and D1 to (+ D0 (* M2 WM)). If M1 > MA, SETQ M1 to (1- M1).
Go to (7)

(6) If d > D2, consider the point to be off the line. SETQ M to 1.0 and M1 (1+ M1).

(7) If M1 < MN or if (+ M1 M2) < MT then go to (2). Otherwise, consider the line segment to be lost and terminate the follow procedure.

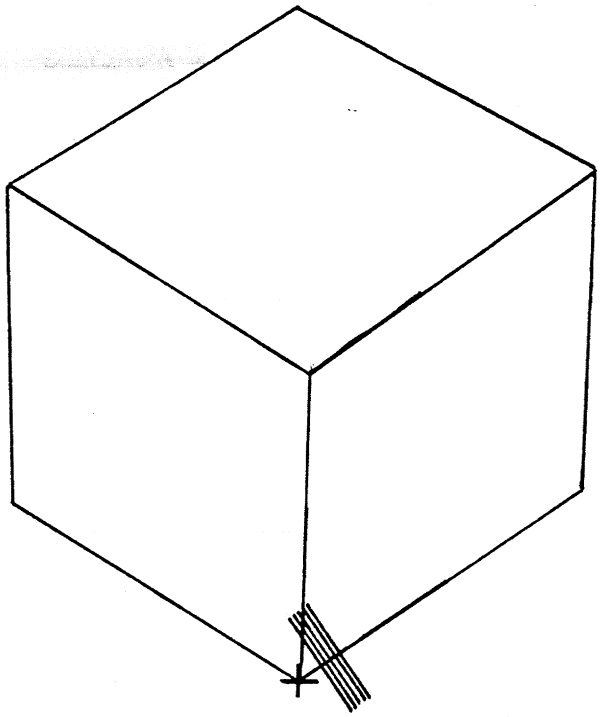
FINDING THE END-POINT OF THE LINE SEGMENT

When TRACK-LINE loses a line segment, we know only that the line segment must have terminated somewhere between the last GOOD POINT obtained and the point examined immediately following that point. We still need to find, as accurately as possible, the end-point of the line segment.

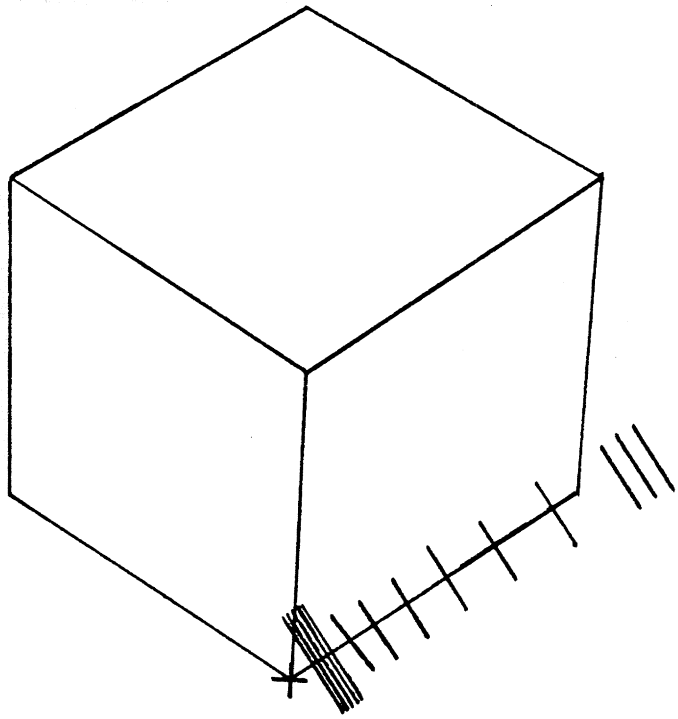
TRACK-LINE finds the end-point of a line segment by employing a simple binary search algorithm. An intensity profile is taken at a point midway between the last GOOD POINT obtained and the subsequent point tried. If that profile reveals a GOOD POINT, its position becomes that of the last GOOD POINT obtained. Otherwise, its position becomes that of the subsequent point tried. We iterate this procedure until the distance separating the last GOOD POINT obtained and the subsequent point tried is not more than 2. vidissector units.

In figure 7 we show an example of TRACK-LINE applied to a line in a stored picture.

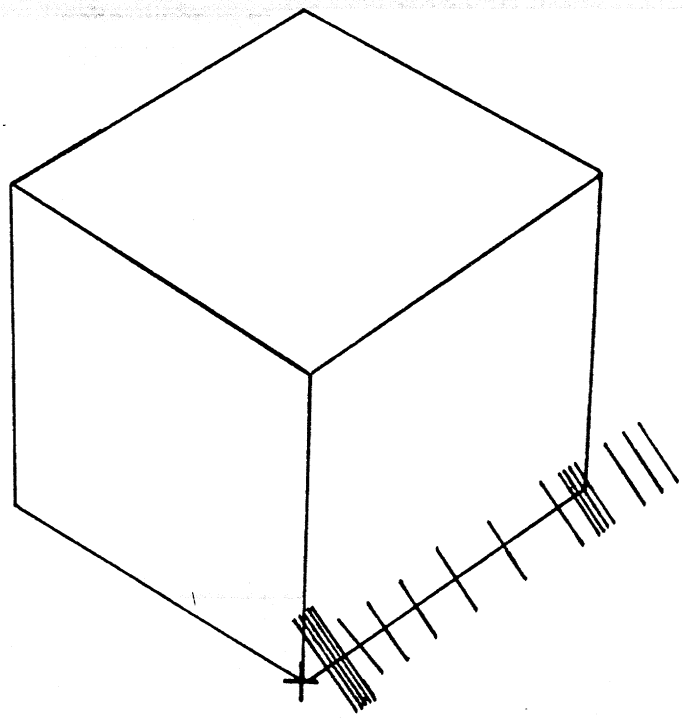
TRACKING A LINE SEGMENT



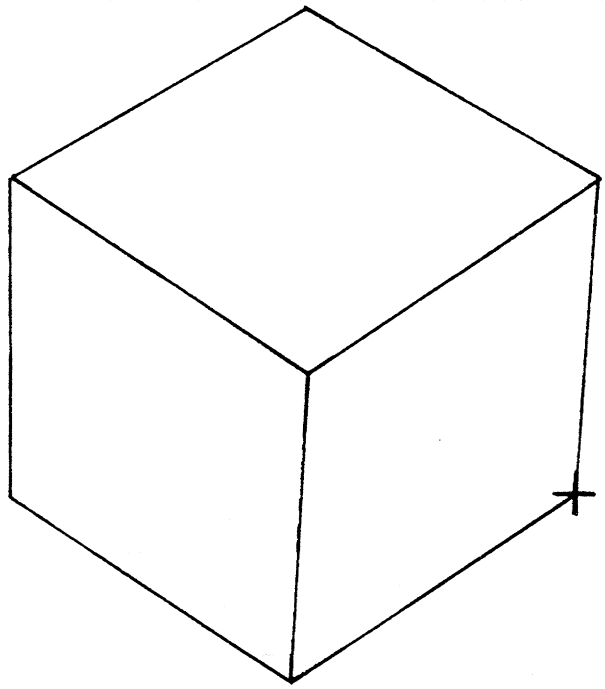
(a) BEGIN-TRACK



(b) FOLLOW-LINE



(c) END-TRACK



(d) END-POINT

FIGURE 7

WALTZ'S SCENE ANALYSIS SYSTEM

We now undertake a three part discussion of how physical constraints can be exploited in vision. In the first of these parts, we see how Waltz was able to use the constraints placed on line drawings by the nature of the three-dimensional blocks world. Waltz's system consists of a working set of computer programs which categorize the lines in a scene into boundary, shadow, convex, concave, and crack types. In addition, the system groups regions which belong to the same object, calculates object orientation, and notices such relations as contact or lack of contact, support, in-front-of, and behind. Figure 1 and figure 2 are typical of the scenes that can be handled.

LINE LABELS

Waltz's first subproblem was to develop a language that allows one to relate the two-dimensional line drawing and the real-world scene. He did this by assigning names called labels to lines in the line drawing, after the manner of Huffman (1971) and Clowes (1971). Thus, for example, in figure 3 line segment J1-J2 is labeled as a shadow edge, line J2-J3 is labeled as a concave edge, line J3-J14 is labeled as a convex edge, line J4-J5 is labeled as an obscuring edge and line J12-J13 is labeled as a crack edge. Thus, these terms are attached to parts of the drawing, but they designate the kinds of things found in the three-dimensional scene.

Pay particular attention to the notation used to label the lines. When we talk of junction lables we refer to the various possible combinations of such line labels around a junction. Each such combination is thought of as a particular junction label.

KINDS OF KNOWLEDGE

The knowledge of this system is expressed in several distinct forms:

(1) A list of possible junction labels for each type of junction geometry includes the a priori knowledge about the possible three-dimensional interpretations of a junction.

(2) Selection rules which use junction geometry, knowledge about which region is the table, and region brightness. These can easily be extended to use line segment directions to find the subset of the total list of possible junction labelings which could apply at a particular junction in a line drawing.

(3) A program to find the possible labelings; it knows how to systematically

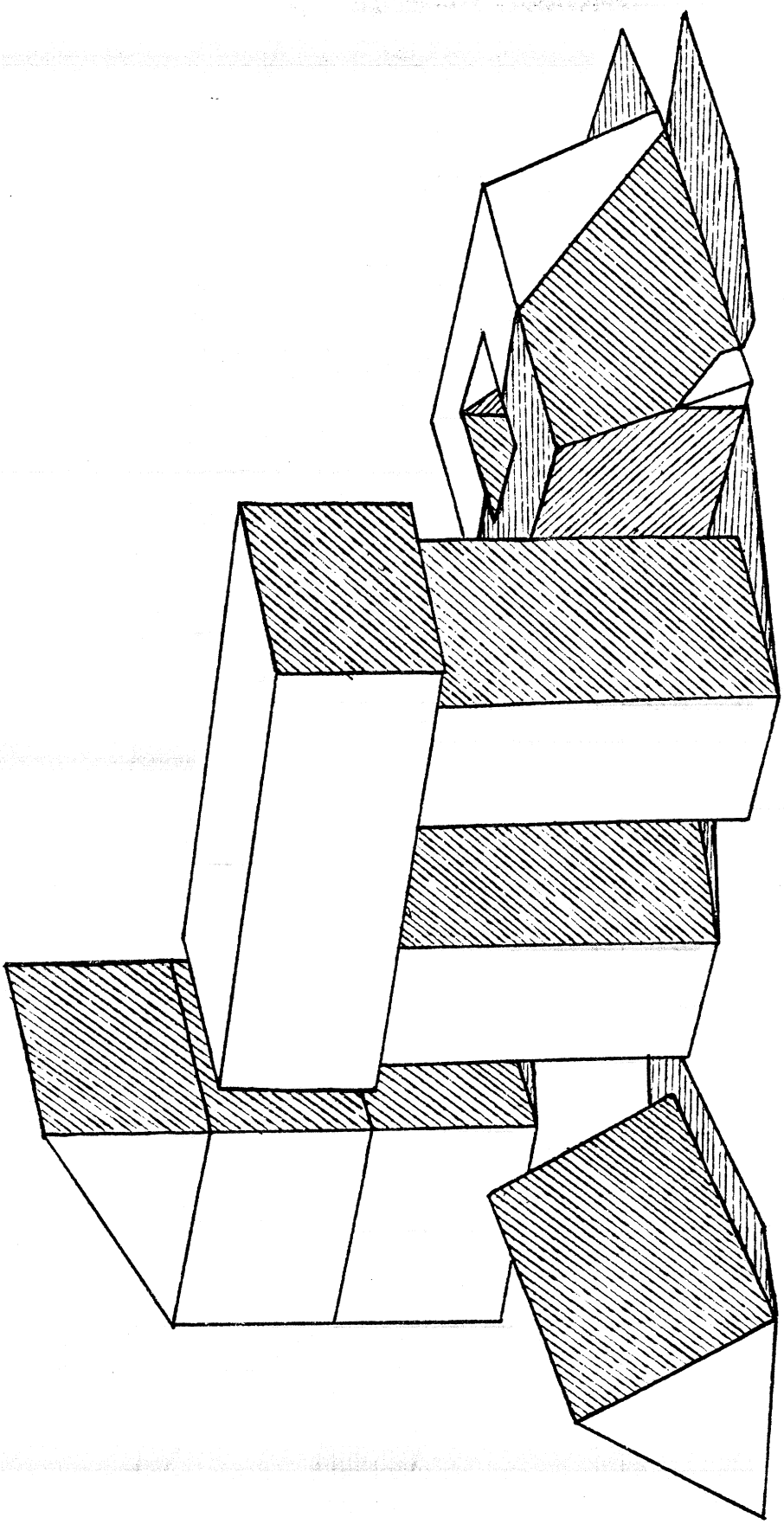


FIGURE 1

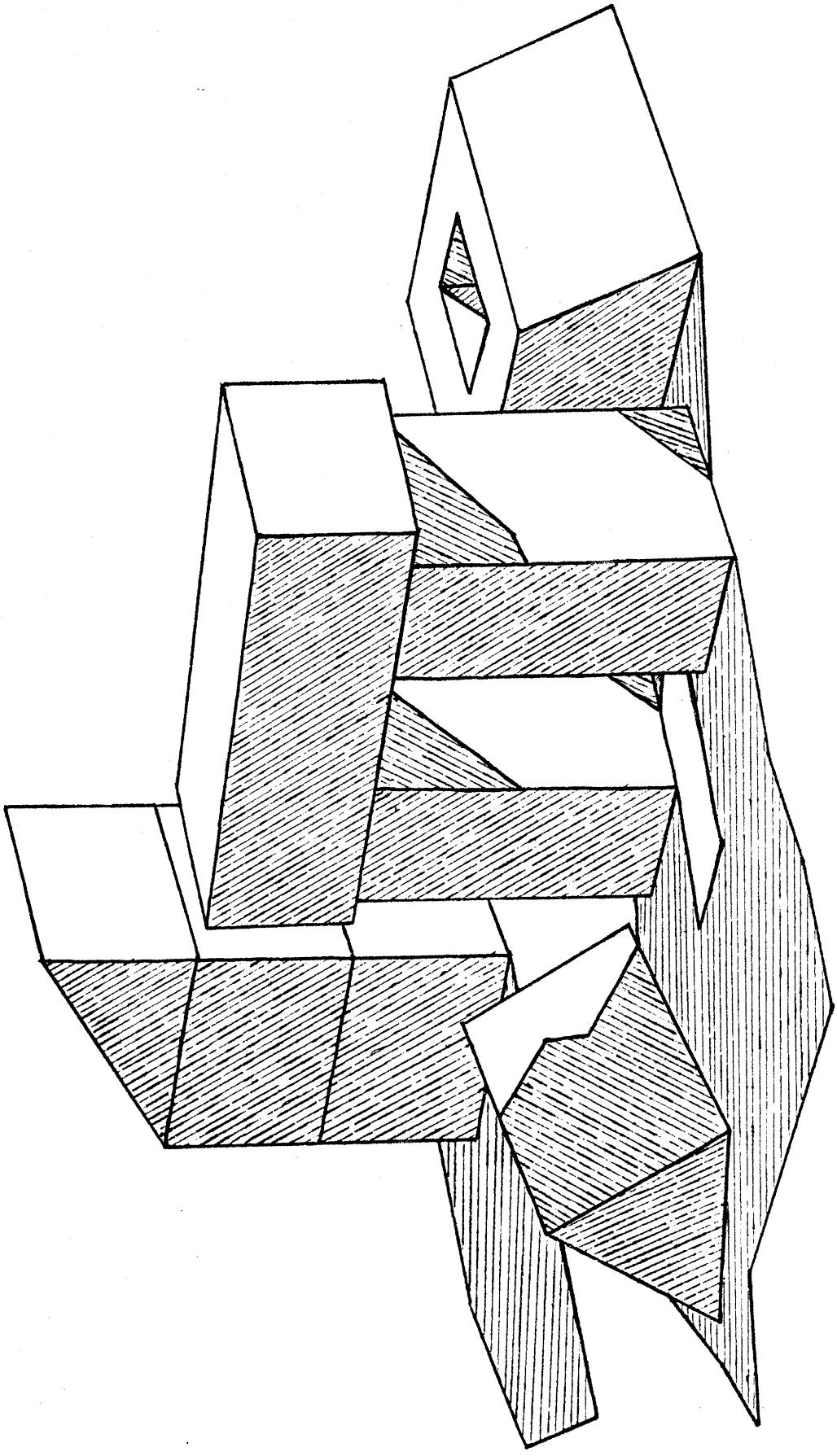


FIGURE 2

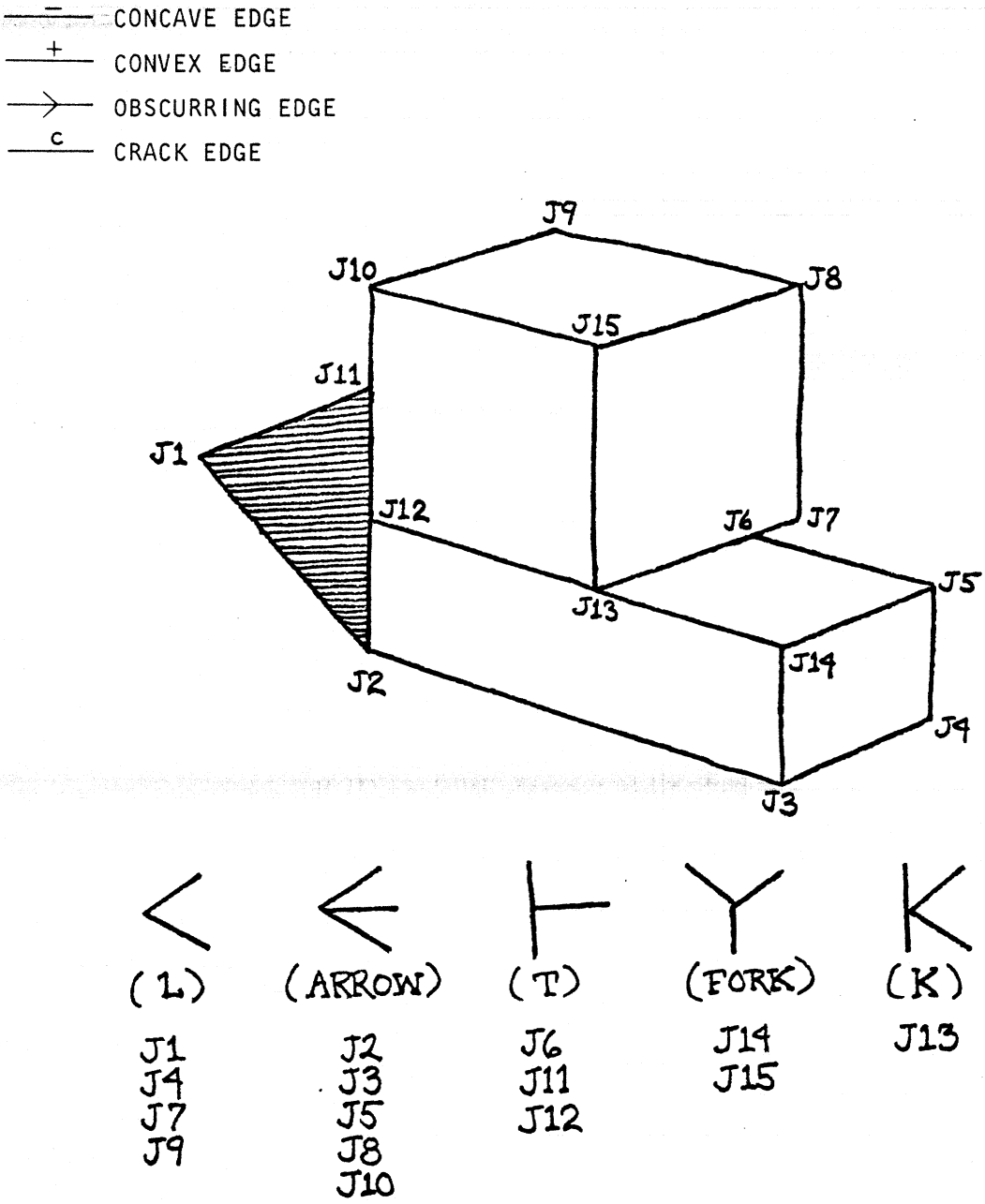

CONCAVE EDGE
CONVEX EDGE
OBSCURRING EDGE
CRACK EDGE
J9
J10
J8
J15
J11
J1
J12
J6
J7
J5
J13
J14
J2
J4
J3
(L)
J1
J4
J7
J9
(ARROW)
J2
J3
J5
J8
J10
(T)
J6
J11
J12
(FORK)
J14
J15
(K)
J13

FIGURE 3

eliminate impossible combinations of labels in a line drawing and, as such, contains implicit knowledge about topology.

(4) Optional heuristics which can be invoked to select a single labeling from among those which remain after all the other knowledge in the program has been used. These heuristics find a "plausible" interpretation if required. For example, one heuristic eliminates interpretations that involve concave objects in favor of ones that involve convex objects, and another prefers interpretations which have the smallest number of objects; this heuristic prefers a shadow interpretation for an ambiguous region to the interpretation of the region as a piece of an object.

BETTER EDGE DESCRIPTION

A large proportion of Waltz's energy and thought has gone into the choice of the set of possible line labels and the sets of possible junction labels. In this he was have been guided by experiment with his program -- there are simply too many labels to hand simulate the program's reaction to a scene.

The program's evolution has generally involved (1) the subdivision of one or more edge labels into several new labels embodying finer distinctions, and (2) the recomputation of the junction label lists to include these new distinctions.

Suppose, for example, that we further break down each class according to whether or not each edge can be the bounding edge of an object. Objects can be bounded by obscuring edges, concave edges, and crack edges. Figure 4 shows the results of appending a label analogous to the "obscuring edge" mark to crack and concave edges. This approach is similar to one first proposed by Freuder.

Given a set of line labels, there is of course the problem of finding all possible trihedral junctions. Huffman observed that three intersecting planes, whether mutually orthogonal or not, divide space into eight parts so that the types of trihedral vertex can be characterized by the octants of space around the vertex which are filled by solid material.

Consider the general intersection of three planes shown in figure 5. These planes divide space into octants. We can then generate all possible geometries and non-degenerate views by imagining various octants to be filled in with solid material. There are junctions which correspond to having 1, 2, 3, 4, 5, 6, or 7 octants filled. When considered from all possible viewing positions, the allowed geometries produce 196 different junction labelings. There are some other geometries which Waltz does not use to generate junction labels because each is an extremely unlikely arrangement when

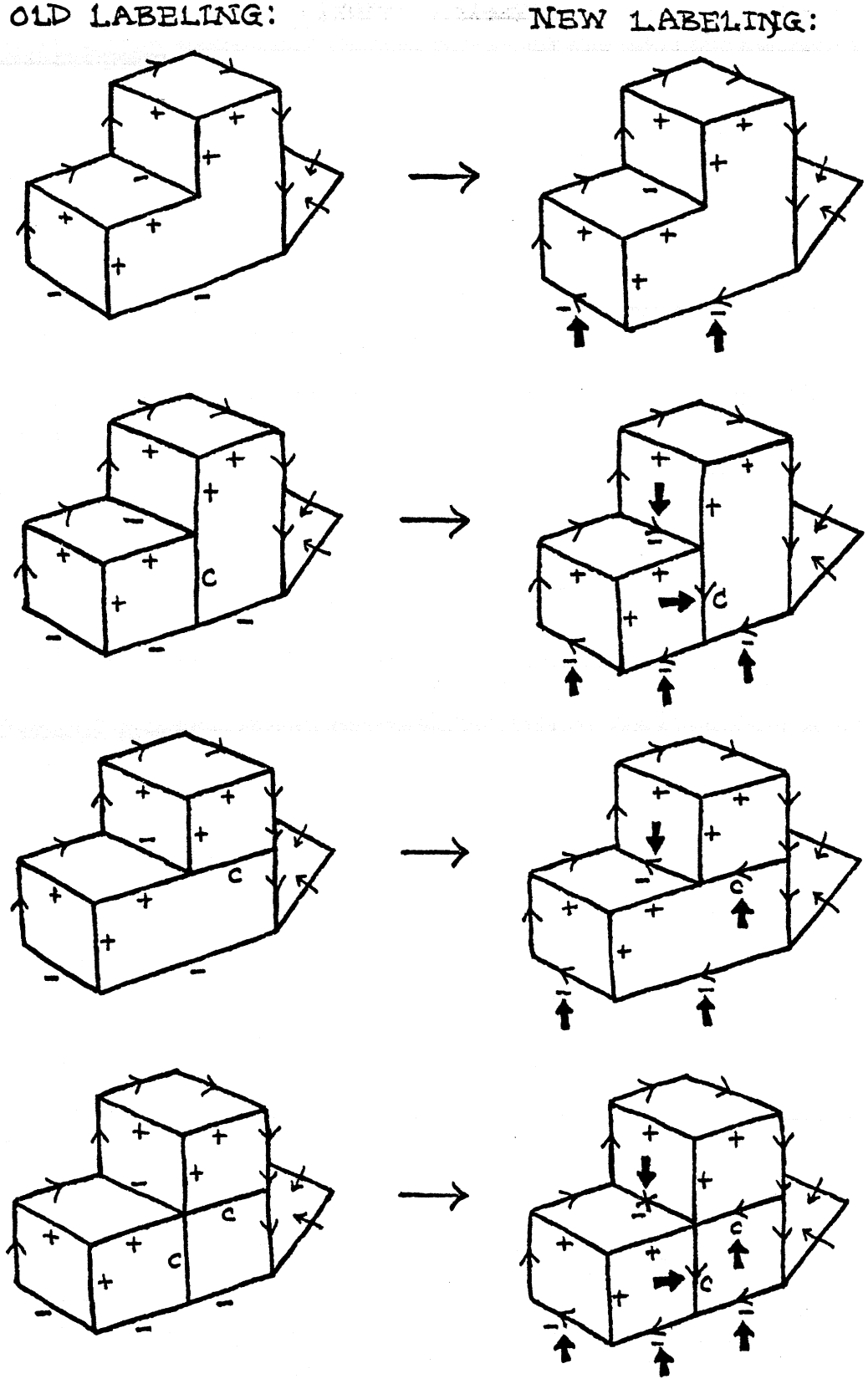


FIGURE 4

INTERPRETATION:

R1 / R2 — AN INSEPARABLE CONCAVE EDGE; THE OBJECT OF WHICH R1 IS A PART [OB(R1)] IS THE SAME AS [OB(R2)].

R1 / R2 —< A SEPARABLE TWO-OBJECT CONCAVE EDGE; IF [OB(R1)] IS ABOVE [OB(R2)] THEN [OB(R2)] SUPPORTS [OB(R1)].

R1 / R2 —> SAME AS ABOVE; IF R1 IS ABOVE R2, THEN [OB(R2)] OBSCURES [OB(R1)] OR [OB(R1)] SUPPORTS [OB(R2)].

R1 / R2 —>< A SEPARABLE THREE-OBJECT CONCAVE EDGE; NEITHER [OB(R1)] NOR [OB(R2)] CAN SUPPORT THE OTHER.

R1 / R2 —C> A CRACK EDGE; [OB(R2)] IS IN FRONT OF [OB(R1)] IF R1 IS ABOVE R2.

R1 / R2 —C< A CRACK EDGE; [OB(R2)] SUPPORTS [OB(R1)] IF R1 IS ABOVE R2.

SEPARATIONS:

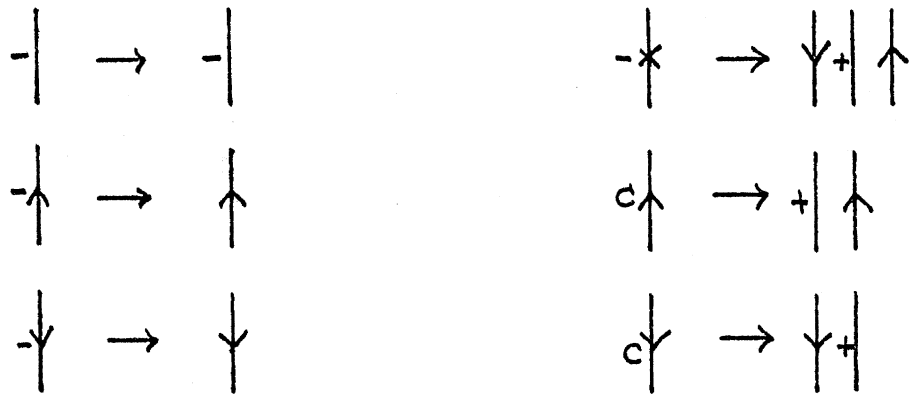


FIGURE 4

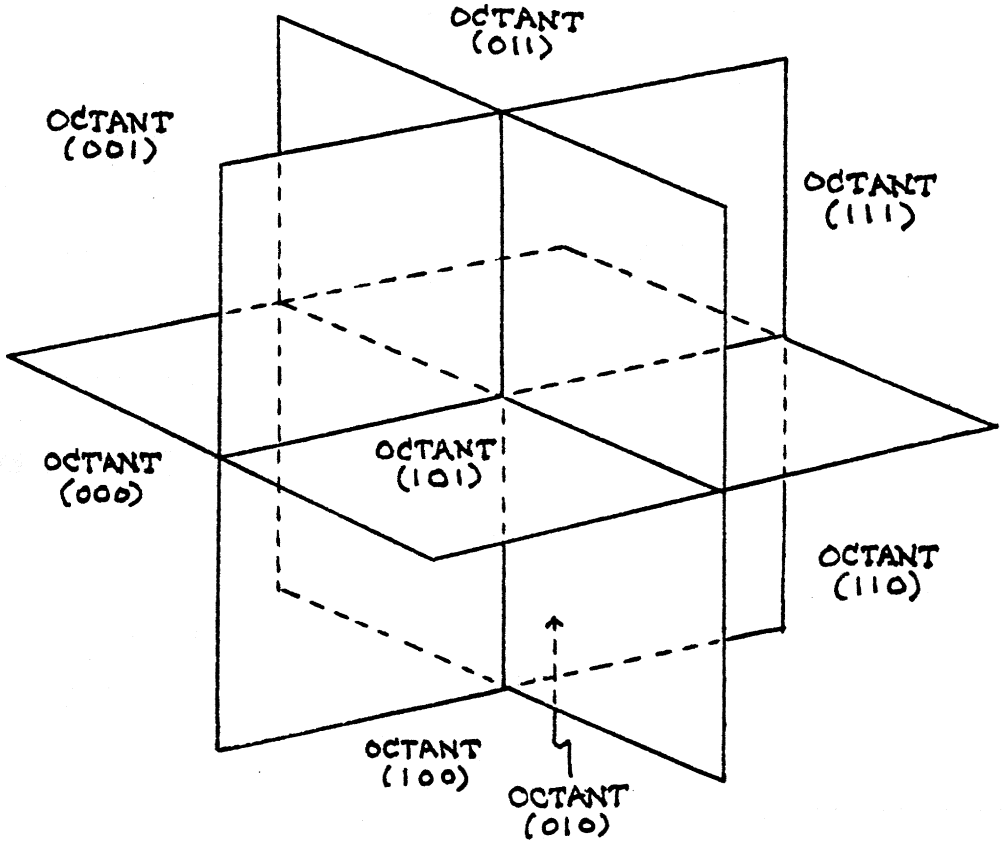

OCTANT
(011)
OCTANT
(001)
OCTANT
(111)
OCTANT
(000)
OCTANT
(101)
OCTANT
(110)
OCTANT
(100)
OCTANT
(010)

FIGURE 5

compared to the allowed geometries.

SHADOWS AT TRIHEDRAL VERTICES

To find all the variations of these vertices which include shadow edges, first note that vertices with 1, 2, 6 or 7 octants filled cannot cause shadows such that the shadow edges appear as part of the vertex. This can be stated more generally: in order to be a shadow-causing vertex (i.e. a vertex where the caused shadow edge radiates from the vertex) there must exist some viewing position for the vertex from which either two concave edges and one convex edge or one concave edge and two convex edges are visible.

After considering shadows, Waltz works from a list of 505 junctions. The interesting thing is that the number of junction labels, while fairly large, is very small compared to the number of possibilities if the branches of these junctions were labeled independently. The set, however, is already sufficient for most scenes which a person would construct out of plane-faced objects, provided that he did not set out to deliberately confuse the program.

ILLUMINATION

The final expansion of the junction list comes by way of introducing illumination information into the line labels. We will not go into this here. It is sufficient to note that the list of junctions is consequently enlarged to hold a few thousand entries.

COMBINATION RULES

Combination rules are used by a filtering program to select the label, or labels, which correctly describe the scene features that could have produced each junction in the given line drawing. The simplest type of combination rule merely states that a label is a possible description for a junction if and only if there is at least one label which "matches" it assigned to each adjacent junction. Two junction labels "match" if and only if the line segment which joins the junctions gets the same interpretation from both of the junctions at its ends.

EXPERIMENTAL RESULTS

Waltz's program computes the full list of dictionary entries for each junction in the scene, eliminates from the list those labels which can be precluded on the basis of local features, assigns each reduced list to its junction, and then the filtering program computes the possible labels for each line, using the fact that a line label is possible if and only if there is at least one junction label at each end of the line which contains the line label. Thus, the list of possible labels for a line segment is the intersection of the two lists of possibilities computed from the junction labels at the ends of the line segment. If any junction label would assign a interpretation to the line segment which is not in this intersection list, then that label can be eliminated from consideration. The filtering program uses a network iteration scheme to systematically remove all the interpretations which are precluded by the elimination of labels at a particular junction.

Figure 6 supplements figure 1 and figure 2 in showing some of the scenes which the program is able to handle. The segments which remain ambiguous after its operation are marked with stars, and the approximate amount of time the program requires to label each scene is marked below it. The program is written partially in MICRO-PLANNER and partially in compiled LISP.

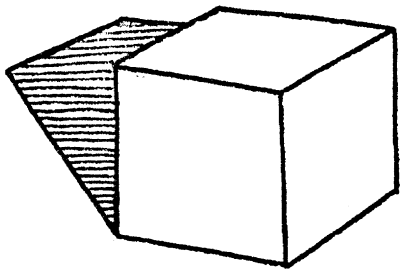
THE WALTZ EFFECT

At this point let us look at a summary of the strongest features of Waltz's theory, which when taken together, constitute what we have come to call "The Waltz Effect:"

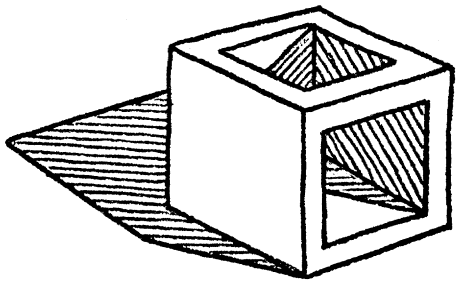
THE FINITE SIZE OF THE TASK: Waltz ends with a need to enumerate a few thousand junction possibilities. A project of such scope is large enough to frighten away anyone who spends only a few hours looking at the problem with a fear that the job is too big for all practical purposes. Yet, as often seems to be the case, the apparent infinity turns out to be quite manageable.

LACK OF AMBIGUITY IN ANALYSIS: Generally the Waltz procedure converges on one interpretation for each line in a scene. Only a few lines will have more than one interpretation, and then usually only two or three. This seems in part a result of the constraint contributed by observed shadows. Curiously, early work was made more difficult by the supposition that shadows were only an annoying complication!

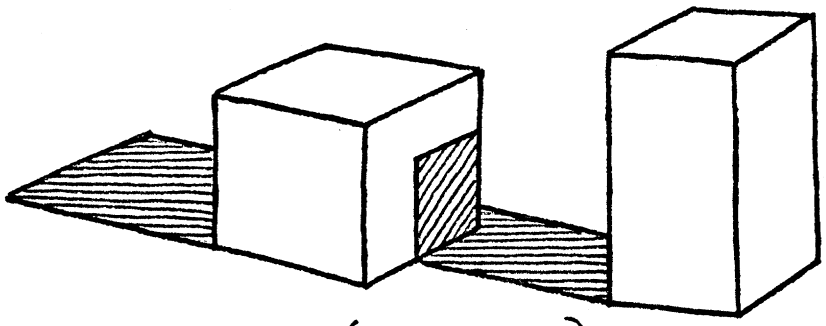
RAPID CONVERGENCE: Potentially, the known nature of a particular line could propagate its influence over a considerable distance and be required at a



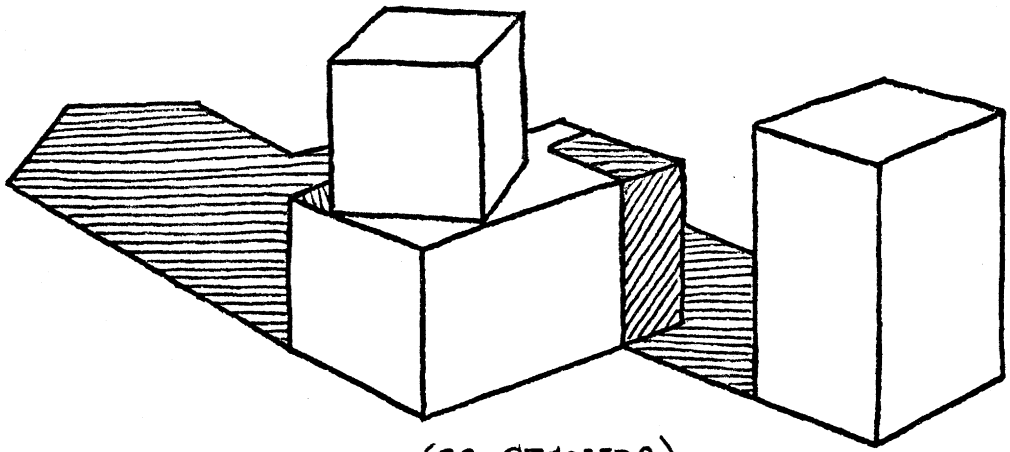
(5 SECONDS)



(15 SECONDS)

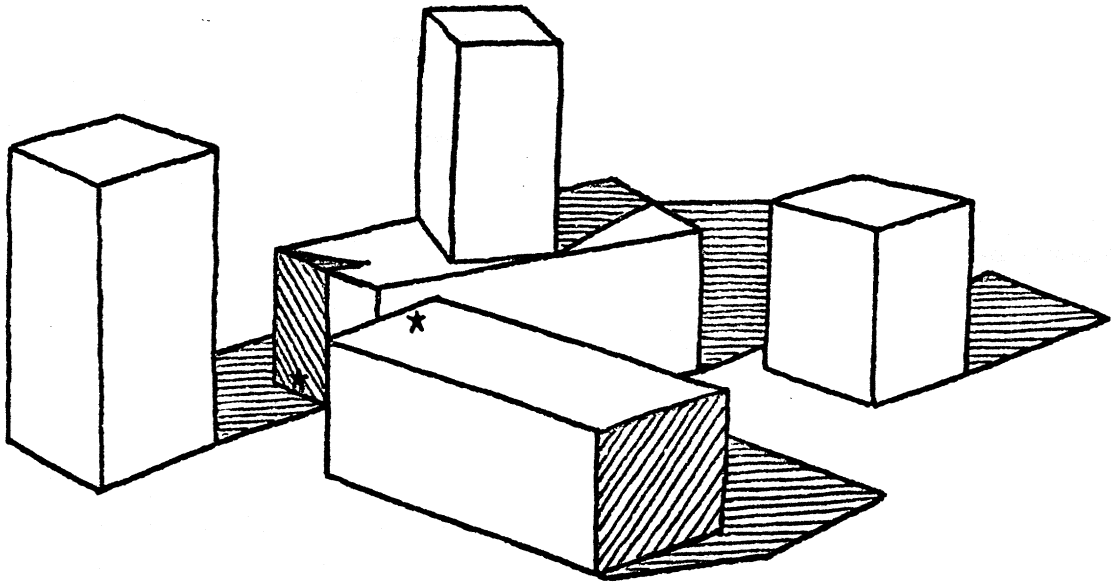


(15 SECONDS)

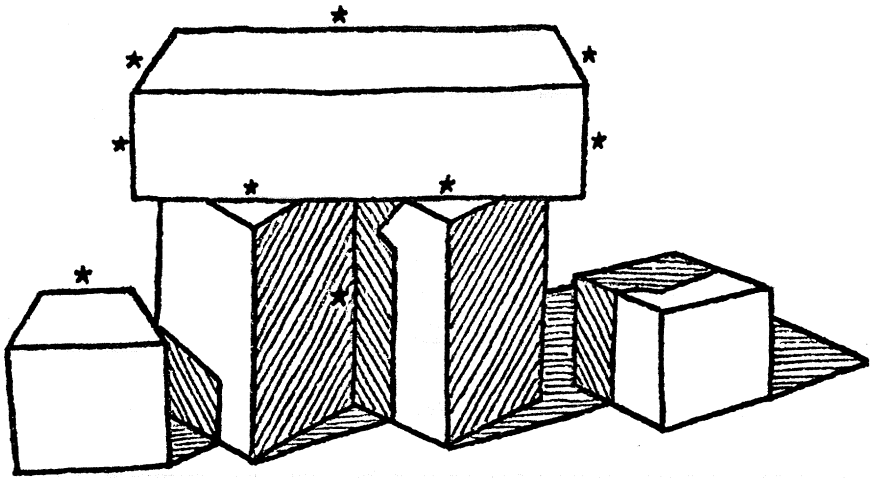


(22 SECONDS)

FIGURE 6



(39 SECONDS)



(48 SECONDS)

FIGURE 6 (cont.)

considerable distance to effect some final interpretation decision. This does not seem to be the case for two reasons: First, influence does not readily pass through the T-joints proliferate on object boundaries. Second, the physical constraints seem powerful enough to fully disambiguate the identity of a line after only a few vertexes near it are explored. Waltz's work suggests that scene analysis on this level is to a large extent a local phenomenon, and incidentally, it should be amenable to parallel processing.

DESCRIPTION REFINEMENT SIMPLIFIES ANALYSIS: Surely proper refinement of one's descriptive base should always lead to better, faster analysis. Waltz's result is indeed a crisp demonstration of that end of a spectrum: instead of combinatorial explosion and deductive chaos, we see good description forcing analysis. But arbitrary refinement will not work. For example, subdividing the lines according to their length would not add much useful constraint.

EXPLANATION OF OLDER WORK: Waltz's work gives theoretical substance to a good bit of prior work conducted in an exploratory, often ad hoc mode. Older empirical observations based on experiments with functioning programs are now explained in terms of understood particulars of Waltz's catalogue of physical constraints. Indeed Waltz's theory is so well worked out that the knowledge needed in analyzing drawings is reduced in his system to a sort of compiled form, with the major procedural activities being table look up and straightforward comparisons. Now less seems required of sophisticated problem solving to handle the problem.

SOME RELEVANT READING

Clowes, M. "On Seeing Things." Artificial Intelligence, 2, No. 1 (1971).

Huffman, D. "Impossible Objects as Nonsense Sentences." Machine Intelligence 6. Ed. D. Michie and B. Meltzer. Edinburgh: Edinburgh University Press, 1971.

HORN'S MATHEMATICAL LIGHTNESS THEORY

Next we turn to Horn's work on lightness and see that it is to image analysis what Waltz's is to scene analysis, namely an exploitation of natural constraints with a consequent reduction in the problem solving required to deal with a problem, in this case, the naming of colors under drift in illumination intensity and spectrum.

Of course there has always been great interest in how we perceive colors and numerous explanations have been forwarded (Newton 1704) (Goethe 1810) (Young 1820) (Maxwell 1856) (Helmholtz 1867) (Hering 1875). A selection of some of the early works on this subject can be found in (MacAdam 1970).

In any event the human perceptual apparatus is remarkably succesful in coping with large variations in illumination. The colors we perceive are closely correlated with the surface colors of the objects viewed, despite large temporal and spatial differences in color and intensity of the incident light. This is surprising since we cannot sense reflectance directly. After all, the light intensity at a point in the image is the product of the reflectance at the corresponding object point and the intensity of illumination at that point. It would seem that disentangling this product must be prerequisite to the perception of color.

LIGHTNESS JUDGING

Let us define lightness as the perceptual quantity closely correlated with surface reflectance. Only after determining lightness can three color images be compared to reliably determine colors locally.

In pursuing this, the following notation will be used:

s' Intensity of incident illumination at a point on the object.
r' Reflectance at a point on the object.
p' Intensity at an image point. Product of s' and r'.

s, r, p Logarithms of s', r' and p' respectively.

d Result of applying forward or differencing operator to p.
t Result of applying threshold operator to d.
l Result of applying inverse or summing operator to t.

D Simple derivative operator in one dimension.
T Continous threshold operator, discards finite part.
I Simple integration operator in one dimension.
L Laplacian operator -- sum of second partial derivatives.
G Inverse of the Laplacian, convolution with $(1/2\pi)\log_e(1/r)$.

D*, T*, I*, L* and G* Discrete analogues of D, T, I, L and G.

The output I, will not be called lightness since there is probably not yet a generally accepted definition of this term. It is, however, intended to be monotonically related to lightness. Note that I is related to the logarithm of reflectance, while the perceptual quantity is perhaps more closely related to the square-root of reflectance.

LIGHTNESS IN TWO DIMENSIONAL IMAGES -- THE CONTINUOUS CASE

Horn's original paper introduces the concepts involved in lightness calculation by way of a one-dimensional simplification. Here we choose to plunge directly into the two-dimensional case needed to deal with actual images. The price will be a loss of some intuitive understanding for readers without a good knowledge of the mathematics.

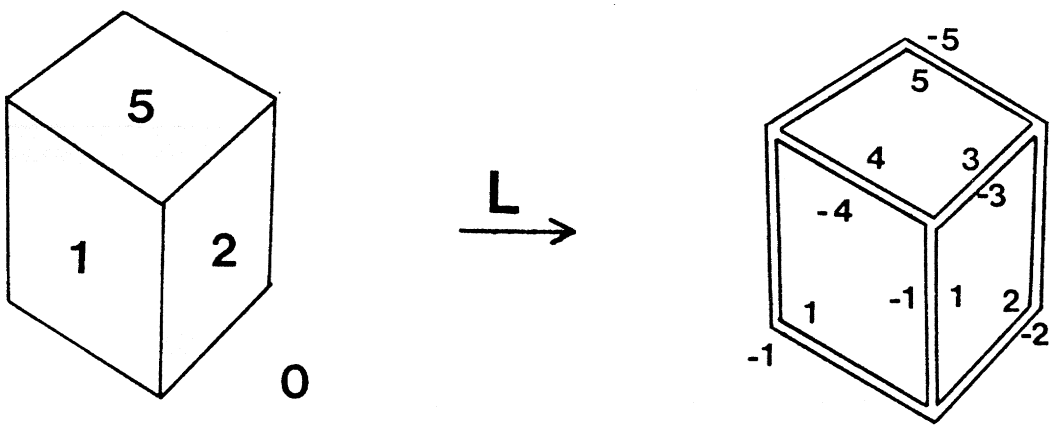
One needs to find two-dimensional analogues to differentiation and integration. The first partial derivatives are directional and thus unsuitable since they will, for example, completely eliminate evidence of edges running in a direction parallel to their direction of differentiation. Exploring the partial derivatives and their linear combinations one finds that the Laplacian operator is the lowest order combination that is isotropic, or rotationally symmetric. The Laplacian operator is, of course, the sum of the second partial derivatives.

APPLYING THE LAPLACIAN TO A MONDRIAN PAINTING

Before investigating the invertibility of this operator, let us see what happens when one applies it to the image of a Mondrian painting (scenes divided into regions of absolutely uniform matte color). Inside any region one will obtain a finite value due to the variation in illumination intensity. At each edge one will get a pulse pair, one positive and one negative. The area of each pulse will be equal to the intensity step.

This can best be seen by considering the first derivative of a step, namely a single pulse. If this is differentiated again one obtains a doubled pulse as described. Since this

pulse will extend along the edge, one may think of it as a pulse-wall. So each edge separating regions will produce a doubled pulse-wall. It is clear that one can once again separate the component due to reflectance and illumination simply by discarding all finite parts (see figure 1).



APPLYING THE LAPLACIAN OPERATOR TO THE IMAGE OF A MODRIAN FIGURE

FIGURE 1

INVERSE OF THE LAPLACIAN OPERATOR

To complete the task at hand one then has to find a process for undoing the effect of applying the Laplacian. Again there are a number of approaches to this problem; we will use the shortest. In essence one has to solve for p(x,y) in a partial differential equation of the form:

$$L(p(x,y)) = d(x,y)$$

This is Poisson's equation and it is usually solved inside a bounded region using Green's function (Garabedian 1964).

$$p(x,y) = \iint G(\xi,\eta;x,y)\, d(\xi,\eta)\, d\xi\, d\eta$$

The form of Green's function G, depends on the shape of the region boundary. If the

retina is infinite all points are treated similarly and Green's function depends only on two parameters, $(\xi - x)$ and $(\eta - y)$. This positional independence implies that the above integral simply becomes a convolution. It can be shown that Green's function for this case is:

$$G(\xi,\eta;x,y) = (1/2\pi) \log_e(1/r)$$

Where $r^2 = (\xi - x)^2 + (\eta - y)^2$

So $$p(x,y) = \iint (1/2\pi) \log_e(1/r)\, d(\xi,\eta)\, d\xi\, d\eta$$

Thus the inverse of the Laplacian operators is simply convolution with $(1/2\pi) \log_e(1/r)$. To be precise one has:

$$\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) \iint (1/2\pi) \log_e(1/r)\, d(\xi,\eta)\, d\xi\, d\eta = d(x,y)$$

This is a two-dimensional analogue of:

$$\frac{d}{dx} \int_{-\infty}^{x} f(t)\, dt = f(x)$$

WHY ONE CAN USE THE CONVOLUTIONAL INVERSE

If the retina is considered infinite one can express the inverse as a simple convolution. If the retina is finite on the other hand one has to use the more complicated Green's function formulation.

Now consider a scene on a uniform background and assume that the image of the scene is totally contained within the retina. The result of applying the forward transform and thresholding will be zero in the area of the uniform background. The convolutional inverse will therefore receive no contribution from outside the retina. As a result one can use the convolutional form of the inverse provided the image of the scene is totally contained within the retina.

NORMALIZATION

One finds that the reconstructed reflectance is not unique. That is, any non-singular solution of $L(p(x,y)) = 0$ can be added to the input without affecting the result. On the infinite plane such solutions have the form $p(x,y) = (ax + by + c)$. If the scene only occupies a finite region of space it can be further shown that the solution will be unique up to a constant and that one does not have to take account of possible slopes. To be specific: the background around the scene will be constant in the reconstruction. Calling the region with highest numerical value white appears to be a reasonable method.

TWO-DIMENSIONAL METHOD -- DISCRETE CASE

We turn from a continous image to one sampled at discrete points. First we will have to decide on a tesselation of the image plane.

For regular tesselations the choice is between triangular, square and hexagonal unit cells. In much past work on image processing, square tesselations have been used for the obvious reasons. This particular tesselation of the image has a number of disadvantages. Each cell has two kinds of neighbors, four adjoining the sides, four on the corners. This results in a number of asymmetries. It makes it difficult for example to find convenient difference schemes approximating the Laplacian operator with low error term. Triangular unit cells are even worse in that they have three kinds of neighbors.

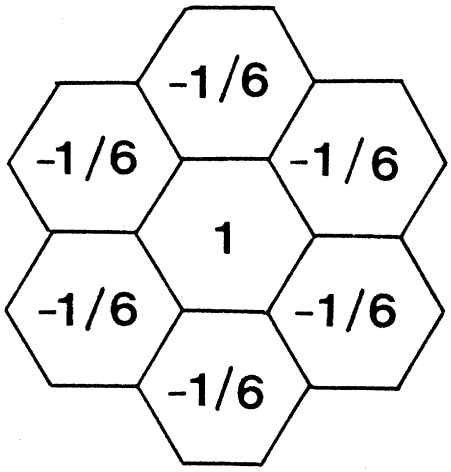
For these reasons we will use a hexagonal unit cell. It should be kept in mind however that it is easy to develop equivalent results using different tesselations.

DISCRETE ANALOGUE OF THE LAPLACIAN

Having decided on the tesselation we need now to find a discrete analogue of the Laplacian operator. Convolution with a central positive value and a rotationally symmetric negative surround of equal weight is one possibility. Aside from a negative scale factor, this will approach application of the Laplacian in the limit as the cell size tends to zero.

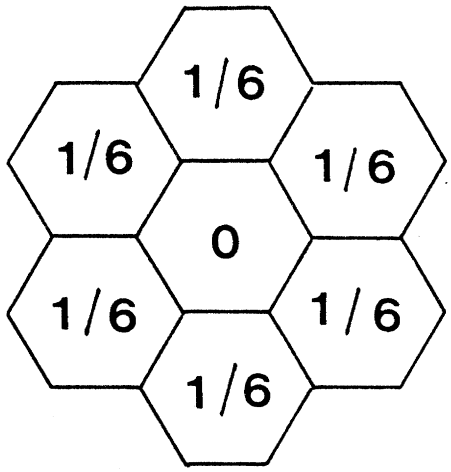
If one were to use complicated surrounds, the trade-offs between accuracy and resolution would suggest using a negative surround that decreases rapidly outward. For the sake of simplicity we will choose convolution with a central cell of weight one, surrounded by six cells of weight -1/6. This function is convenient, symmetric and has a small error term. It is equal to $-(h^2/4)\,L - (h^4/64)\,L^2$ plus sixth and higher order

derivatives (Richtmeyer and Morton 1957). It should again be pointed out that similar results can be developed for different functions (see figure 2).



A DISCRETE ANALOGUE OF
THE LAPLACIAN OPERATOR

FIGURE 2a



DELTA FUNCTION MINUS
THIS DISCRETE ANALOGUE

FIGURE 2b

INVERSE OF THE DISCRETE OPERATOR

The forward differencing operator has the form:

$$d_{ij} = p_{ij} - \Sigma\, w_{k-i,l-j}\, p_{kl}$$

Where p_{ij} is the logarithm of image intensity, w_{ij} are weights, which in our case are 1/6, and the sum is taken over the six immediate neighbors.

We now have to determine the inverse operation that recovers p_{ij} from d_{ij}. One approach is to try and solve the difference equation of the form:

$$p_{ij} - \Sigma\, w_{k-i,l-j}\, p_{kl} = d_{ij}$$

Or in matrix form: $W \underline{p} = \underline{d}$. Note that W is sparse, having 1's on the diagonal and -1/6's scattered around. For a finite retina with n sensor cells one has to introduce boundary conditions to ensure that one has as many equations as there are unknowns. One then simply inverts the matrix W and gets: $\underline{p} = W^{-1} \underline{d}$.

This is analogous to the solution in the continous case for a finite retina. W^{-1} corresponds to the Green's function. Much as Green's function has a large "support" i.e., is non-zero over a large area, so W^{-1} is not sparse. This implies that a lot of computation is needed to perform the inverse operation.

SIMPLICITY OF THE INVERSE

The forward transform, involving only a simple subtraction of immediate neighbors, is clearly a rapid, local operation. The inverse on the other hand is global, since each point in the output depends on each point in the differenced image. Computationally this makes the inverse slow. The inverse is simple in one sense however: the difference equations being solved by the inverse have the same form as the equations used for the forward transform and are thus local. The problem is that the output here feeds back into the system and effects can propagate across the retina. The apparent global nature of the inverse is thus of a rather special kind and, as we will see, gives rise to very simple physical implementations involving only local connections.

PHYSICAL MODELS

There are numerous continous physical models that illustrate the inverse transformation. Anything that satisfies Poisson's equation will do. Such physical models help one visualize what the inverse of a given function might be. Examples in two dimensions are: perfect fluid-flow, steady diffusion, steady heat-flow, deformation of an elastic membrane, electro-statics and current flow in a resistive sheet. In the last model for example, the input is the distribution of current flowing into the resistive sheet normal to its surface, the output is the distribution of electrical potential over the surface.

In addition to helping one visualize solutions, continous physical models also suggest computationally helpful discrete models. These can be arrived at simply by cutting up the two-dimensional space in a pattern corresponding to the interconnection of neighboring cells. That is, the remaining parts form a pattern dual to that of the sensor cell pattern. We will discuss only one such discrete model.

A DISCRETE PHYSICAL MODEL

Consider the resistive sheet described, cut up in the dual pattern of the hexagonal unit cell pattern. What will be left is an interconnection of resistors in a triangular pattern. The inputs to this system will be currents injected at the nodes, the potential at the nodes being the output. This then provides a very simple analog implementation of the tedious inverse computation (see figure 3).

It is perhaps at first surprising to see that each cell is not connected to every other in a direct fashion. One would expect this from the form of the computational inverse. Each cell in the output does of course have a connection via the other cells to each of the inputs. Paths are shared, however, in a way that makes the result both simple and planar.

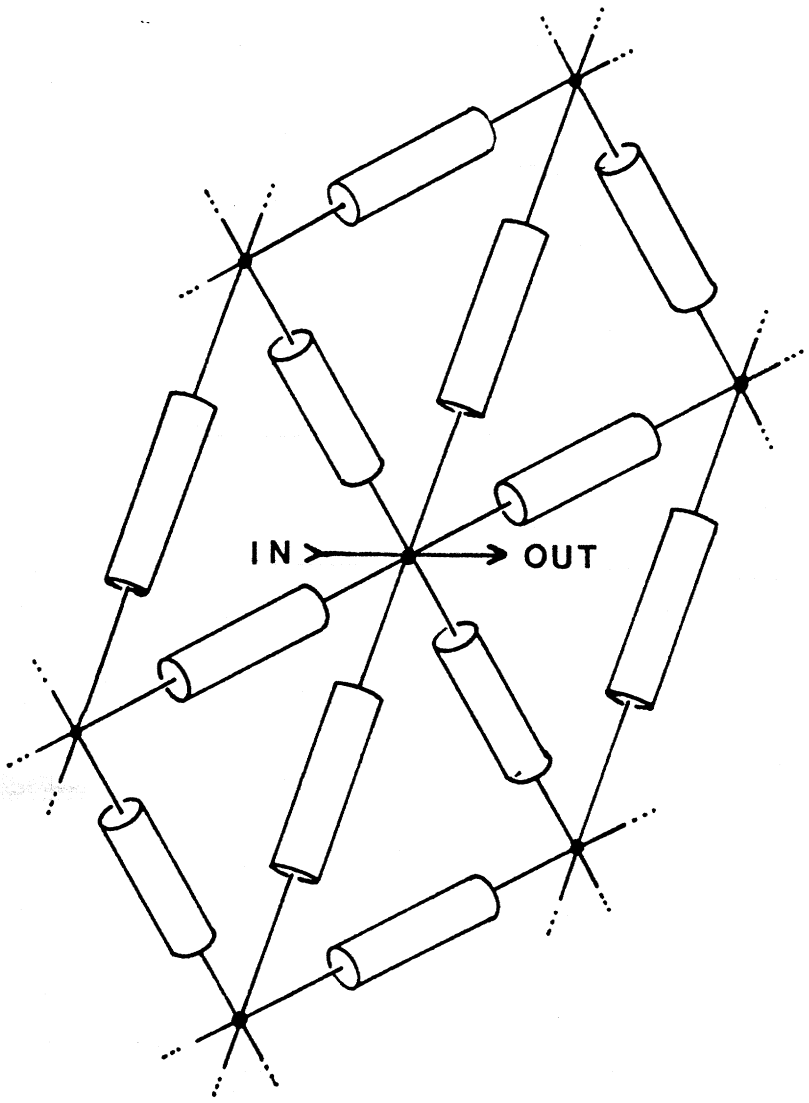
Consider for the moment just one node. The potential at the node is the average of the potential of the six nodes connected to it plus the current injected times R/6, where R is the resistance of each resistor. The economy of connection is due to the fact that the outputs of this system are fed back into it. It also illustrates that this model locally solves exactly the same difference equation as that used in the forward transform, only now in reverse.

This immediately suggests an important property of this model: by simply changing the interconnections one can make an inverse for other forward transforms. Simplest of all are other image plane tesselations, both regular and irregular. One simply connects the resistors in the same pattern as are the cells in the input.

More complicated weighted surrounds can be handled by using resistors with resistances inversely proportional to the weights. The network of resistors will then no longer be planar.

A FEED-BACK SCHEME FOR THE INVERSE

Both the comment about outputs feeding back into the resistive model and the earlier notes about iterative schemes suggest yet another interesting model for the inverse using linear summing devices. Operational amplifiers can serve this purpose. One simply connects the summing element so that they solve the difference equation implied by the forward transform. Once again it is clear that such a scheme can be generalized to arbitrary tesselations and weighted negative surrounds simply by changing the interconnections and attenuations on each input. Some questions of stability arise with esoteric interconnections. For the simple ones stability is assured.



RESISTIVE MODEL OF THE INVERSE COMPUTATION.
THE INPUTS ARE THE CURRENTS INJECTED AT THE NODES.
THE OUTPUTS ARE THE POTENTIALS AT THE NODES.

FIGURE 3

A little thought will show that the resistive model described earlier is in fact a more economical implementation of just this scheme with the difference that there the inputs are currents, while here they are potentials.

SOME NOTES ON THE METHOD

Notice that an illumination gradient that varies as some power of distance across the image becomes a linear slope after taking logarithms and thus produces no component after the differencing operations. Such simple gradients are suppressed even without the thresholding operation.

In practice the parameters used in choosing the threshold may not be known or may be variable. In this case one can look at a histogram of the differenced image. It will contain values both positive and negative corresponding to edges and also a large number of values clustered around zero due to illumination gradients, noise and so on. The threshold can be conveniently chosen to discard this central blob.

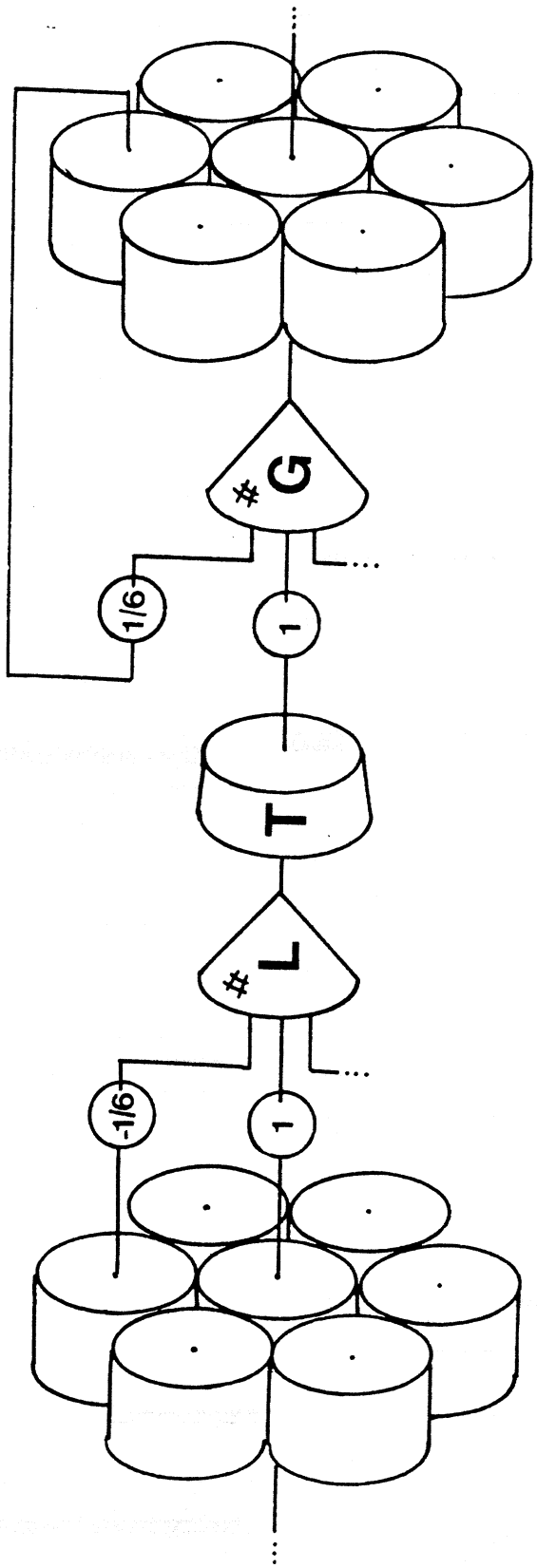
Noise and illumination gradients have an effect similar to that in the one-dimensional case. With finite cell spacing one cannot precisely separate the two components of the image intensity and at each edge the information will be corrupted slightly by noise and illumination gradient. As the density of edges per cell area goes up the effect of this becomes more apparent. In highly textured scenes the illumination gradient is hard to eliminate.

Applying the retinex operation to an image considerably reduces the range of values. This is because the output, being related to reflectance, will only have a range of one to two orders of magnitude, while the input will also have illumination gradients. This will make such processing useful for picture recording and transmission.

LIMITATIONS OF THE SIMPLE SCHEME PRESENTED

The method presented here will not correctly calculate reflectance if used unmodified on general scenes. It may however calculate lightness fairly well. As the method stands now for example, a sharp shadow edge will not be distinguished from a real edge in the scene and the two regions so formed will produce different outputs, while their reflectances are the same. It may be that this is reasonable nevertheless, since we perceive a difference in apparent lightness.

Smooth gradations of reflectance on a surface due either to shading or variations in surface reflectance will be eliminated by the thresholding operations except as far as



THE USE OF SUMMING ELEMENTS AND FEED-BACK IN THE IMPLEMENTATION OF BOTH THE FORWARD AND THE INVERSE TRANSFORM.

FIGURE 4

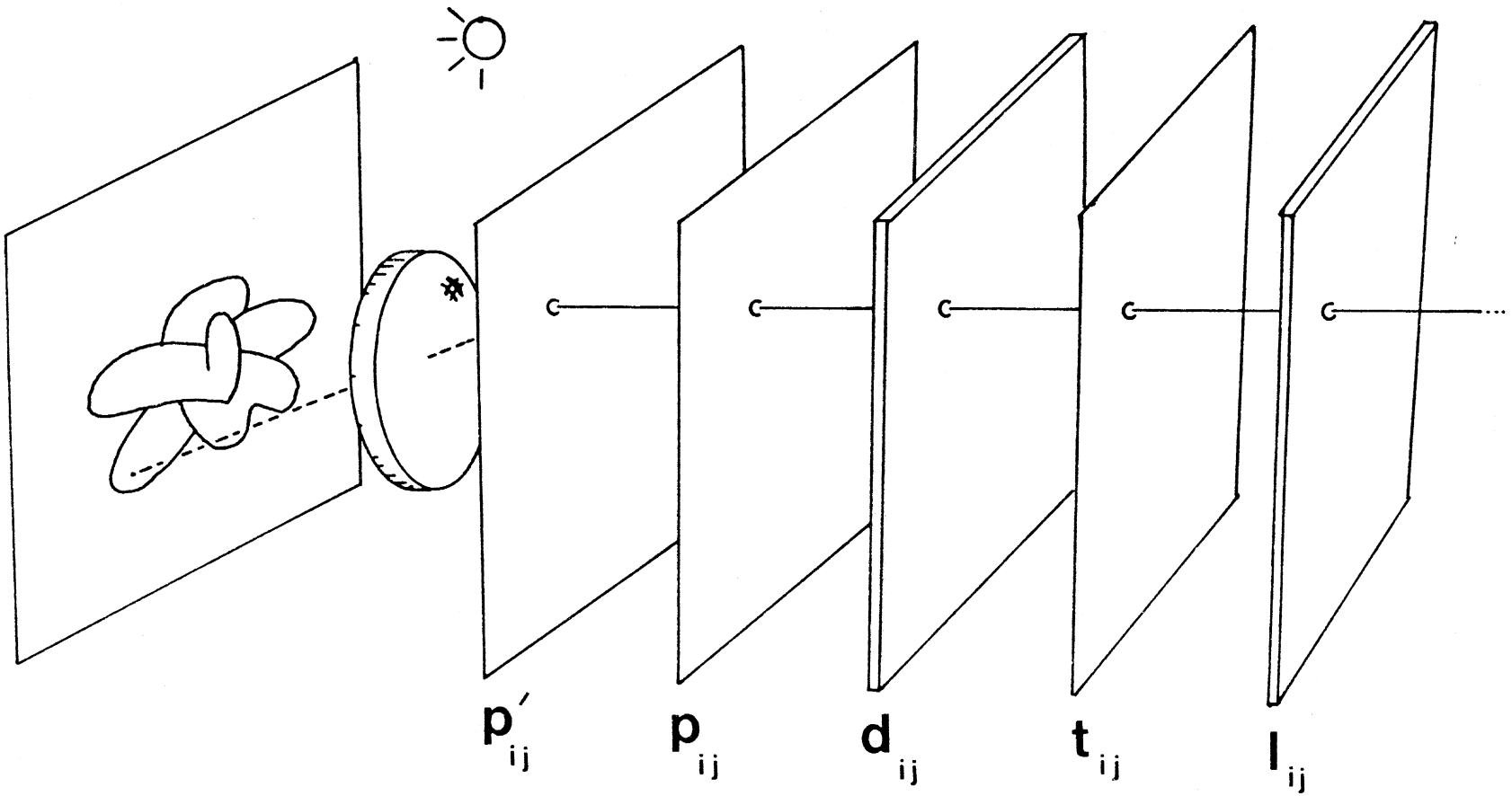


ILLUSTRATION OF THE PARALLEL LAYERS OF OPERATIONS WHICH PRODUCE THE TWO-DIMENSIONAL RETINEX OPERATION. ONLY TWO OF THE OPERATIONS INVOLVE LOCAL INTERACTIONS BETWEEN NEIGHBORING CELLS.

FIGURE 5

they affect the intensity at the borders of the region. This may imply that we need additional channels in our visual system to complement the ones carrying the retinexed information since we do utilize shading as a depth-cue (Horn TR-232).

The simple normalization scheme described will also be sensitive to specular reflections, fluorescent paints and light-sources in the field of view. Large depth-discontinuities present another problem. One cannot assume that the illumination is equal on both sides of the obscuring edge. In this case the illuminating component does not vary smoothly over the retina, having instead some sharp edges.

SOME RELEVANT READING

Brindley, C. S. "Physiology of the Retina and Visual Pathway." Monograph No. 6. Publications of the Physiological Society, Edward Arnold, Ltd., London, 1960.

Cornsweet, T. Visual Perception. New York: Academic Press, 1970.

Garabedian, D. R. Partial Differential Equations. New York: John Wiley, 1964.

Goethe, J. W. von. Zur Farbenlehre. Tuebingen: 1810. Also Theory of Colors. Trans. by C. L. Eastlake. Cambridge: M.I.T. Press, 1970.

Hardy, A. C., Ed. The Handbook of Colorimetry. Cambridge: M.I.T. Press, 1936.

Helmholtz, H. L. F. Handbuch der Physiologischen Optik. Leipzig: Voss, 1867. Also Introduction to Handbook of Physiological Optics, Trans. by J. P. C. S. Southall. Oxford: Oxford University Press, 1937. Reprinted by Dover Publications, New York, 1961.

Helson, H. "Fundamental Problems in Color Vision I." Journal of Experimental Psychology, 23 (1938).

Helson, H. "Fundamental Problems in Color Vision II." Journal of Experimental Psychology, 26 (1940).

Hering, E. "Zur Lehre vom Lichtsinne -- Grundzuege einer Theorie des Farbsinnes." SBK. Akad. Wiss. Wien Math. Naturwiss., 70 (1875).

Judd, D. B. "Hue, Saturation and Lightness of Surface Colors with Chromatic Illumination." Journal of the Optical Society of America, 30 (1940).

Judd, D. B. Color in Business, Science and Industry. New York: John Wiley, 1952.

Land, E. H. "Experiments in Color Vision." Scientific American (1959).

Land, E. H. "The Retinex." American Scientist, 52 (1964).

Land, E. H. and J. J. McCann. "Lightness Theory." Journal of the Optical Society, 61 (1971).

Lettvin, J. Y. "The Color of Colored Things." Quarterly Progress Report 87, Publications of the Research Laboratory for Electronics, M.I.T., 1967.

MacAdam, D. L., Ed. Sources of Color Science. Cambridge: M.I.T. Press, 1970.

Marr, D. "An Analysis of the Primate Retina." A. I. Memo-296, Publications of the Artificial Intelligence Laboratory, M.I.T., Cambridge, Massachusetts, 1974.

Maxwell, J. C. "On the Unequal Sensitivity of the Foramen Centrale to Light of Different Colours." Rep. Brit. Assoc. (1856).

Newton, Sir Isaac. Opticks. London: Samuel Smith & Benjamin Walford, 1704. Also Dover Publications, New York, 1952.

Richards, W. "One-Stage Model for Color Conversion." Journal of the Optical Society, 62 (1971).

Richtmeyer, R. D. and K. W. Morton. Difference Methods for Initial Value Problems. New York: John Wiley, 1957.

Young, T. "On the Theory of Light and Colour." Philosophical Transactions (1820). Also Color Vision. Ed. R. C. Teevan and R. C. Birney. New York: Van Nostrand, 1961.

MARR'S PHYSIOLOGICAL LIGHTNESS THEORY

We now look at Marr's speculations on how Horn's theory might operate in animal retinas. It is not often that investigations of animal mechanisms help in understanding intelligence, since in general the level at which physiology operates and the level at which problem solving is done are so far distant and insulated from one another. But Horn's work hints that quite a lot of vision processing can be understood in terms of simple, uniformly applied operations of the sort that do seem accessable to physiology. And since our goal is to reach all the way around vision, suggestions and confirmations from physiology are not to be ignored.

Therefore we are here concerned with the consequences of assuming that the retina computes lightness, using a parallel method like that of figure 1. The general discrete form of the algorithm is as follows: letters like x, y, denote log of intensity at points X, Y; N(X) refers to the set of points Y in the neighborhood of X; and w is some weighting function on N(X). The difference operation has the form:

$$x' = x - \sum w(Y).y \qquad (3)$$

Letters with a prime, like x', y' refer to differences at X, Y, obtained in this way. The second step is to apply a threshold to the difference signal, by

$$\begin{aligned} x'' &= x' && \text{if } |x'| > \text{some threshold } t \text{ (say)} \\ &= 0 && \text{otherwise.} \end{aligned} \qquad (4)$$

Letters followed by two primes, like x", y", will refer to thresholded difference signals. Finally, the inverse transform

$$x^* = x'' + \sum w(Y).y^* \qquad (5)$$

is applied. Letters like x*, y*refer to the output from the whole process: every point X gives rise to an x*. By inspection, for t=0 (3) and (5) are inverses for point sources (with zero boundary conditions). Hence by linearity, they are inverse transformations.

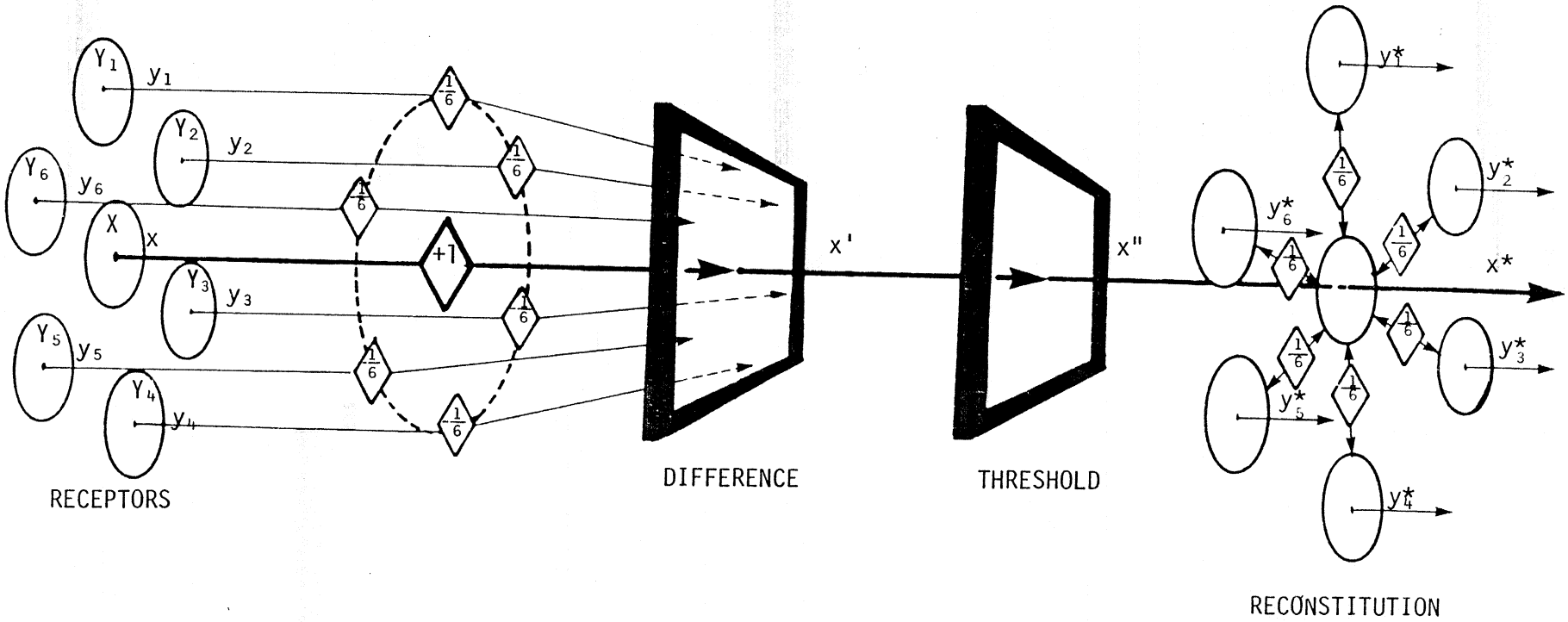

Y1
y1
Y2
y2
Y6
y6
X
x
Y3
y3
Y5
y5
Y4
y4
+1
RECEPTORS
DIFFERENCE
x'
THRESHOLD
x"
y*1
y*2
y*6
x*
y*3
y*5
y*4
RECONSTITUTION

FIGURE 1

THE ANATOMY OF THE RETINA

Most qualitative, and some quantitative aspects of the structure of the primate retina are well understood, thanks to the early work of Cajal (1911), and the recent thorough studies by Missotten (1965), Dowling and Boycott (1966), Boycott and Dowling (with Kolb) (1969), and Kolb (1970). The cell types and connecions of the outer plexiform layer are summarized in figures 2A, 2B, and 2C. The details of the inner plexiform layer are less widely known, and a very brief summary of the cells and synapses described by Boycott & Dowling (1969) is therefore included here. (The word "diffuse," in this context, refers to processes that are distributed perpendicular to the sclera, and the word "stratified" is used to mean layered parallel to the sclera).

AMACRINE CELLS (SEE FIGURE 2A)

(A1) Narrow field diffuse amacrine cells, having a diameter of 10-50mu., average about 25mu., found all over the retina.

(A2) Wide-field diffuse amacrine cells, having processes that spread out gradually as they descend to the level near the ganglion cell bodies, and spread out there to attain a diameter of up to 600mu. These cells are particularly likely to synapse with rod bipolar terminals, and are unlikely to contact the ganglion cell bodies.

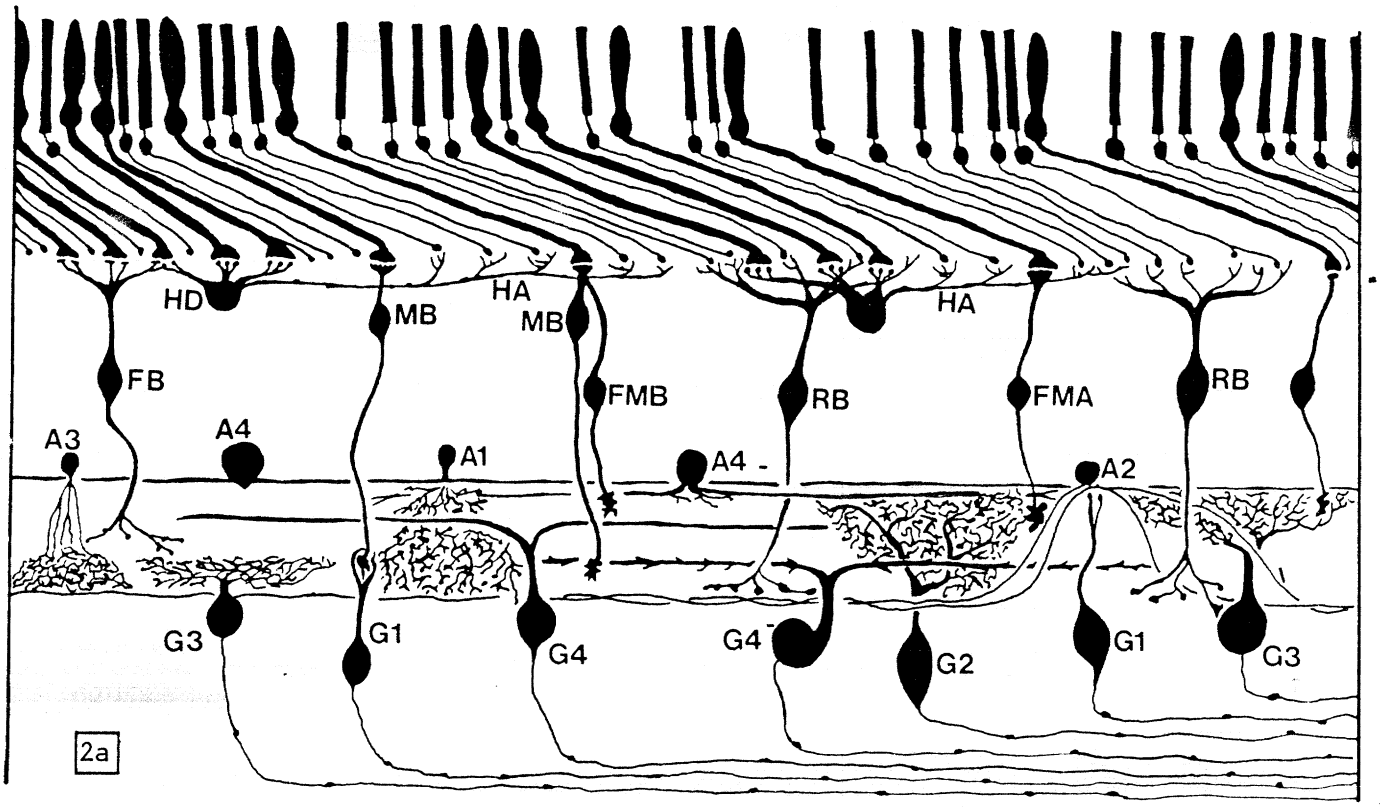
(A3) Stratified diffuse amacrine cells, having a diameter of 20-50mu., are restricted to the top, middle, or to the lower third of the inner plexiform layer, but are diffusely distributed within one of them. A given stratified diffuse amacrine cell probably makes frequent, but not exclusive contact with a particular ganglion cell that has its dendrites similarly distributed.

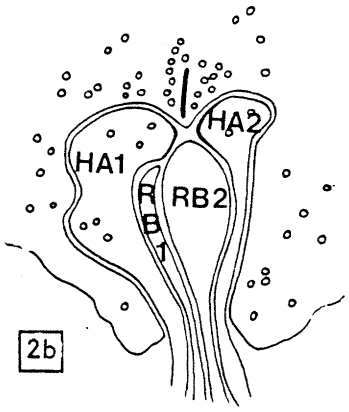
(A4) Unistratified amacrine cells, whose diameter lies between 100 and 1000mu., extend their processes in the plane immediately corneal to the inner plexiform layer.

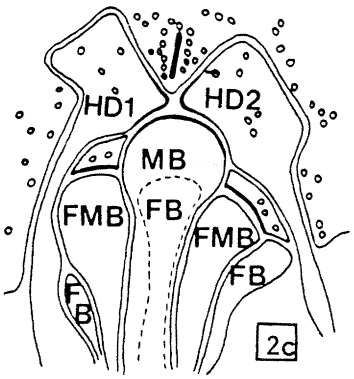
(A5) Bistratified amacrine cells, with a diameter of about 100mu., send horizontally distributed processes to the planes corneal and scleral to the inner plexiform layer.

GANGLION CELLS (SEE FIGURE 2A)

(G1) The midget ganglion cells are of two kinds, one with terminals in the outer third of the inner plexiform layer, and the other with terminals in the inner third. The middle third seems to be free of midget ganglion cell terminals. This fits well with the known distribution of the midget bipolar terminals (see above). There is probably a one-to-one


HD
MB
HA
MB
HA
FB
FMB
RB
FMA
RB
A3
A4
A1
A4
A2
G3
G1
G4
G4
G2
G1
G3
2a


HA2
HA1
RB2
RB1
2b


HD1
HD2
MB
FMB
FB
FMB
FB
FB
2c

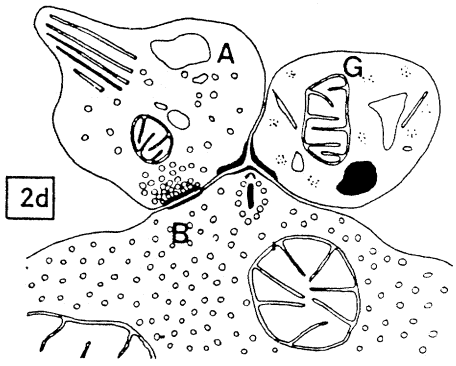

A
G
2d
B

FIGURE 2

correspondence between midget bipolar and midget ganglion cells.

(G2) Diffuse ganglion cells, dendritic diameters ranging from 30-75mu, the smaller diameters occuring nearer the fovea.

(G3) Stratified diffuse ganglion cells, like the stratified diffuse amacrine cells, are diffuse within the outer, middle, or inner third of the plexiform layer. There may be more in the outer third than in either of the others. Diameters range from 40mu near the fovea, to 80mu in the periphery.

(G4) Unistratified ganglion cells, occurring at all levels, have a diameter of about 200mu.

SYNAPSES OF THE INNER PLEXIFORM LAYER

The most common synaptic complex found in this region of the retina is the so-called dyad synapse (see figure 2D). At a dyad synapse, a bipolar cell contacts both a ganglion and an amacrine cell, and close by there is (probably) a further synapse from the amacrine cell back onto the bipolar terminal (Dowling and Boycott 1966). In addition to the dyad synaptic complex, amacrine to amacrine, and amacrine to ganglion dendrite synapses are seen.

The proportions in which the various types of synapse occur in the human retina are roughly as follow:

the complex of dyad + amacrine to bipolar: 3
amacrine to amacrine : 1
amacrine to ganglion : 1
bipolar to amacrine soma : 1/12

(from Dowling and Boycott 1966 table 1).

THE DIFFERENCE OPERATION

The receptor response and the horizontal cell response both are roughly linear over a large range. This is supported by the psychophysical findings of Alpern & Rushton (1967), Alpern, Rushton & Torii (1970a,b,c,d), (see also Alpern (1965)). Up to the stage just preceding the bipolar signal, therefore, retinal signals are probably linear functions of intensity; but beyond this, they may be logarithmic (see Werblin & Dowling's (1969) bipolar cell records).

Werblin & Dowling also found that the effect of stimulating the surround of a

bipolar cell receptive field is to remove part or all of the hyperpolarisation due to stimulating its center; but that surround stimuli could not on their own cause positive depolarisation of the bipolar cell. This finding, together with the fact that some bipolar cells have on-center receptive fields, and some have off-center ones, shows that the difference signal is split at this stage into its positive and negative parts, which are transmitted down two different channels. Kolb (1970) found that each cone contacts two midget bipolar cells (MB and FMB of figure 2): presumably these are the two channels. The separation of the positive and negative parts of the signal gives rise to some complexity in the later parts of this analysis; but it simplifies greatly the question of how a threshold might be applied to the difference signal. If the whole difference signal was carried by a single cell, zero would have to be coded as half the maximum response. Applyng a threshold that depends upon the absolute value of the transmitted signal is not easily done under such circumstances If the signal is split into positive and negative channels, however, it is merely a matter of applyng a threshold to the signals in each one.

Because we shall need to refer to the positive and negative parts of the signal, the following notation will be used:

$$\begin{aligned} POS(x) &= x \quad \text{if} \quad x > 0 \\ &= 0 \quad \text{otherwise.} \\ NEG(x) &= -x \quad \text{if} \quad x < 0 \\ &= 0 \quad \text{otherwise.} \end{aligned}$$

and similarly for POS(x'), POS(x"), POS(x*), etc.

THRESHOLDING THE DIFFERENCE SIGNAL

The heart of the computation is the application of a threshold to the difference signal, for it is this that removes the effects of slow changes in luminance from the image. There is some freedom in how the threshold is applied: for example, x" may be defined as in equation (4), for threshold t>0; or x" may be defined by the two expressions

$$POS(x'') = POS(x' - t) \qquad (6)$$
$$NEG(x'') = NEG(x' + t) \qquad (7)$$

which may be slightly easier to implement. The more important question is, however,

what size should the threshold t be?

In order to answer this question, let us consider the effects of doubling the illumination of a scene. The energy received from each point doubles, hence gradients double, and hence the necessary threshold must also double. This is, however, only a guide because bright sunlight produces views that are much more contrasty than the (roughly) uniform illumination provided by a thick layer of cloud. It will be important (when adequate transducers become available) to explore how the threshold should vary with lighting conditions: if it is set too high, valuable shading information may be lost. Until then, the linear approximation is the best available, and it is consistent with psychophysical evidence on contrast detection. If the bipolar (difference) signal were logarithmic, a fixed amount t subtracted from all signals in a bipolar axon would have the effect of equations (6) and (7).

RECONSTITUTING THE IMAGE

To reconstitute the image from the bipolar signals, it is necessary to solve a set of equations like those of (5) above. For the sake of precision, we limit ourselves to the analysis of the message from a single cone, via the midget bipolar, and the midget ganglion cells. The principles are, however, independent of the particular choice of image resolution.

A straightforward implementation of the equations (5) exists (see figure 1, and Horn AIM-295). In the retina, however, the difference signal is split into positive and negative channels; and the existence of on- and of off-center ganglion cells (Kuffler 1953) suggests that this is also true of the output signal. Because of this, and because the resting discharge frequency of amacrine cells appears to be zero (Werblin & Dowling 1969), it is taken as a basic constraint on the form of the reconstitution algorithm that the whole computation should be carried out using variables whose signs do not change. This constraint will allow each variable used during the computation to be coded by a nervous element in such a way that zero corresponds to the lowest extreme of its dynamic range. It is fortunately possible to be quite specific about the implementations that are consistent with the constraint: the argument will be divided into two parts. Firstly the basic constraint is used to select one of the various ways of splitting (5) into two halves: secondly, an explicit implementation is exhibited, together with an indication of which of its features are robust enough to support predictions by which the theory may be tested.

PERFORMING THE RECONSTITUTION IN TWO HALVES

Perhaps the most obvious way in which one might split equation (5) into two halves is the following, (w(y) is taken to be 1/n for simplicity):

$$x_1^* = POS(x'') + 1/2n \sum y^* \qquad (8)$$
$$x_2^* = NEG(x'') - 1/2n \sum y^* \qquad (9)$$

together with the condition that $x^* = (x_1^* - x_2^*)$. The basic constraint allows us to rule this out, because there is no reason why x_1^* and x_2^* should always be positive. For example, consider a situation where the local average lightness is high, and x" is small but negative. From equation (8), x_1^* is large and positive, and x_2^* is large and negative. Furthermore, if x" is negative, this information is carried by NEG(x"); the effect of x" in this case is to make x_2^* a little less negative. Therefore x_2^* does not satisfy the basic constraint; and this failure is important, in the sense that a coding of x_2^* that ignored its negative values would cause the reconstitution to fail in certain commonly occuring situations.

In order for a system to be consistent with the basic constraint, it will need to take something like the following form:

$$f(X) = POS(x'') + 1/n \sum f(Y) \qquad (10)$$
$$g(X) = NEG(x'') + 1/n \sum g(Y) \qquad (11)$$

where f and g are non-negative functions, that hopefully have some relation to the operator *. At first sight, this pair of equations does not appear to compute anything useful: but observe the following. Subtracting (10) and (11), we obtain:

$$(f(X) - g(X)) = (POS(x'') - NEG(x'')) + 1/n \sum (f(Y) - g(Y)) \qquad (12)$$

Now (POS(x") - NEG(x")) = x", so that (12) is the same as (5) where the operator has been replaced by the operator (f - g). Thus the expression (f(X) - g(X)) is in fact a solution of the equation (5). What this means is that a solution may be obtained from (10) and (11) provided that the two solutions are coupled in a subtractive way. In general, the two halves f(X) and g(X) will both be large and positive, since there is nothing negative in either of (10) or (11) to pull them down. The solution is, however, not disturbed by subtracting a suitable function h(X), provided that it is done to both f(X)

and g(X) simultaneously. To satisfy the basic constraint, h(X) must never exceed the smaller of f(X) and g(X): subject to this, however, a subtractive coupling between f(X) and g(X) is permissible.

Hence we obtain the result that (10) and (11), together with the operations:

$$f(X) \quad \text{goes to} \quad (f(X) - h(X)) \tag{13}$$
$$g(X) \quad \text{goes to} \quad (g(X) - h(X)) \tag{14}$$

still represents a solution to (5), as long as the condition

$$f(X) \quad \text{and} \quad g(X) \quad \text{are kept positive} \tag{15}$$

also holds. It is perhaps worth pointing out that the determination of h is quite separate from the problem of fixing the DC level for the output: nor can variations in h account for the disappearance of stabilized retinal images, since this would correspond to tampering with the difference (f(X) - g(X)): as long as this difference is preserved, the output of the process is a faithful processed copy of the image.

IMPLEMENTATION DETAILS

The method outlined by equations (10) - (15) leads to an explanation of many features of the inner plexiform layer. There are two basic problems that need to be discussed: firstly, how does one implement the set of linear equations represented by equation (10) or (11); and secondly, how exactly can the subtractive coupling between the two halves (10) and (11) be done?

The first question is the more straightforward. The realisation of equation (10) requires a device with reciprocal connexions to devices that compute the solution at neighboring points (see figure 1). Since the retinal ganglion cells are not pre-synaptic to any other retinal cells, the expression of, for example, f(X) cannot exist only at a ganglion cell, because if it were, it would not be available for the computation at neighboring ganglion cells. Hence if f(X) is computed in the retina, it will be found either in the bipolar cell axon terminals, or in the amacrine cells, or both.

The two kinds of midget bipolar cell (FMB and MB) terminate in the top and the bottom thirds respectively of the inner plexiform layer (Boycott & Dowling 1969, Kolb 1970). The additive coupling between solutions of f(X) at neighboring points must therefore be provided by stratified amacrine cells that are driven by, and drive, the

bipolar terminals. From figure 2, we see that the stratified amacrine cells coupled in this way with the midget bipolar cells must be those of category A3. The only candidate for the place at which f(X) is computed, which is consistent with retinal structure, is therefore the bipolar cell axon terminals.

The hypothesis, that additive coupling between neighboring points is provided by the stratified amacrine-bipolar cell interaction, is consistent with the observed synaptic arrangements, because the complex of a dyad synapse plus a return synapse from the amacrine cell to the bipolar cell (figure 2d) is exactly what it requires. It furthermore follows from this interpretation, that the signs of all the synapses in the dyad complex should be positive, from the point of view of the computation. In practice, this means that the bipolar to stratified amacrine, and the bipolar to midget ganglion cell synapses should be excitatory; and that the stratified amacrine to bipolar synapse should have the same effect on the midget bipolar terminal as stimulation of the center of that cell's receptive field. It is also necessary that a large proportion of the dyad synapses with a midget bipolar cell involve stratified amacrine cells, and one would expect all dyad synapses that do involve a stratified amacrine cell to be accompanied by a synapse from the amacrine cell back to the midget bipolar.

The final point that concerns the additive interactions is the size of the stratified amacrine cell processes. In order for the inverse computation to proceed accurately, these cells should reflect the dimensions of the horizontal cells that, we suppose, are responsible for the difference operation. The diameter of the A3 amacrine cells is 20-50mu, which is consistent with the figure of about 35mu for the radius of the horizontal cell interaction (Boycott & Dowling 1969, Kolb 1970).

AN EXPLICIT MODEL FOR THE SUBTRACTIVE COUPLING

The second issue concerns the subtractive coupling between the plus and the minus halves of the solution. It is clear that this coupling must be provided by diffuse amacrine cells, probably of category A1, since the only other candidates (A5) are both large and uncommon; and it is probable that the coupling is achieved through amacrine-amacrine and amacrine-ganglion cell synapses. This coupling can be achieved in a number of ways, but it is helpful to exhibit one of them, because variants on the method are then rather easy to understand.

Equations (10) and (11) may be modified to be the following:

$$f(X) = POS(x'') + POS(1/n \sum (f(Y) - g(Y))) \qquad (10')$$

$$g(X) = NEG(x'') + POS(1/n \sum (g(Y) - f(Y))) \qquad (11')$$

Then (f(X) - g(X)) is still a solution. Now write:

$$MG+(X) = POS(f(X) - g(X))$$
$$MG-(X) = POS(g(X) - f(X)) \qquad (12')$$

Then one of MG+(X), MG-(X) will be positive, and will represent the solution.

The idea behind this formulation is that the subtractive interaction is done in two ways, and it is illustrated in figure 3. Firstly, there are inhibitory connexions acting upon the stratified amacrine cells in the top and the bottom thirds of the inner plexiform layer. In figure 3, this is provided by the diffuse amacrine cells (A1) that receive excitation from a midget bipolar terminal (or possibly from A3 cells) in one layer, and send inhibitory synapses to the stratified amacrine cells in the other. Secondly, there is direct inhibition driven by the bipolar terminal in one layer, acting on the midget ganglion cell in the other.

The important qualitative features of this arrangement are:

(a) The form of the coupling between the two systems of stratified amacrine cells (i.e. the number of terms collected under the POS function), is very flexible; it must however correspond closely in both layers.

(b) The sizes of f(X) and of g(X) (the quantities in the bipolar terminal) are kept positive. This is vital for allowing a POS(x") signal to influence the solution even if the solution at X is strongly negative i.e. MG-(x) is strongly positive. This requirement is so important that it forbids the existence of any significant negative connection from a diffuse amacrine cell directly onto the bipolar terminal: it is true of all models, because conditions must never totally prevent the signal POS(x") from being able to influence the computation.

(c) The diffuse amacrine cells should receive excitatory synapses from the midget bipolar cells, and possibly from the stratified amacrine cells, in one layer; they should send inhibitory synapses to the stratified amacrine and to the midget ganglion cells in the other. (This is a requirement of all models that implement the coupling between the layers, because it has to be provided by diffuse amacrine cells, and the theory requires that the coupling be subtractive.) It is interesting that according to this method, the synapse from a midget bipolar to a diffuse amacrine cell need not be accompanied by a reciprocal synapse, whereas that to a stratified amacrine cell should be.

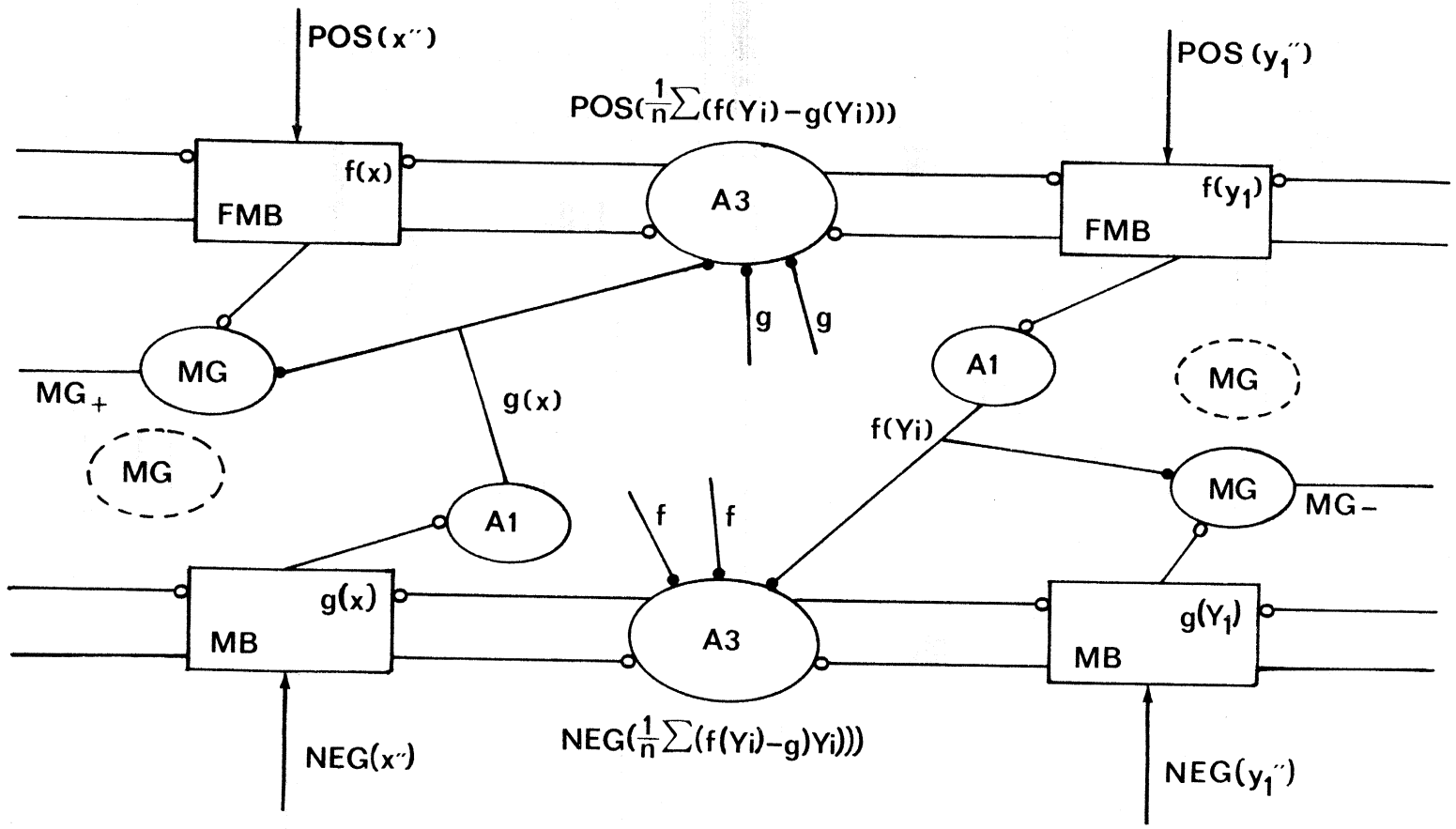

POS(x'')
POS($\frac{1}{n}\sum(f(Yi)-g(Yi))$)
POS(y_1'')
f(x)
FMB
A3
f(y_1)
FMB
g
g
MG
MG+
g(x)
A1
MG
MG
f(Yi)
MG
MG-
A1
f
f
g(x)
MB
A3
g(Y_1)
MB
NEG(x'')
NEG($\frac{1}{n}\sum(f(Yi)-g)Yi)$)
NEG(y_1'')

FIGURE 3

UNIQUENESS

The arguments that were set out above impose some constraints upon the way in which the reconstitution stage (5) may be implemented, but they fall short of establishing that a particular method must be the one that is used. There are, however, two arguments -- one compelling and one strong -- that greatly constrain the methods that are consistent with retinal structure: they depend upon details that would be difficult to express without having exhibited one method completely. Firstly, should the vertical interaction between the two halves of the solution occur before the lateral interactions? In other words, are both halves of the difference signal, POS(x") and NEG(x"), available to both halves of the inverse transform; or is the bipolar signal necessarily contaminated by lateral interactions before it can escape to the other layer? The latter is correct, because the lateral interactions must be carried by the stratified amacrine cells, which send synapses back to the midget bipolar axon terminals. This means that a formulation is appropriate in which interaction terms between the two layers already contain components due to the lateral interactions.

The second question concerns the degree to which the lateral interactions are "complete" before the vertical interactions begin. In the formulation of equations (10) -- (15), the lateral interactions are represented as being complete before the vertical interaction, the subtraction from both layers of h(X), is carried out. This extreme position is undesirable, because f(X) and g(X) will be large compared with (f(X) - g(X)). In the explicit neural model of figure 3, the vertical and the lateral interactions proceed simultaneously. The strong argument for this is that the numbers will all be kept small, thus aiding accuracy; but a variety of possibilities lie between these two extremes.

The summarizing diagram shown in figure 4 puts the whole model together and illustrates the computation of lightness in which the inverse is performed in the manner described by equations (10) -- (15). The analysis of two stimuli is shown: one, a light spot on a dark background, and the other, a dark spot on a light background.

SOME RELEVANT READING

Alpern, M. and Rushton, W.A.H. "The Nature of Rise in Threshold Produced by Contrast-Flashes." J. Physiol., 189 (1967), 519-534.

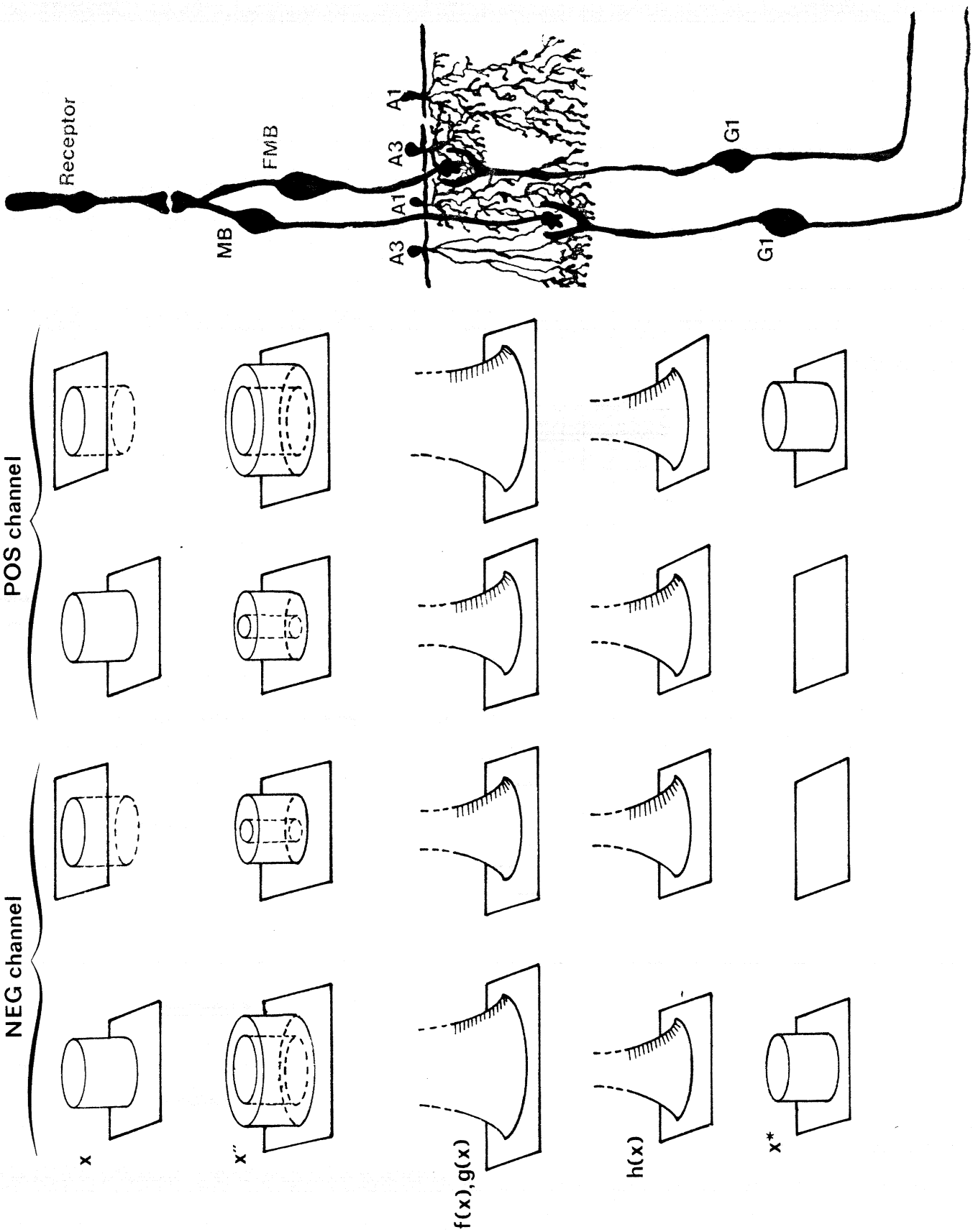

Receptor
MB
FMB
A3
A1
A1
A3
G1
G1
POS channel
NEG channel
x
x''
f(x),g(x)
h(x)
x*

FIGURE 4

Alpern, M., Rushton, W.A.H. and Torii, S. "The Size of Rod Signals." J. Physiol., 206 (1970A), 193-208.

Alpern, M., Rushton, W.A.H. and Torii, S. "The Attenuation of Rod Signals by Backgrounds." J. Physiol., 206 (1970B), 209-227.

Alpern, M., Rushton, W.A.H. and Torii, S. "The Attenuation of Rod Signals by Bleachings." J. Physiol., 207 449-461. Alpern, M., Rushton, W.A.H. and Torii, S. (1970D). "Signals from Cones." J. Physiol., 207 (1970C), 463-475.

Barlow, H.B. "Retinal Noise and Absolute Threshold." J. opt. Soc. Amer., 46 (1956), 634-639.

Barlow, H.B. "Increment Thresholds at Low Intensities Considered as Signal-Noise Discriminations." J. Physiol., 136 (1957), 469-488.

Barlow, H.B. "Temporal and Spatial Summation in Human Vision at Different Background Intensities." J. Physiol., 141 (1958), 337-350.

Barlow, H.B., Fitzhugh, R. and Kuffler, S.W. "Change of Organization in the Receptive Field of the Cat's Retina During Dark Adaptation." J. Physiol., 137 (1957), 338-354.

Barlow, H.B. and Sakitt, B. "Doubts About Scotopic Interactions in Stabilized Vision." Vision Res., 13 (1973), 523-524.

Boycott, B.B., and Dowling, J.E. "Organization of the Primate Retina: Light Microscopy." (with an appendix by H. Kolb). Philos. Trans. Roy. Soc. 255 (1969), B., 109-184.

Brindley, G.S. Physiology of the Retina and Visual Pathway. (Physiological Society Monograph no. 6). London: Edward Arnold, Ltd., (1970).

Cajal S.R. Histologie du Systeme, Nerveux. Madrid: C.S.I.C. (1911).

Cleland, B.G., Dubin, M.W. and Levick, W.R. "Sustained and Transient Neurones in the Cat's Retina and Lateral Geniculate Nucleus." J. Physiol., 217 (1971), 473-496.

Cone, R.A. and Ebrey, T.G. "Functional Independence of the Two Major Components of the Rod Electroretinogram." Nature, 221 (1965), 818-820.

De Valois, R.L. "Analysis and Coding of Colour Vision in the Primate Visual System." Cold Spr. Harb. Symp., 30 (1965), 567-579.

Dowling, J.E., and Boycott, B.B. "Organization of the Primate Retina: Electron Microscopy." Proc. Roy. Soc. B., 166 (1966), 80-111.

Dowling, J.E. and Werblin, F.S. "Organization of the Retina of the Mud-Puppy." Necturus maculosus: I, Synaptic structure. J. Neurophysiol., 32 (1969), 315-338.

Enroth-Cugell, C. and Robson, J.G. "The Contrast Sensitivity of Retinal Ganglion Cells of the Cat." J. Physiol., 187 (1966), 517-552.

Fukada, Y. "Receptive Field Organization of Cat Optic Nerve Fibres with Special Reference to Conduction Velocity." Vision Res., 11 (1971), 209-226.

Fukada, Y. and Saito, H-A. "The Relationship Between Response Characteristics to Flicker Stimulation and Receptive Field Organization in the Cat's Optic Nerve Fibres." Vision Res., 11 (1971), 227-240.

Gouras, P. "Rod and Cone Independence in the Electroretinogram of the Dark-Adapted Monkey's Perifovea." J. Physiol., 187 (1966), 455-464.

Gouras, P. "The Effects of Light-Adaptation on Rod and Cone Receptive Field Organization of Monkey Ganglion Cells." J. Physiol., 192 (1967), 747-760.

Gouras, P. "Identification of Cone Mechanisms in Monkey Ganglion Cells." J. Physiol., 199 (1968), 533-547.

Gouras, P. and Link, K. "Rod and Cone Interaction in Dark-Adapted Monkey Ganglion Cells." J. Physiol., 184 (1966), 499-510.

Helmholtz, H. (1962) Treatise on Physiological Optics. New York: Dover Publications Inc. (First edition of Handbuch der Physiologischen Optik published in 1867 by Voss, Leipzig), (1960).

Hubel, D.H. and Wiesel, T. N. "Receptive Fields of Optic Nerve Fibres in the Spider Monkey." J. Physiol., 154 (1962), 572-580.

Hubel, D.H. and Wiesel, T.N. "Spatial and Chromatic Interactions in the Lateral Geniculate Body of the Rhesus Monkey." J. Neurophysiol., 29 (1966), 1115-1156.

Kaneko, A. and Hashimoto, H. "Recording Site of the Single Cone Response Determined by an Electrode Marking Technique." Vision Res., 7 (1967), 847-851.

Kolb, Helga. (1970) "Organization of the Outer Plexiform Layer of the Primate Retina: Electron Microscopy of Golgi-Impregnated Cells." Philos. Trans. Roy. Soc. B., 258 (1970), 261-283.

Kuffler, S.W. "Discharge Patterns and Functional Organization of Mammalian Retina." J. Neurophysiol., 16 (1953), 37-68.

Land, E.H. "The Retinex." Am. Scientist, 52 (1964), 247-264.

Land, E.H. and McCann, J.J. "Lightness and Retinex Theory." J. opt. Soc. Amer., 61 (1971), 1-11.

Lennie, P. and MacLeod, D.I.A. "Background Configuration and Rod Threshold." J. Physiol., 233 (1973), 143-156.

McIlwain, J.T. "Receptive Fields of Optic Tract Axons and Lateral Geniculate Cells: Peripheral Extent and Barbiturate Sensitivity." J. Neurophysiol., 27 (1964), 1154-1173.

McIlwain, J.T. "Some Evidence Concerning the Physiological Basis of the Periphery Effect in the Cat's Retina." Exp. Brain Res., 1 (1966), 265-271.

McKee, S. and Westheimer, G. "Specificity of Cone Mechanisms in Lateral Interactions." J. Physiol., 206 (1970), 117-128.

Maffei, L. and Fiorentini, A. "Retinogeniculate Convergence and Analysis of Contrast." J. Neurophysiol., 35 (1972), 65-72.

Missotten, L. The Ultrastructure of the Human Retina. Brussels: Arscia Uitgaven N.V. 1965.

Naka, K.I. "Computer Assisted Analysis of S-Potentials." Biophys. J., 9 (1969), 845-859.

Naka, K.I. and Rushton, W.A.H. "S-Potentials from Colour Units in the Retina of Fish (Cyprinidae)." J. Physiol., 185 (1966), 536-555.

Naka, K.I. and Rushton, W.A.H. "The Generation and Spread of S-Potentials in Fish (Cyprinidae)." J. Physiol., 192 (1967), 437-461.

Naka, K.I. and Rushton, W.A.H. "S-Potential and Dark Adaptation in Fish." J. Physiol., 194 (1968), 259-269.

Ratliff, F. Mach Bands: Quantitative Studies on Neural Networks in the Retina. San Fransisco: Holden-Day, 1965.

Rodieck, R.W. "Receptive Fields in the Cat Retina: a New Type." Science, 157 (1967), 90-92.

Rushton, W.A.H. "Pigments and Signals in Colour Vision." (Invited lecture to the Physiological Society). J. Physiol., 220 (1972), 1-31.

Stone, J. and Dreher, B. "Projection of X- And Y- Cells of the Cat's Lateral Geniculate Nucleus to Areas 17 and 18 of Visual Cortex." J. Neurophysiol., 36 (1973), 551-567.

Stone, J. and Hoffman, K-P. "Very Slow-Conducting Ganglion Cells in the Cat's Retina: a Major. new functional type?" Brain Res., 43 (1972), 610-616.

Tomita, T. "Electrical Responses of Single Photoreceptors." Proc. IEEE, 56 (1968), 1015-1023.

Toyoda, J., Nosaki, H. and Tomita, T. "Light-Induced Resistance Changes in Single Photoreceptors of Necturus and Gekko." Vision Res., 9 (1969), 453-463.

Weber, E.H. De Pulsu. resorptione, Auditu et Tactu Annotationes Anatomicae et Physiologicae. Leipzig: 1834.

Werblin, F.S. and Dowling, J.E. "Organization of the Retina of the Mud-Puppy." Necturus Maculosus: II, Intra-Cellular recording. J. Neurophysiol., 32 (1969), 339-355.

Westheimer, G. "Rod-Cone Independence for Sensitizing Interaction in the Human Retina." J. Physiol., 206 (1970), 109-116.

Westheimer, G. and Wiley, R.W. "Distance Effects in Human Scotopic Retinal Interaction." J. Physiol., 206 (1970), 129-143.

Zeki, S.M. "Colour Coding in Rhesus Monkey Prestriate Cortex." Brain Res., 53 (1973), 422-427.

6 PRODUCTIVITY TECHNOLOGY

Work reported in this section is grouped into three subsections: hardware and software, arm theory, and understanding circuit boards.

The first of these three parts describes some of the interim and permanent equipment constituting the low-cost "mini-robot" laboratory concept.

THE MINI-ROBOT

The mini-robot has evolved into a set of modules now supporting substantive research. Since the technical details of these modules are to be reported in a separate document and since those details are not of interest to the general reader, we confine this section to an enumeration of the major components now operational.

1. A six-degree-of-freedom Scheinman arm designed specifically for this project (see figure 1). Two more will be added.

2. An Ikon X-Y table driven by digital motors.

3. A PDP-11/40 computer with two disk drives and a high-speed connection to the laboratory's PDP-10 time-sharing system.

4. Eight D/A's and sixty-four A/D's by Analogic for the arm and other devices. More D/A's of our own design are under construction.

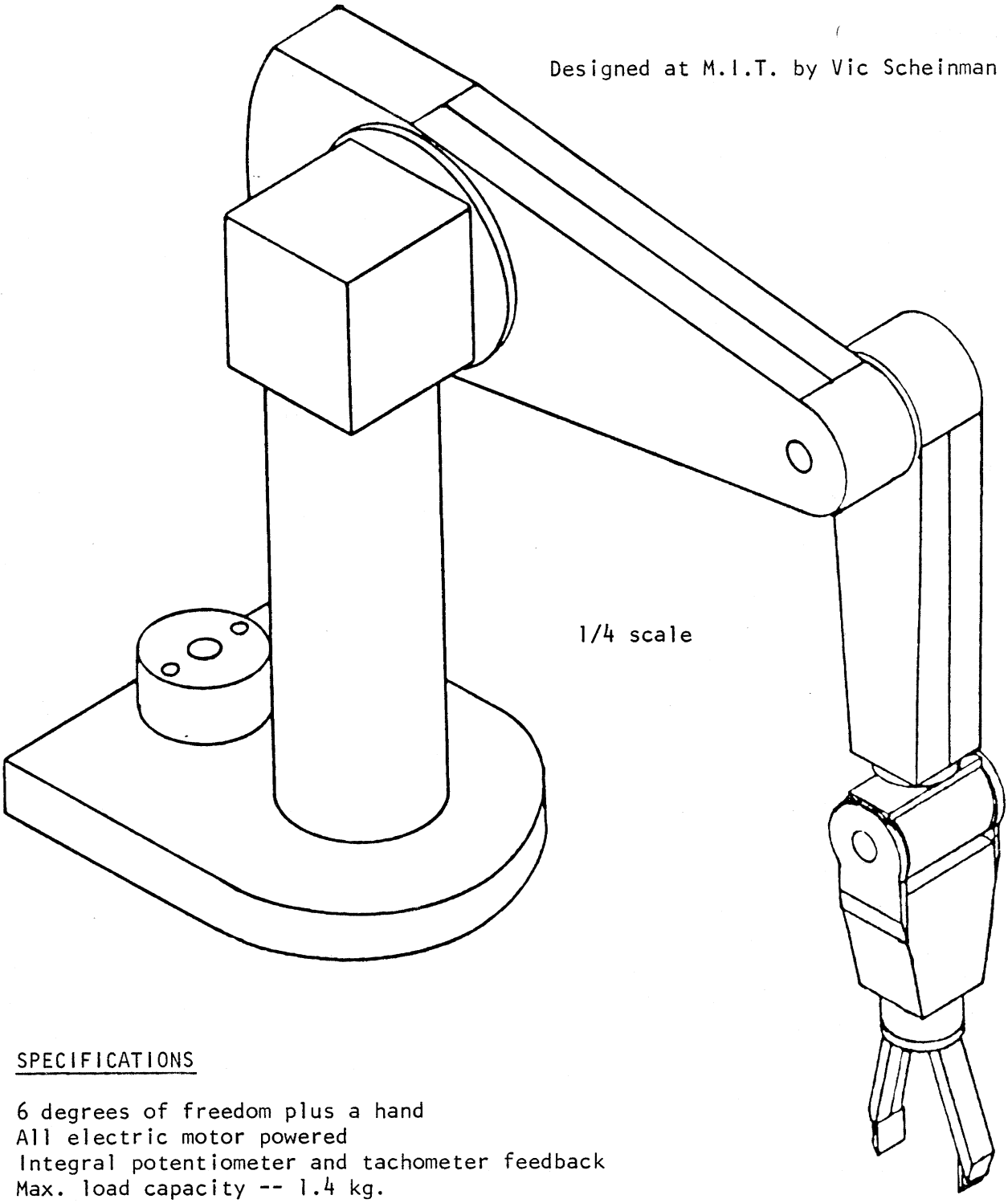
5. A Media Systems' vidicon with interface for picking off a specified point per line of scan so as to match video speed with computer memory speed.

6. Several scanning mirrors for use in deflecting our low power laser and for use in a low cost, high quality P.I.N. diode and/or linear array imaging device.

7. A Redicon 512 element linear array with a 16 level interface.

8. A GT-40 terminal for programming and display of grey level images.

Designed at M.I.T. by Vic Scheinman



1/4 scale

SPECIFICATIONS

6 degrees of freedom plus a hand
All electric motor powered
Integral potentiometer and tachometer feedback
Max. load capacity -- 1.4 kg.
Speed -- 1 second to complete most simple motions
Resolution -- 1.5 mm.
Accuracy -- 3mm.
Workspace -- 20 cm. radius hemisphere

FIGURE 1

SILVER'S ARM AND MINSKY'S WRIST

While waiting for the advanced Scheinman 6-degree-of-freedom arm, manipulation work has progressed through experiments with David Silver's mini-robot, which incorporates a six-sensor wrist invented (or perhaps reinvented) by Minsky. The degrees of freedom are split in Silver's configuration between (1) an X-Y table bearing a rotating vise, and (2) an overhead gripper which rotates and moves up and down. These motions provide five degrees of freedom rather than the fully general six, but nevertheless very successful experiments in force sensitive motion have been carried out. Indeed success with this configuration has led to current considerations of further developments along the same line.

One of Silver's first programs using his mini-robot duplicated earlier Japanese crank-turning demonstrations -- a task which a purely position oriented system does poorly since efforts to correct position errors tend to rip a crank out of its housing. Silver was also able to spin nuts onto bolts using the same priniciples.

Very recently Hiroshika Inoue, with collaboration from Silver, has devised a program in LISP for putting together a radial bearing assembly consisting of a shaft, two ball races, two spacers, a housing, and a nut. These pieces fit together with real-world tolerances of between one and three mils.

BILLMER'S SOFTWARE

The mini-robot's computer system consists of a PDP-11 processor with 24K of core and the extended instruction set (EIS) option, a GT40 graphic display terminal and associated PDP 11/05 processor with 8K of core, two RK05 disks with 1.2 million words of bulk storage apiece, a high speed paper tape reader and a General Electric Terminet 1200 printer/keyboard. There is also a 4800 baud TTY line to the A.I. PDP-10 system.

Meyer Billmers, in collaboration with Richard Waters and others, has put together a considerable amount of PDP-11 software for support of the system. This software includes the following samples, some of which are now ready for sharing with other PDP-11 users:

LISP

PDP-11 LISP provides the user with many of the features of a full scale LISP. Atom print names are restricted to being three characters in length, and the print names

should be alpha numeric. This is a shallow binding LISP with saved values of the variables kept on the stack along with function calls and associated information (such as function return addresses and parameters being passed).

RUG

Rug is a symbolic debugger. It was initially developed by the laboratory's education research group.

VIDIN

VIDIN is a program used to take pictures from the Vidicon scanner, digitize them, and store them as contiguous (random access) files on disk.

MAPPER

MAPPER is a program for inspection of the picture files taken by VIDIN. In MAPPER one can select a specific sector of a picure file, obtain statistics about that sector (such as maximum and minimum intensities, average intensity and standard deviation of the intensity distribution). It can also obtain a histogram of the intensity values for a sector or a single x or y value of a sector, and then make a map of the sector.

WATER'S ARM DYNAMICS THEORY

At this point we move on to three sections which deal with some theoretical issues that come up in arm control and design. The first item is Water's approach to arm control. We concentrate here on non-mathematical issues, but we want to emphasize in passing that Richard Waters has had considerable success in speeding up the exact solution of the complex equations of arm dynamics. It seems very likely that his methods will enable fast calculation of the motor torques required to produce a given set of arm joint accellerations -- fast enough indeed to do the calculation while the arm is actually making the required move.

SOME EXAMPLE SCENARIOS

In order to clarify some issues involved with arm control, scenarios of four typical arm actions are given. Each scenerio indicates:

1. What information is needed to initiate the action.
2. What additional information must be developed by the arm controller in order to perform the action.
3. Some of the local errors the arm controller can correct itself.
4. Some of the errors it is not expected to handle.

MOVE TO POINT B

This first action is perhaps the most basic arm motion. The arm is at point A, and it is asked to move along a smooth path to point B without hitting anything. Only point B needs to be specified in order to start the action. The arm controller has to do a lot of work:

(a) It has to decide on an exact path from A to B that the arm is capable of following without any high acceleration which would make the motion jerky.

(b) The system should also have some basic ideas about object avoidance. For instance, if A and B are both on the table, and far apart, then the arm controller could plan a path that comes up off of the table in an arc, not a path that skims over the table. Here is a place where a model of the environment would be very useful, but how much model is enough and of how hard would it be to keep accurate? The problem can be avoided if the higher level programs have a good model of the arm, and how the arm controller plans trajectories. If the higher level concludes that a (MOVETO B) command

would cause the arm to hit something then it just needs to break the move into two parts (MOVETO C) (MOVETO B) which avoid the obstacle. It seems that the simple heuristic of moving up in an arc on a motion will avoid most collisions.

Turning to error situations, the main error the controller is expected to be able to deal with is the drift discussed above. In short, unless the arm strikes something, the controller will get it to B one way or another. Note that if position B is not a possible configuration of the arm, the controller will detect this through its model of the actual arm and complain to the higher level.

The arm controller is not expected to handle the problem of striking something on the way to B. In general it has to appeal to higher authority in order to determine if the arm has done any damage. It could have a criterion, such that if a collision was soft enough it would assume that the object was undamaged, and then try to avoid it and continue on as though nothing had happened

In this case the controller must be careful that it does not go into an infinite motion trying to avoid an object. A particularly pathalogical case occurs if point B is in an object.

PUSH A BLOCK ASIDE

Here is another rather basic motion. A block with dimensions D is at position A and the arm is asked to push it a distance d in direction R. The request must specify A, d, R, and at least some information about D, in order to start the action.

The arm controller is going to have to figure out:

(a) How the hand is to approach the block. In general, it should be moving along R towards the center of mass of the block (point A) with the longest axis of the hand perpendicular to the motion (so that the block will not rotate when contact is made with the hand, and will move in direction R).

(b) It must also decide how much force to apply to the block. Again the local errors the controller can deal with are drift-like.

(a) There is drift in the motions.

(b) In addition several problems can arise which are very similar to drift. For instance if the block sticks, the arm tries a little more force; if the block skates along it tries a little less force. More interesting, if the block slides off of the hand, and loses contact, then the arm must back up a little move over and try again (as long as the arm does not move too far afield). The arm controller can deal with each of these problems by simply doing what it was doing before with some parameters updated.

The kind of global errors the controller cannot handle are typified by:

(a) The arm hits something.

(b) The block hits something and stops moving (there is a sudden large rise in force).

(c) When the arm gets to A it does not find any block (the arm controller must be careful not to push the wrong thing).

PUTTING A RESISTOR'S LEADS INTO A CIRCUIT BOARD

This is a much more complex action. Let us assume that the leads are already bent and cut. The arm is asked to pick up a resistor and insert the two leads into the proper two holes. Here the position (and orientation) of the resistor, and the positions of the holes need to be specified.

This is a task requiring considerable precision of the arm system. First it must pick up the resistor very accurately so that it will have a reasonable idea of the position of the ends of the leads. Then it must move the end of one lead to the first hole and insert it. Finally it must get the other lead in the other hole. All through this the arm controller must decide how fast to move, how hard to push.

Some of the errors the arm can handle in this action are so expected, they will probably always happen. In particular, the arm is probably not so precise that it can just put a lead in a hole. However, the arm should be able to get the lead near the hole. (If it does not, the lead may end up in the wrong hole.) The arm must drag the lead over the board trying to find the hole. It should not do this for too long, since the hole might be missing or plugged. In either case the arm must ask for help from above. It cannot even tell what went wrong: is the hole missing or did the arm miss it?

This whole procedure could be simplified if the arm system could ask specific questions of the vision system (such as: "where is the end of the lead?") and get a quick answer. This is the one place where it appears that a small increment in communication could yield important gains in the power of the arm system. For most problems, total communication is needed and partial communication is not helpful.

Here are a couple of other errors the arm system could not handle without extensive communication:

(a) If the arm drops the resistor it is probably not going to be able to find it. It might pick up the wrong thing.

(b) While trying to locate the hole, one of the leads might bend. The arm system probably would not even detect this except through finding itself unable to get the leads

in the board.

This points up the important fact that while the arm system is working, the rest of the robot system should be looking on, even if there is no direct feedback, so that it can detect errors the arm would miss. Also, the eye must check that things were done correctly after the arm is finished.

TIGHTENING A NUT WITH A WRENCH

It is possible that if the arm was strong enough it would not need to use a wrench to tighten a nut, but let us assume it did need one. There is a wrench at location A, and a bolt at location B with a nut already started and partly tightened with the hand (this is another difficult task). The action is a repeated cycle of putting the wrench on the nut (this is the hard part), swinging the wrench through an arc, and taking the wrench off the nut. The action stops when the nut is tight.

To start the action the request must specify the position of the nut and bolt and of the wrench. An interesting variation would be for the request to specify the size of the nut, and have the arm controller remember where the tools are. This view looks at the tools as part of the arm.

As always the arm system must develop specific trajectories for all of the motions involved. Here are some points of particular interest in this action:

(a) When the arm picks up the wrench it must pick it up in a very precise manner, so that it will know where the end of the wrench is. In general the arm system should have a good model of the arm wrench combination, so that it can perform tasks like putting the wrench on the nut easily.

(b) The next difficult point is putting the wrench on the nut. People often do this with two hands. In order to do it with one, the arm will have to be able to accurately rotate the mouth of the wrench about the center of the nut waiting for the wrench to slip on.

A special tool like a socket wrench or a nut driver would simplify the scenerio because the arm would only have to put the wrench on the nut once.

(c) Next force feedback is used to tighten the nut by swinging the wrench. The arc the wrench can be swung through will usually be constrained by obstacles, leaving a pie shaped region in which the wrench can operate. This region must be at least 60 degrees in size for a typical wrench, and should be specified in the initial request.

(d) An interesting case of local error which arises in this task is when the wrench pops off of the nut. The arm just puts it back on and continues.

(e) It is interesting to note that the arm cannot tell whether it was successful in tightening a nut, or whether the bolt is cross threaded.

In all these tasks, the arm controller can only correct local errors which do not interact with anything else it is trying to do, unless the arm has access to accurate models of what is going on.

THE INTERNAL STRUCTURE OF WATER'S ARM CONTROLLER

Let us summarize the inputs and outputs of the Scheinman arm hardware.

A) Inputs from arm controller (direct effects of controller)
   (1) Motor torques (at each joint (6))
   (2) Joint lockers (at each joint (6))
   (3) Hand control inputs (open close)
B) Output to controller (observables)
   (1) Internal sensors
      (a) Joint positions (at each joint (6))
      (b) Joint velocities (maybe)
   (2) External sensors
      (a) Force sensors in wrist
      (b) Touch sensors on hand
C) Ouputs to the environment (direct effects of arm)
   (1) Motion through environment
   (2) Forces applied to objects in the environment
D) Inputs from environment (external sensation)
   (1) Due to contact with objects in the environment
      (a) Force sensation
      (b) Touch sensation

A notational convenience, the ARM STATE VECTOR (ASV) formed mainly of the sensor outputs from the arm to the controller (i.e. the observables) is introduced.

ASV = (position, velocity, force, touch, hand, lockers; time).

This vector contains all of the information that the arm controller has about the state of

the arm without getting additional information from the rest of the robot system. The information in the ASV is directly as it comes from the sensors. It is in the generalized coordinates appropriate to the arm (i.e. the joint angles and velocities).

Similarly the ARM CONTROL VECTOR (ACV) which gathers together the control inputs to the arm (i.e. the direct effects of the controller on the arm) is introduced.

ACV = (torques, joint lockers, hand control; time)

Note that these two vectors are not unrelated.

ASV(T+dT) = F(ASV(T), ACV(T), ENV(T))

That is, the new state is a function of the old state, the control inputs, and the environment. In order to control the arm, the controller attempts an inversion of this relation.

ACV(T) = G(ASV(T), ASV(T+dT), ENV(T))

Unfortunately this inversion can only be partial for several reasons:

(a) The controller has very little knowledge of the environment (ENV(T)). Therefore it continually runs the risk of encountering unexpected difficulties. Each command to perform an action gives the minimal description of the environment that is necessary for the execution of the action, but no more.

(b) A less important problem is that G is not single valued. The controller must often choose between several ways of making a state transition.

(c) More unfortunately, G is not a total function. Many (in fact most) state transitions are not possible at all. Intuitively this is clear because while the ASV has 18 independently varying major components (6 positions, 6 velocities and 6 forces), the ACV has only 6 independently varying major components (the 6 torques). Thus in general, the controller can only control a 6-dimensional subspace of the ASV in one time interval.

(d) Lastly, even when G is computable, it can be computed only approximately due to the great mathematical complexity. This is a major source of drift in the arm's actions.

Stepping back a moment, consider that the input to the controller from the robot system is a high level command such as (MOVETO B), while the input to the arm is summarized in ACV(T). This suggests a two stage controller, with one stage that transforms the high level command into ASV(T+dT) and the other that computes ACV(T) by using G.

TWO STAGE CONTROLLER

The belief that having a two stage controller will be very convenient is based on the following assumptions on the interactions of the two stages depicted.

(a) First, it is assumed that the procedural level which transforms a high level command into a series of states of the arm does not need to know how the state transitions can be realized.

(b) Second, it is assumed that when the dynamic level is evaluating G, in order to calculate how to get from state A to state B, it does not need to know why the procedural level wants to make the state transition or where the arm is going next.

It seems that in general these assumptions are valid. There are several clear advantages to the two stage design.

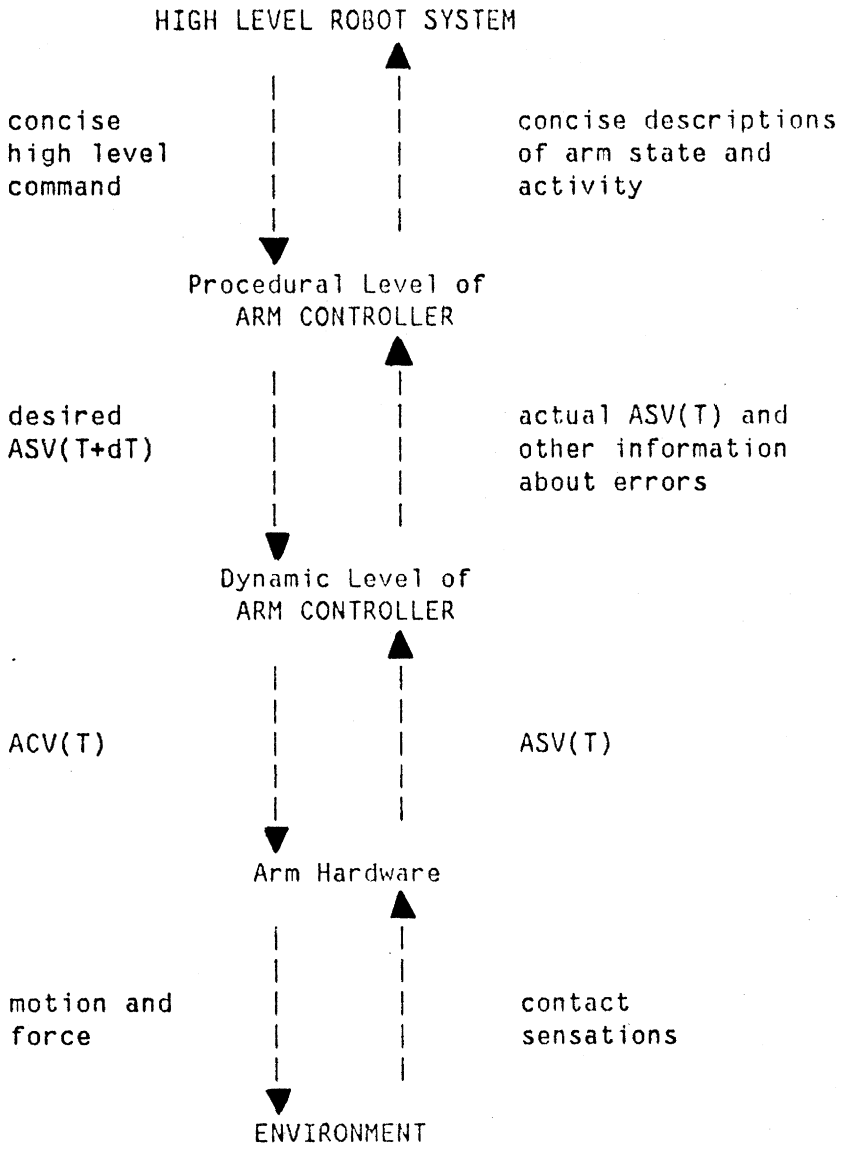
(a) The computation of G is very complex, and elaborate approximations must be used. Thus it is very important to isolate the calculation of G in one place where it can be done with the greatest possible efficiency.

(b) The division also makes the system clearer and more tractable. The dynamic level can be looked at as an emulator which makes the arm hardware look like it takes desired ASVs as inputs instead of ACVs. Thus from the point of view of the procedural level, the direct effects have been improved. It never has to deal with the complex mathematics in G. Nor does it ever have to deal with the generalized arm coordinates. It can work in coordinates suited to the action, not the arm.

(c) Another important point involves the 'intelligence' of the mini-robot as a whole. In order to 'understand' what it is doing, it must have some way of elucidating the inner structure of its actions. But how far should this decomposition be allowed to progress?

With the two stage controller, the answer is clear. The dynamic level operates as a black box, the complex mathematical knowledge in it forever hidden from the rest of the system. The procedural level on the other hand has all of its knowledge symbolically represented in procedures which are composed of named primitive procedures.

Thus the mini-robot can introspect down to the level of these primitive procedures, and no farther. This closely parallels the human situation. A person can introspect about his motions down to a certain level, but no farther. For example, most people think of wiggling their ears as a primitive action (even though it is produced by the combined action of many muscles), because they are powerless to control these muscles below a certain level. Most people can wiggle both ears, but very few can wiggle just one, even though there are separate groups of muscles on each side. This is because they cannot decompose the action. They have no 'names' for its constituents -- no way to get a hold



A diagram of the two stage arm controller showing the information communicated between levels.

on them.

(d) A final point is that some time in the future, the entire dynamic level could be implemented in hardware, increasing its speed by switching to analog methods of computing G.

THE DYNAMIC LEVEL OF THE ARM CONTROL SYSTEM

Here is an input/output description of the dynamic level:

(A) Outputs to the arm hardware (1) ACV(T) the control inputs

(B) Inputs from the arm hardware (1) ASV(T) the current state of the arm

(C) Inputs from the procedural level

  (1) The desired ASV (DASV) at the next time interval DASV(T+dT)

(D) Outputs to the procedural level

  (1) ASV(T)

  (2) DELTA(T) this is a concise statement of the difference between ASV(T) and DASV(T). It is expected that they will not be the same because the dynamic level's evaluation of G is only approximate, and it has almost no knowledge of ENV(T).

  (3) When an error condition arises

    (a) The procedural level is interrupted and

    (b) Passed a concise description of the error. (Note that this exactly parallels the interaction of the arm controller as a whole with the rest of the mini-robot system.)

THE DASV AND CONTROL STRATEGIES

As mentioned above, the dynamic level cannot control all of the variables in the ASV at once. In recognition of this, the format of the desired ASV, DASV(T), is designed so that the procedural level can indicate which components of the ASV it is really interested in controlling. The DASV has the same components as the ASV, but each one can be either:

(a) A specific number, which indicates that this component is to be explicitly controlled.

(b) A list (NOMINALVALUE, RANGE), which indicates that an error should be signaled if the component gets outside of the interval (NOMINALVALUE +- RANGE).

(c) NIL, this indicates that the procedural level is not concerned with the value of this component.

Based on the form of the DASV, the dynamic level uses one of the following basic strategies to compute G.

(a) If the DASV indicates that position and/or velocity is to be controlled, then the equations of motion for the arm (actually an approximation to them) are used to compute the ACV.

(b) If the DASV indicates that force is to be controlled, then direct feedback is used to control it. How the motor torques effect the forces sensed must also be calculated. This depends on the arm position and the point of contact.

(c) If less than a 6 dimensional set of components is specified for control (in particular if none are specified), then the extra freedom of choice for ACV(T) is used to keep components of DASV(T) specified as intervals closer to their nominal values. An example of this is in pushing a block aside. The DASVs have no specific components. Before the hand touches the block, it moves at the nominal velocity. After contact, if the force rises above the nominal value, then the velocity will drop.

(d) The preceding was an example of a situation where control strategies a) and b) had to be mixed. This is done by applying both, and using the extra freedom of choice in each one to make the two solutions fit. For example if 4 positions were specified then, this would fix 4 joints. If in addition 2 forces were specified, then the 2 remaining joints would have to be used to create the forces. If this could not be done, the dynamic level would complain.

(e) Note that no attempt is made to control the touch sensations. Their state depends on ENV(T) which is unknown at this level. There is no way that the sensors can be made to turn on in a short time dT. The dynamic level just reports their value, and can be set to cause an interrupt when they change state.

(f) At the opposite extreme, the dynamic level regulates the state of the joint lockers and hand. Thus an error in these is never detected.

THE BASIC ALGORITHM OF THE DYNAMIC LEVEL

The dynamic level is activated at a discrete set of times Ti, in order to produce a step function approximation ACV(Ti) to the actual ACV(T) which would produce the desired action. The Ti are specified in the time component of the DASVs. The closer together they are, the better the approximation is. How good an approximation is needed depends on the precision of the action.

At each time interval the dynamic level executes the following algorithm:

(1) Obtain ASV(Ti) from the arm sensors.

(2) Compute DELTA(Ti) from ASV(Ti) and DASV(Ti) as follows: for each element of DASV(Ti) one of three cases applies
   (a) component absent, DELTA(Ti) = ASV(Ti)
   (b) component an interval, DELTA(Ti) = ASV(Ti) - (NOMINALVALUE(DASV(Ti))). In addition, if ASV(Ti) is outside of the allowed interval then interrupt the procedural level and inform it of the error.
   (c) Component a number. DELTA(Ti) = ASV(Ti) - DASV(Ti) (note that the probability is near zero that DELTA(Ti)=0, even though this is the goal. However, the dynamic level takes no action here no matter what DELTA(Ti) is. Rather it waits until step (4). That is, no matter how bad it did last interval, it will go on if it thinks that it can get to where it should be at the next time interval. In any case, it can use DELTA(Ti) to improve its approximation to G.

(3) Get DASV(Ti+1). The procedural level communicates with the dynamic level through a queue of DASVs. If this queue is empty, the dynamic level interrupts the procedural level and requests a DASV. The arm must be continually controlled. Until the dynamic level gets a DASV it will continue the current ACV with possibly bad results.

   How close the coupling between the two levels is depends on the length of the queue. The procedural level can take complete control by supplying the DASVs one at a time. Alternatively the procedural level can leave the dynamic level alone indefinitely by putting a loop in the queue.

(4) Calculate ACV(Ti) and apply it to the arm. To do this, the dynamic level must first check to see that DASV(Ti+1) is achievable. If not, it interrupts the procedural level and requests a new DASV(Ti) which is achievable. Also note that some subcomputations (such as evaluation of the equations of motion) are very time consuming and will have to be spread over several time intervals.

(5) Wait until Ti+1 and then go to (1). It is very important that the dynamic level begin its computations at time Ti+1; it cannot wait for anything. Also the entire computation must be short compared with the length of the time interval, or the control will deteriorate. To this end, it is clear that the arm system as a whole will have to run as a set of asynchronous tasks sharing the CPU time.

THE PROCEDURAL LEVEL OF THE ARM CONTROL SYSTEM

Here is an input/output description of the procedural level:

(A) Output to the dynamic level
  (1) DASV(Ti) the desired ASV (communicated through a queue)

(B) Inputs from the dynamic level
  (1) ASV(Ti) the current state of the arm
  (2) DELTA(Ti) the summary of differences between ASV(Ti) and DASV(Ti)
  (3) Interrupts signaling errors and key information detailed in DELTA(Ti)

(C) Inputs from the high level robot system
  (1) High level commands in the form of LISP programs
  (2) Interrupts signaling error conditions or key information detected by other elements of the mini-robot system (for example the vision system)

(D) Outputs to the high level robot system
  (1) Signals indicating the completion of an action
  (2) ASV(Ti) (in some high level coordinates) if requested
  (3) Interrupts signaling error situations
  (4) A description of any error

The procedural level is a LISP system with a large body of built-in functions. These functions can be divided into three classes:

(a) The first class consists of the basic LISP functions (PROG, LAMBDA, COND, etc.) plus a set of functions to implement the multiple asynchronous control paths necessitated by the real-time demands on the system. These include functions for synchronizing communication between, creating, and deleting control paths.

In addition there are functions for creating and fielding interrupts.

(b) The second group of functions embody mathematical information about the arm and its capabilities. These include functions for transforming between coordinate systems, planning trajectories, etc. The dynamic level can be thought of as one of these programs. These programs are implemented in machine code for efficiency.

(c) The last class is the most interesting. It is the set of procedures for performing actions. These programs are kept as lists, and are available for inspection by the higher levels. The commands sent by the high level system are either direct calls on these procedures, or more complex procedures built out of the functions already in the system. Procedures can be added, deleted or altered by the higher level system in order to suit

its needs. Some examples of basic procedures are:

(1) MOVETO B.

(2) HALT arm where it is.

(3) APPLYFORCE F to point A.

(4) MOVEINDIRECTION R from point A until contact is made with an object, then halt.

(5) GRASP the object between the hand's fingers (some routine called by this routine must calculate the change in the inertial description of the arm when it picks up the object).

Some examples of more complex procedures that might be in the system if it were working in the environment of circuit board construction are:

(1) INSERTRESISTOR at A in holes at B.

(2) STARTNUT at A on bolt at B.

(3) SOLDERLEAD in hole at A.

(4) ATTACHTESTPROBE to lead at A.

Any action can be encoded as a procedure as long as it can proceed by dead reckoning without feedback from the rest of the system. Whether procedures like 1-4 above will be done with or without visual feedback depends on the taste of the users, and the speed of visual processing.

HANDLING ERROR SITUATIONS

It is clear how everything works as long as no errors arise. However, it is the error situations that are the most interesting.

Information about errors is brought to the attention of procedures able to deal with them through interrupts issued by other procedures which have gotten into trouble. The interrupt facility is basically a simple pattern-directed invocation facility with two key differences.

(a) It is assumed that only one procedure will be appropriate at a time. This assumption does not seem unreasonable in light of the simple kinds of errors with which the arm control system is trying to deal.

(b) Control is almost never returned to the routine issuing the interrupt. The issuer is usually supplanted by an updated approach to the problem facing the arm control system.

The interrupt mechanism is implemented by two functions:

(a) (SIGNAL INT ADDEDINFORMATION) Where INT is an atom (the pattern for

pattern-directed invocation), and ADDEDINFORMATION is anything that the signaler feels will be useful to the invoked procedure.

(b) (ON INT PRED ACTION) When an ON is executed, it enters the triple (INT PRED ACTION) on a stack. The triple is removed from the stack when the PROG the ON was executed in is exited.

When an interrupt is signaled, the stack is searched for an appropriate triple. A triple is appropriate if INT is the same as the one signaled, and PRED is true. When an appropriate triple is found, ACTION is evaluated in the environment of the PROG in which the ON was executed.

Here is a simple example of an action procedure using these ideas.

```
(DEFUN MOVE_TO  (B)
       (PROG ()
             (ON COMPLETION T (RETURN))
             (ON UNABLE_TO_CONTINUE_MOTION T (GO RETRY))
       RETRY (SETQ DASVQ (MAKE_TRAJECTORY B))
             (SUSPEND) ))
```

In the above, DASVQ is the queue of DASVs used to communicate with the dynamic level. COMPLETION is the interrupt that is generated by the additional element of the last DASV in the queue. UNABLETOCONTINUEMOTION is the interrupt signaled by step (4) of the algorithm running in the dynamic level. MAKETRAJECTORY is a mathematical routine that plans a trajectory from the present location to the point B. The returned queue of DASVs is terminated by one set to signal COMPLETION, and one that points to itself to keep the arm stationary and waiting at point B. SUSPEND is a function that stops execution in a control path. Execution can be resumed by the action of another control path.

The structure of this program is typical. A procedure usually sets up a series of alternatives for future action, computes some number of DASVs, and then suspends itself. It will later be re-excited by an interrupt from the dynamic level.

FLATEAU'S MINIATURIZATION STUDY

One of the laboratory's goals is to understand how very small manipulators can be used in research and in applications. In this connection, Flateau has studied the problems of small sizes.

DESIGN CONSTRAINTS

Before one starts examinations of the various technologies that might be applicable to very small robot arms, it seems very useful to summarize in as general terms as possible the design constraints that seem reasonable at this time.

SIZE

One goal is the creation of a desk-top A.I. robot arm system whose primary aim is to allow A.I. experimental work to go on in a medium size office and thereby allow a large number of research organizations to acquire the basic equipment. This requirement alone can probably be satisfied by a fairly broad optimization. An arm scaled by 2/3 might be small enough while a 1/4 sized arm would not be too small.

ASPECT RATIOS

At least two aspect ratios must be distinguished. The first is the arm aspect ratio and relates to the arm length versus diameter ratio. This would be of interest if the resulting mini-robot arm resembled existing larger manipulator or robot arms. Aside from intuitive esthetic arguments, the aspect ratio of that portion of the arm near the terminal device must be kept within reasonable bounds in order to reduce obscuration of vision for the imaging system in use. Also access to recessed volumes becomes restricted if the arm aspect ratio is allowed to become too small.

The second aspect ratio relates to the ratio of the cube root of the accessible working volume to manipulator or robot arm length. The traditional motivation in making this ratio large stems from nuclear hot cell practice, where it is desirable to have remote access to volumes kept small by shielding cost considerations. Except in special cases, like surgical manipulators, no motivation seems to exist to demand extreme volume aspect ratio for mini-robots.

POSITION ACCURACY

In a robot arm, accuracy refers to the tolerance with which the arm may be placed using position transducers information only. Present programming practices, particularly in industrial robotry, use position information only for the placement of objects by the robot arm. In industrial robotry, better placement accuracy than that is achieved by use of appropriate fixturing, a fact that contributes to the slow adoption of robots in industry.

Critical examination, however, of the human arm anatomy fails to detect anything which could possibly be analogous to a position transducer. Position accuracy seems to be achieved solely by successive approximations in the arm-eye-force sensing systems. Complete adoption of such programming schemes would eventually require only the most rudimentary accuracy capabilities. But for the moment arms are required to accept absolute positional programming, and reasonable accuracy should be provided. Also if multi-degree of freedom force transducers are used, transformations from transducer coordinates to actuator coordinates are required. The accuracy of these transformations depends on the precision of position information available.

A reasonable accuracy number to strive for seems to be 1/1000 of the largest available arm excusion, taking into account all the effects influencing final positional accuracy.

POSITIONAL RESOLUTION AND REPEATABILITY

In order to take advantage of more advanced programming techniques, a position resolution at least 10 times better than the absolute accuracy should be provided.

In setting the accuracy limitation, capabilities of transducers with reasonable cost limitation have been tacitly introduced. Peculiarities of some of these allow one to obtain extremely high resolution with ease. The resolutional limits are then given by the resolution of the ADC used.

We also must be aware that actuator smoothness must allow position changes within resolution limits.

Repeatability will automatically be better than absolute accuracy as it is not limited by linearity effects. But various thermal effects usually limit repeatability to less than resolution limits. Repeatability, of course, becomes the accuracy obtainable if one is willing to store a linearity correction map. It is hard to give a general number for this, but 3 to 5 times better than accuracy seems to be feasible.

FORCE ACCURACY AND RESOLUTION

Force information has quite a different character than position information. In principle, position may be referred to any convenient origin of the coordinate system and required accuracy is independent of the position with respect to the chosen origin. Yet in force measurements we must recognize the existence of an absolute zero state and required accuracy is signficantly related to the magnitude of the non-zero force that is measured. Of particular significance is the resolution with which the zero point can be established. The latter is usually limited by hysteresis and thermal drift effects. It is unfortunate that force transducer manufacturers lump hysteresis and thermal drift effects together with linearity and quote these as percentage of full scale. It is not easy to establish sufficient requirements as they are intimately connected to programming techniques that will be employed.

COMPLIANCE

Deflection under full load should ideally be limited to the amount allowed for accuracy. To make this statement meaningful one must define how this is to be measured. For instance, if we are concerned about the difference in position as indicated by transducer readings and real terminal device position we can get these to agree within the limits of accuracy by just making the arm structure strong enough and placing the position transducer close to the terminal device. In this way one can remove actuator and motion transmission stiffness from the specifications. While this probably constrains structural stiffness sufficiently to allow working within desired accuracy limits, it is not a practical approach for small arms, primarily because of position transducer sizes.

The stiffness due to motion transmission links, which may or may not be different from structural members, and that due to transmission link deflection must also be sufficient to place natural frequencies outside the response regions of the arm. This is a requirement for which each proposed design must be examined.

A PROPOSED DESIGN

Given these design constraints it seems that a tendon driven arm with force feedback could be a good first step into small size robotics.

This section sumarizes Flateau's description of such an arm. The proposed arm

design is illustrated in figure 1. It accomplishes the following:

1. It is a sufficiently small arm size (8" total length) to make the project interesting, achieved through use of tendon technology.

2. It is a completely self-counterbalanced design allowing the use of full motor torque resolution for all degrees of freedom.

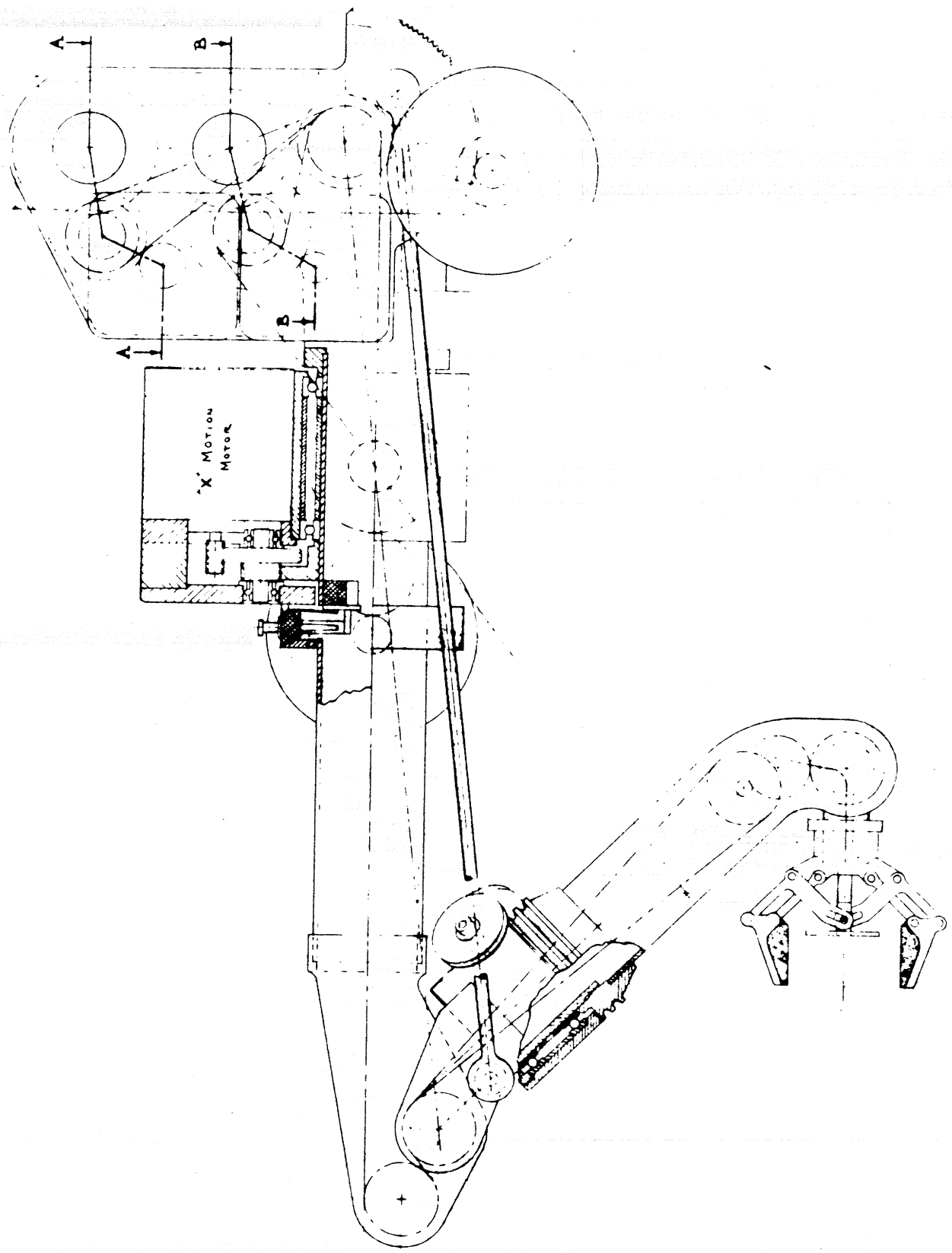
3. It has an exact tendon joint articulation compensation system, which is coordinated with the counterbalance system.

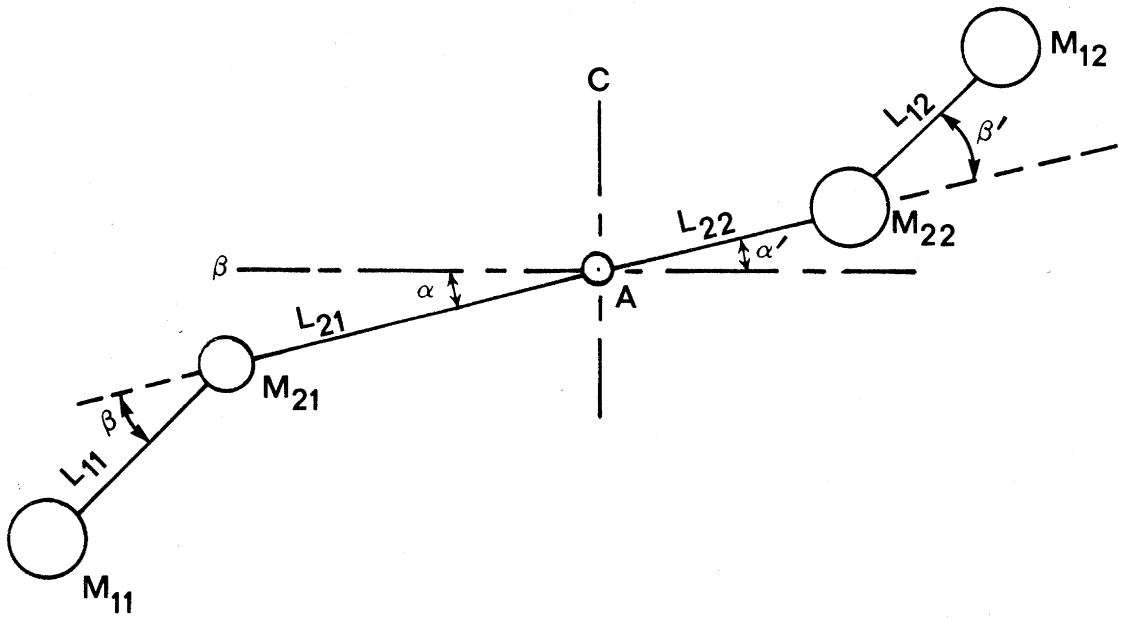
4. It has exceptionally low compliance in both the structural and motion transfer systems. Error due to worst case full load deflection is of the order of position accuracy due to transducer linearities.

The proposed design is possibly not as small a scale as this technology would allow. It was chosen as a reasonable compromise between the desire to achieve a significant scale reduction and the need to verify the scaling rules and problem approach.

SELECTION OF KINEMATIC ARRANGEMENT

To show that a counterbalance arrangement is in principle exact for all possible motion of the upper and lower arms we first confine ourselves to the planar argument with equivalent point masses representing the arm masses involved.


A
B
B
A
'X' MOTION MOTOR



To obtain balance in a gravitational field one must satisfy the following relationships where $\alpha = \alpha'$ and $\beta = \beta'$ are design constraints.

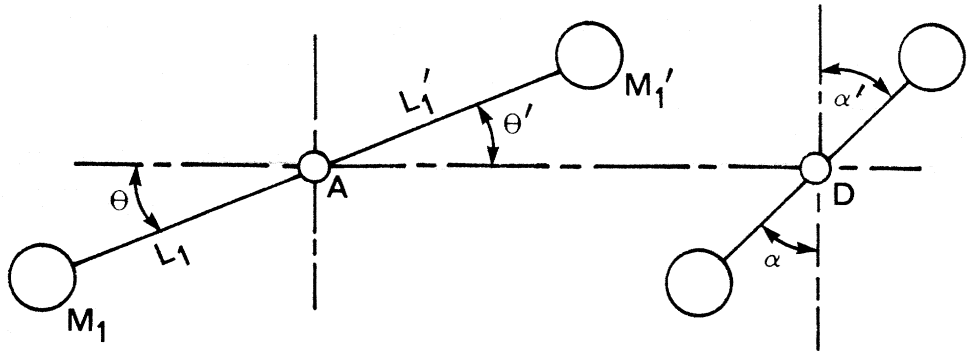
$$m_{12}(L_{12} \cos \beta' + L_{22} \cos \alpha') + m_{22}(L_{22} \cos \alpha) =$$
$$m_{11}(L_{11} \cos \beta + L_{21} \cos \alpha) + m_{21}(L_{21} \cos \alpha)$$
$$m_{12}(L_{12} \cos \beta') = m_{11}(L_{11} \cos \beta)$$

from the above for $\alpha = \alpha'$ and $\beta = \beta'$ we get for all α and β:

$$m_{12}L_{12} = m_{11}L_{11}$$
$$(m_{12} + m_{22})\, L_{22} = (m_{11} + m_{21})\, L_{21}$$

as requirements for planar balance. It is obvious that these requirements are also satisfied for rotation around the "c" axis.

For examination of the third possible rotation around the "D" axis one replaces the planar system with its equivalent:



where

$$m_1L_1 = m_{11}(L_{11} + L_{21}) + m_{21}(L_{21})$$
$$m'_1L'_1 = m_{12}(L_{12} + L_{22}) + m_{22}(L_{22})$$

we then have:

$$m_1L_1 \sin\theta \sin\gamma = m'_1L'_1 \sin\theta' \sin\gamma'$$

and since $m'_1L'_1 = m_1L$ and $\theta = \theta'$ by transformation of c, we get as remaining requirement $\gamma = \gamma'$.

In the actual mechanism $\gamma = \gamma'$, and $\alpha = \alpha'$ is satisfied by the one piece upper arm. A connecting rod was chosen to maintain $\beta = \beta'$.

TENDON COMPENSATION

Having already introduced the constraint of $\beta = \beta'$ to obtain counterbalancing, a satisfactory solution for the tendon compensation of the elbow exists which utilizes this arrangement.

The basic problem of tendon compensation consists in keeping tendon length between actuator pulley and motion output pulley unchanged due to intermediate joint motion. As guide pulleys must exist at each intermediate joint to keep tendons under tension, the problem is reduced to compensation for change in length due to change in the angle of wrap of the guide pulley. In our case, this is easily accomplished by using equal size pulleys at the elbow joint and counterbalance joint and wrapping the tendon in opposite directions. This keeps the sum of the wrap angles a constant and therefore tendon length is unaffected.

An additional pulley or two per cable path are required to accomplish the reverse wrap at the counterbalance joint. These pulleys can also be utilized to pre-tension tendons to the required degree.

Once these problems have been satisfactorily solved, the wrist drive motors can be conveniently placed in the counterbalance mass, thus eliminating addition of extraneous weight for this purpose.

ARRANGEMENT OF DEGREES OF FREEDOM

Figure 1 shows an elbow convex configuration. This is made necessary by the choice of wrist arrangement. The wrist kinematic arrangement is identical to that used in virtually all existing BFR manipulators. It is probably the simplest arrangement possible for obtaining wrist action from tendon motion. It involves a simple differential action for the pitch and roll motions. When both differential input pulleys rotate in the same direction at the same speed, the differential adds these motions and pure pitch results, or an (A + B) operation. Similarly, pure roll results from a (A - B) operation. Combined motion can be obtained from linear combinations of these two. These simple operation should not involve programming difficulties.

As a result of yaw motion the tendons for pitch and roll must twist in the tube. Surprisingly this can be done with impunity and with very little position error in pitch (0.0007" for the worst case). This small error is achieved by having the effect subtract in the differential at a sacrifice of about 0.006" error in roll.

The disadvantage of the above arrangement is that a motion singularity exists when the roll axis is parallel to yaw axis. As simple priority logic can resolve the resulting redundancy, this means that for position near the singularity the arm has effectively only five degrees of freedom.

To avoid this situation, pitch motion is restricted to angles larger than 15° above the singularity position, as the assumption is that the preferred working position for the

terminal device is parallel to the main attitude of the arm.

An elbow concave attitude either requires another more elaborate wrist arrangement or a "terminal device down" peferred position.

Linear motions and other articulated motion arrangement have not been discussed here as new counterbalance and tendon compensation geometrics must be developed for them.

POSITION ACCURACY AND POSITION TRANSDUCERS

Even the smallest available transducers are quite large for this arm. Small transducers are, however, not available in the highest linearity ratings. Therefore the following compromise has been arrived at for this arm:

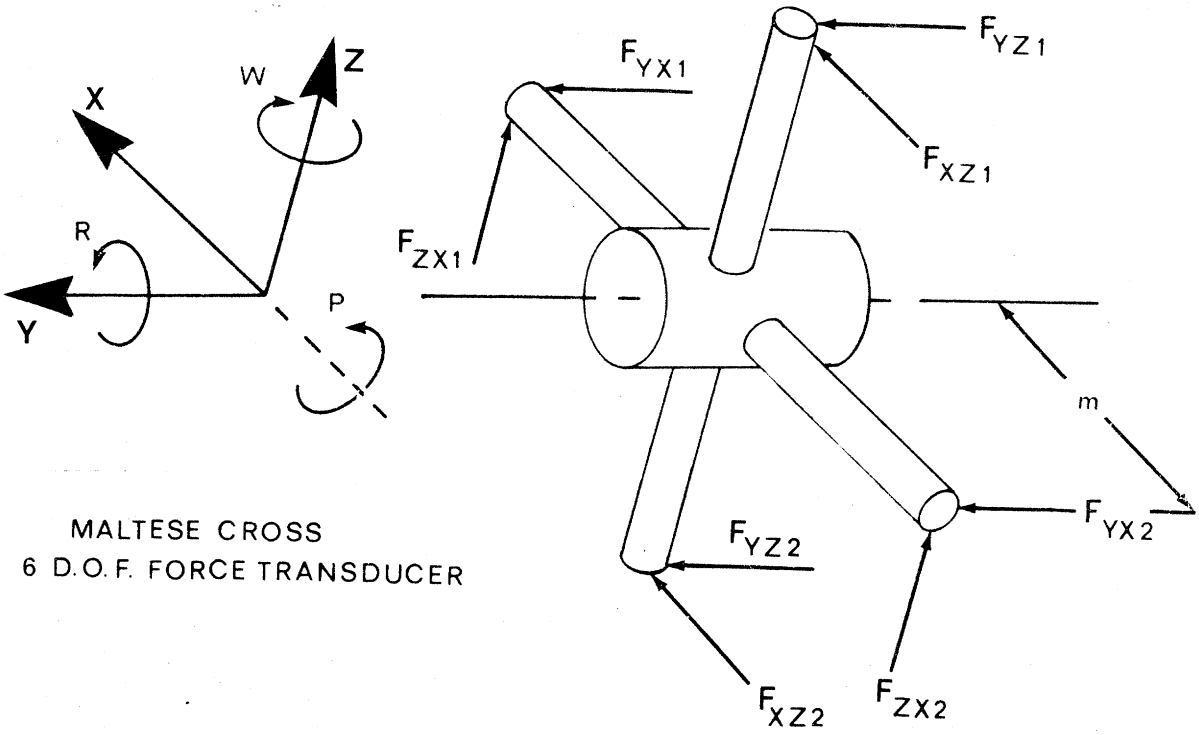
For x, y, and z motions 1-7/16 diameter conducting plastic potentiometers could be used either geared or as elements as shown for the x motion. These should be available with 0.05% linearity. In the counterweight mass no room for such large transducers can be made available. Consequently 1/2" diameter potentiometers which recently have become available in 0.15% linearity would be used.

The larger position error due to this would affect only the wrist motions, which because of the smaller moment arms, does not reduce overall accuracy that drastically. The derivation of the complete error function for the entire arm is beyond this study. However, a quite accurate approximation method for this familiar geometry yields a very worst case accuracy error of 0.015". By very worst case we mean that all possible error sources combine in the worst possible way, something which is statistically rather unlikely.

If higher accuracy is desired one would have to resort to mapping the transducer nonlinearities and storing correction tables.

FORCE TRANSDUCERS

No design proposal for the force transducer interface is offered at this point. The volume assigned to the transducer is sufficient to introduce a 6 degree-of-freedom transducer of the Maltese Cross configuration which is well known.



MALTESE CROSS
6 D.O.F. FORCE TRANSDUCER

But the Maltese Cross configuration is probably far from optimal. Hopefully with more study an improved configuration can be found. Contacts with commercial force transducer specialists have been established to make existing expertise and technology available to the solution of this problem.

ACTUATORS

Much of the dynamic advantages of the tendon design can be undone if one chooses motors which do not have sufficiently low inertia to allow a truly bilateral behavior of the robot arm. To that end an Inland "inch square" motor model NT-0716 has been chosed. This choice results in a load inertia at 1 g acceleration referred to the arm loading point of nearly 20% capacity. Over 3/4 of this is due to motor inertia. While this is excellent compared to what is possible with joint proximate actuator techniques, it is just barely tolerable for the proposed operating conditions. Since our scaling laws call for L scaling of acceleration, 1 g acceleration would correspond to an excessively high acceleration of 3 g of a unity scaled arm, and such high accelerations are not

necessary. This result was to be expected from the scaling laws. Another contributor to a relatively high no load inertia is the 6 degree-of-freedom force transducers placed very near the output end of the arm.

The inertial contribution due to the motor could be reduced by a factor of four if use is made of full motor torque capability and gear ratio adjusted accordingly. This, however, would reduce available motor duty cycle at full load from 100% to 25%, which is not really a viable approach as the thermal time constant of the above motor is only about 15 seconds. Addition of a brake would just add considerable bulk to arm and mask the problem, but not solve it.

Another consequence of motor scaling laws results in friction torque contribution of about 10% of full torque capability. That means that for loads up to 10% of maximum bilateral behavior can only be obtained by closing a loop from the force sensor through the computer control system.

CHOICE OF TENDONS

Miniature wire rope was chosen for the tendons, primarily because a usable size, in very flexible construction, has very recently been made available. The wire rope allows rather lower complicance than a tape design with similar arm aspect ratio. The high degree of stiffness has been achieved by using a fairly large gear ratio between the differential and the lower tendon pulleys. This requires multiple turns at the tendon input and output pulleys which would lead to serious position error with a tape system. With the wire rope, deflection due to tendons of the terminal device in pitch at full load is about 0.002 in. The disadvantage of wire rope tendons consists of somewhat higher friction. In this scale this effect turns out to be small compared to the friction contribution of the motor.

MINSKY'S DESIGN VIGNETTES

In this section we review some of Minsky's thoughts on arm and wrist design.

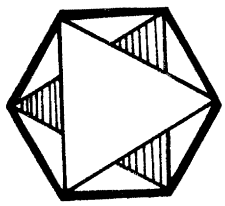
PARALLEL MANIPULATORS

The parallel approach to manipulator design has different problems and features from serial approaches. To introduce it, consider the general problem of positioning a rigid body in space relative to a frame of reference.

Suppose there are three fixed reference points in space and three points on the rigid body. Then there are nine distance-pairs between these points, but specifying almost any six of these causes the other three to be determined. This observation suggests a very simple and direct approach to design. Choose six of these distances. Then construct a manipulator by realizing physically each length constraint as some mechanical length-determining device, such as a hydraulic cylinder or other "linear actuator."

Each actuator must be terminated by a spherical joint or elastic coupling. Then we achieve complete mechanical control of the rigid object simply by instructing separately the motors for all degrees of freedom, without cascading their effects.

A particularly symmetrical example of such a design is shown below: we choose six edges of a cube to position one equilateral triangle with respect to another. The resulting structure outlines an octahedron:



The six edges are the ones that form the hexagonal shadow when a cube is balanced on one of its vertices!

In principle, it follows from the basic constraint-determination idea, that one can position the body in any location and orientation in three-dimensional space. However,

there are practical problems:

In certain positions the constraints degenerate (e.g., two rods coincide) making the position indeterminate. The structure will flop.

As we approach a degenerate situation, the mechanical advantages become too small, great forces are needed, and the structure will buckle.

It is hard to find large ranges in which the supports do not interfere with one another by colliding. This is particularly severe in regard to axial rotations.

The supports must subtend a substantial space angle to get much stiffness. On the other hand, an arm should subtend a small space angle (looking from the hand) to achieve dexterity.

The ability to get around obstacles is limited.

Although parallel systems have disadvantages, these seem quite different from those of the serial system. There are important advantages, as well.

Very great simplicity, because of the non-interactions.

The instrumentation of force-sensing is particularly convenient; the longitudinal stresses in the supports determine completely the forces operating on the mobile plate (unfortunately including gravity). As shown in below, one can exploit this to get very elegant and useful kinds of force and tactile feedback.

The forces are entirely axial, so that light, thin tubes can yield great strength. Serial arms in general, and angular actuators in particular, have bad weight-strength characteristics.

FORCE FEEDBACK

Consider a rigid bar supported by two force sensors A and B:

If we press at point "A" with unit force we will read

$$A = 1 \quad B = 0$$

If we press with unit force at "B" we will read

$$A = 0 \quad B = 1$$

If we press at "C" we will read

$A + B = 1$ (total downward force) and
$2A + B = 0$ (zero net torque),
hence

$A = -1$
$B = 2.$

More generally, if "X" is the distance from A to C, and "F" is the force, we will have:

$$A + B = F$$
$$AX + (B-1)X = 0 \quad \text{or} \quad FX-B = 0 \quad \text{(torque cancellation)}$$

so that we get

$$X = B/(A+B)$$

showing that we can read from the two dials, almost directly, both the location of the force and its magnitude! By a similar argument we will see that if a triangular plate is supported by three scales, we can again locate the point of application of an arbitrary force and its magnitude, even though it can now be anywhere in a plane.

Can we do better? A fixed body is subject to six constraints, so we can make three more measurements. We do this later and show that with six such readings, one can determine the force vector up to lying on a definite line in space.

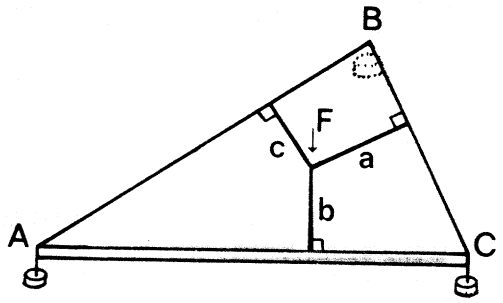
Before doing the calculation, we point out an important consequence. If we can determine the line of application of a force then we can determine the point of application of the force -- provided only that we know the shape of the surface! There is no reason to restrict our surface to be a particular plane; given any convex (and most any non-convex) shape, a space line enters the surface at a unique point. So the six-

axis force measurement can serve as a tactile-position sensor over the entire surface!

EXACT CALCULATION OF THE OCTAHEDRAL STRAIN-GUAGE

We now show how to resolve the force on the "octahedral" arm by a sequence of easy superposition steps. This derivation illustrates the immense power of knowing a system is linear: it is just a sequence of superposition arguments.

First consider an equilateral triangular plate supported by three scales at A, B, and C. Let "a", "b", and "c" be the distances from an arbitrary force point to the corresponding opposite sides -- these are the so-called Barycentric Coordinates.

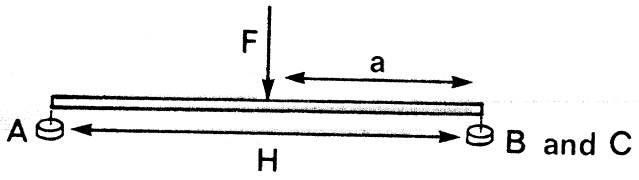


If F is the magnitude of the force, then F = A + B + C because the system is in vertical equilibrium. And the three weights are in simple proportion to the three distances a, b, and c. One can see this, for example, by viewing the scene in a plane orthogonal to the line BC:

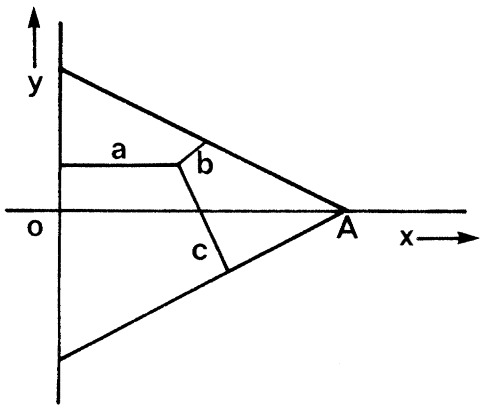
AH = Fa

BH = Fb

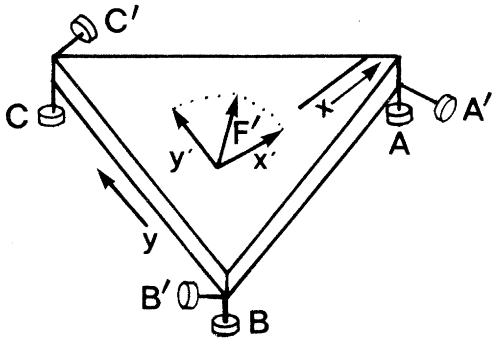
CH = Fc



where aF = AH, for example, simply because of cancellation of the two moments aF and AH. So it is easy to find the exact point of application of F, given these three measurements. It is convenient now to introduce some Cartesian Coordinates, and one can see using simple geometry that
$x = a$ and $y = \sqrt{3}/4\,(c - b)$.



A force on the plate may have tangential as well as normal (downward) components. We will detect these by adding three more strain gauges as shown here. Each senses differential motion along an axis parallel to the opposite side of the triangle.



We have shown a tangential force F' lying IN THE PLANE, with components x' and y'. No matter where such a force is applied, equilibrium dictates that

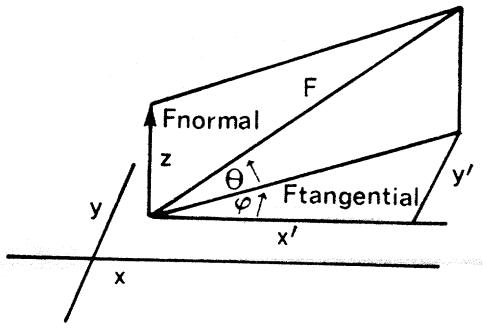
$x' = \sqrt{3}/2(C'-B')$ and $y' = C'/2 + B'/2 - A'$

So these three measurements give us the direction $\tan^{-1}(y'/x')$
and magnitude A' + B' + C' - A'B' - A'C' - B'C' of the tangential force component. If the actual force has both normal and tangential components we note that |A,B,C| still determines the point of application, since the three new gauges do not affect the normal measurements.

Thus from our knowledge of the tangential F' and the normal F, we get a complete three dimensional description of the total force F* in azimuth-elevation coordinates:

elevation = $\tan^{-1}(A+B+C)/|F'| = \theta$

azimuth = $\tan^{-1}(C'+B'+2A')/x' = \varphi$



showing that the actual computation is not particularly complicated.

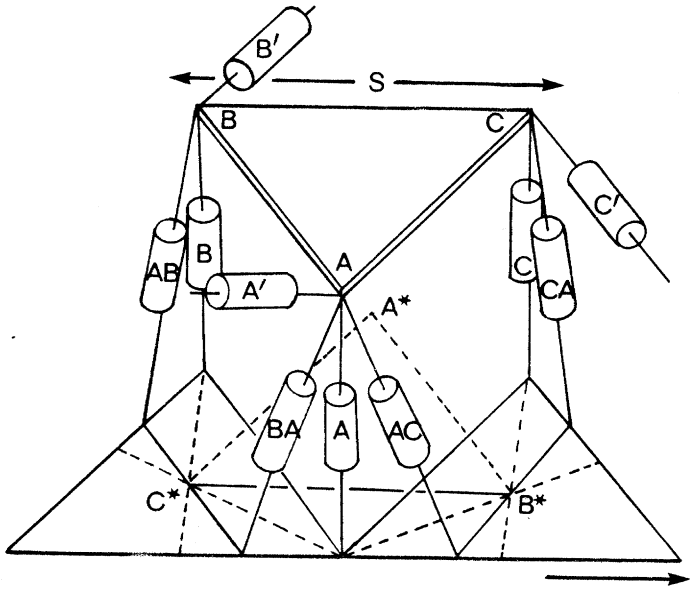
We derived only two quantities $[\theta\varphi]$ from the three measurements |A', B' , C'|. What is the third dimension? It is the tangential torque on the plate. To see this, assume that the strain-gauges' deflections are approximately linear in force, and consider the energy of distortion of the system regarded as supported by springs. Given a certain force, the energy in the tangential springs will be minimal when there is no torque, simply because an applied torque will always be resisted, hence will increase the system's energy, which is proportional to $A'^2 + B'^2 + C'^2$. For a pure y-force we have B' = C' and F' = C'- A'. In this case $E = A'^2 + 2(A' + F')^2$ which is a minimum when F' = -3A'/2.

It follows that 0 = A'+B'+C'.

A similar argument shows that a pure X-force also yields A'+B'+C'= 0. Hence, by superposition, since each term is a linear function of all force components, this sum is zero for any torque-free force. Furthermore, A'+B'+C' is proportional to torque applied at the center, is invariant of all translation forces, and is the only remaining linear function that is so invariant, so it must be the torque -- assuming correctly that the torque is a linear function of the measurements.

Now, finally, we return to the octahedral strain gauge. We will first replace our "normal" and "tangential" gauges A' and A by the oblique gauges BA and AC shown here. Note that now all four of A, A', BA, and AC lie in one plane! Therefore:

$A = pBA + qAC$
$A' = rBA + sAC$



by a simple planar change of coordinates. Now in our diagram, C* is the base point for attaching two gauges in our initial octahedral configuration. Again we can use a planar configuration, for by choosing the indicated dimensions for d and d/3 we have arranged that all of A, B, C*, AB, and BA lie in one plane. Hence, we can replace AB and BA by the correct pair BC* and C*A -- not shown in the diagram to avoid clutter -- by another planar transform:

$BA = p'BC* + q'C*A$
$AB = r'BC* + s'C*A$

which completes the entire analysis. The entire transformation can be condensed into a 6x6 matrix:

$A = p"BC* + q"C*A + r"AB* + s"B*C$

etc.

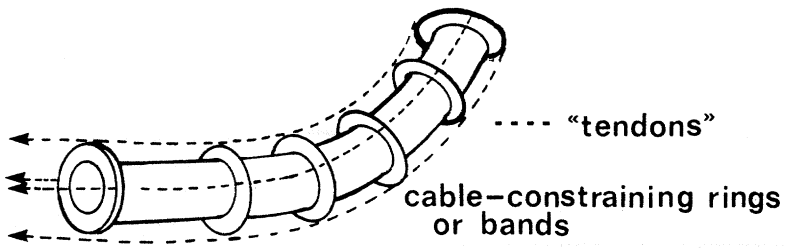
All this applies only to small distortions in the structure, as one would indeed have in a quasi-isometric strain-gauge. If one wants substantial motion, as one would in a bilateral force-servo in which the beams of the octahedral supports are also the manipulator's degrees of freedom, the coefficients would not be constant, and one would need trigonometrical approximations.

FLEXIBLE WRISTS

A wrist needs mobility, strength, and a protected instrumentation channel for its hand. To give the hand access to the work from different directions, a wrist should be slender -- the arm should subtend a small space angle as seen from the hand. Because the wrist is near the hand, large angular errors mean relatively small absolute errors, so absolute accuracy of angular control is not very critical. For the systems we envision, wrist control would be adequate even with increments of several angular degrees.

These requirements suggest the use of a deformable tubular structure. In very small scale systems the requirements of strength and rigidity virtually dictate using "exoskeletal" or tubular structures: central rods are not stiff enough and pin-joints are too weak. The tubular construction is also attractive for instrumentation, because the information channels are insulated by the exoskeleton, and the interior is a nearly constant-length environment.

An apparent solution to all these problems is to use a continuously flexible tube. This section is about some aspects of such a design. The simplest such wrist, no doubt, is a plastic tube or rod deflected by external tendons constrained to its exterior, as shown here. This has problems and advantages:



FLEXIBILITY AND HARMONIC ANALYSIS

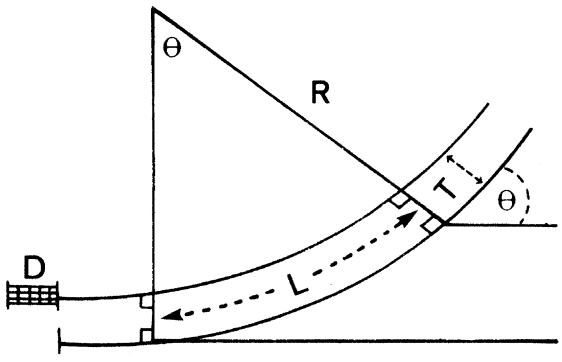
If we want to distribute angular deflections over appreciable path lengths, we have a theoretical problem about controlling the excess degrees of freedom. (To count degrees of freedom for a continuous rod, one has to use a signal theory method. If one describes the configuration in terms of spatial frequencies, there is a rapid attenuation of high frequencies.) In any case, we are interested in the case where there are more degrees than we are prepared to control explicitly.

The curvature in a beam tends to be constant -- other things being equal -- because the strain energy is a faster than linear function of local curvature. So the energy of the beam has its minimum for a given total curvature when it is uniformly distributed. Unfortunately, this quadratic minimum means that the resistance of the system to perturbations that transfer a small amount of curvature between two different segments is very small. Thus the uniformizing tendency is poorly coupled to the gross spatial configuration.

Now observe an interesting phenomenon that occurs in a flexible wrist with tendons constrained close to the surface of the tube.

To a very good approximation the net change in direction is independent of the tube's conformation.

The basic phenomenon stems from the fact that for small angles the $x = \sin x$ approximation is very good. Consider a flexible rod with opposed deflection tendons, and shorten the upper one a distance D. (Actually, the tendon on the convex side will be pulled in by about the same amount. In the analysis below, let D be the difference.)

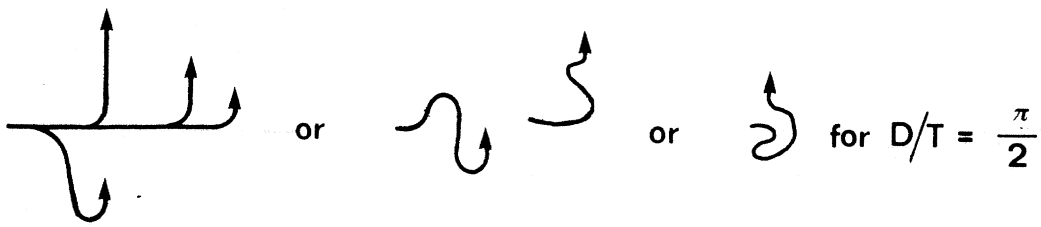


If the rod has thickness T and length L, and one side is D shorter than the other, the rod will bend in a curve for which

$$A(R+T) - AR = AT = D$$

Hence A = D/T, where A is the bending angle in radians. Note that this is independent of L.

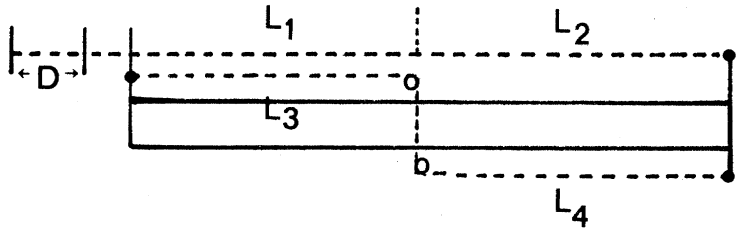
Now we assumed uniform curvature in this argument. But suppose that the rod is actually bent sharply in one part and gently in another. Then A = D1/T + D2/T = D/T again, so that for a given total difference in tendon length one gets the same net angular deflection of the tip independent of how the curvature is actually destributed along the rod! In other words, the same D is compatible with all configurations that begin and end with the same directions, such as in the following:



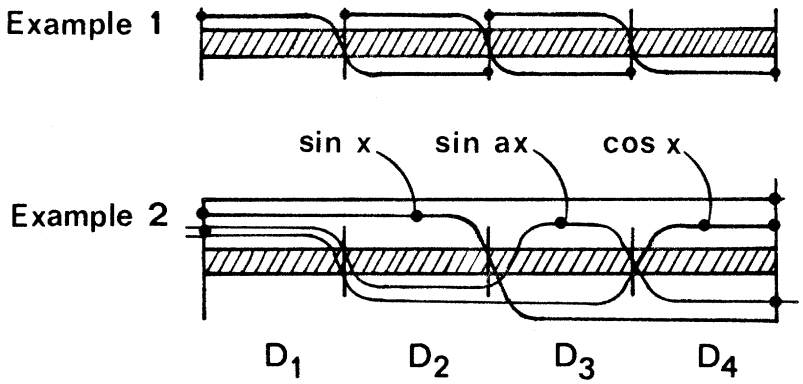
Now to enforce uniform curvature, we must distribute increments of D uniformly (in length) along our beam. Consider a first approximation in which D = D1 + D2 in two equal halves of L; then we want somehow to constrain D1 = D2. Solution: add an extra tendon firmly attached at both ends but crossing through the rod at the middle.

Then one can write:

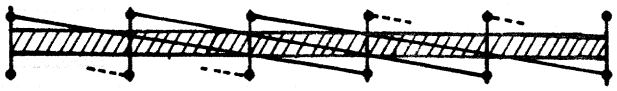
L1 = L3	identity
L3 + L4 = L	initial condition
L2 + L4 = L	good approximation



Hence L1 = L2. Thus the extra constraint divides the curvature equally between the two halves. One could add still other constraint-cords, crossing over at other points: the problem is to equalize the deflections in each segment. Two examples of how one might do this for more segments are:



Discussion of Example 1: To the extent that the oblique cords are inextensible, this scheme ought to be quite effective in uniformizing the deflection. It is like the well-known "lazy-tongs" device. It tends to have a cumulative error, in that the curvature can slowly drift from one end to the other if there are many segments. As drawn, there is also a problem about unconstrained parts of the rod near the cross-over points; this should be corrected by running each constraint over three segments, and this also relieves the sharp curves experienced by the stabilizing tendons.

Example 1′ 

Note that each wire needs a symmetrically opposite one for negative curvature control.

Discussion of Example 2: We have shown several constraint cords with suggestive labeling. Each cord runs the full length of the rod, so that the effects are global (no accumulated error). Let Di and -Di be the length changes at the top and bottom of the i-th segment. Then we have:

sin A:	D1 + D2 - D3 - D4 = 0
cos A:	D1 - D2 - D3 + D4 = 0
sin 2A:	D1 - D2 + D3 - D4 = 0

The first two equations yield	D1 - D3 = 0
The last two equations yield	D1 - D2 = 0
The first and last yield	D1 - D4 = 0

So all four segments have equal change: the total change D is controlled by the top straight fibre.

The choice of trigonometric labels was based on a strong analogy. We can use the "constraint" cables for control as well! If we shorten the "sin X" cable by an amount D* the equations still yield D1 = D2 and D3 = D4, and (assuming the straight fibre is unchanged) the rod assumes a sigmoid with the original tip direction preserved:

If we let C(x) be the curvature of the rod as a function of length, we now have

$$C(x) = D^* \sin(2x\pi / L)$$

Controlling the other fibres similarly, we can develop the curvature of the rod in a Fourier series:

$$C(x) = \sum_k D_{\sin kx} \sin \frac{2\pi kx}{L} + \sum_k D_{\cos kx} \cos \frac{2\pi kx}{L}$$

It would be impractical to do this for more than eight segments, and four would be quite enough for most purposes.

THREE DIMENSIONS

Note that by winding the tendons around the beam instead of through it, we avoid penetrating the central core. This same scheme can eliminate the occurence of "dead" intervals at which the tendons are near the axis and so provide little constraining force (poor mechanical advantage). Our scheme is to use Helical winding instead of Sinusoidal.

Using helical winding, one avoids the problem of tendons escaping from concave portions of the beam. It appears that a large number of fibres are required.

The four pairs of the 2-d case must each be duplicated in two planes. Our idea is to make two sets of windings, 90 degrees out of phase, in both directions around the cylindrical rod. Each must be duplicated again, in the opposite direction, to oppose the effects of axial torque. That makes 16 pairs of fibres. Since we really do have 8 degrees of freedom, the system certainly needs 8 tendon pairs. Possibly, half the windings could be eliminated if the tube has enough inherent axial stiffness, but this is not easily compatible with longitudinal flexibility. One way to get the required kind of stiffness is to have hinge joints in alternating orientation, as in the lobster arm.

These issues are not so serious. Even a two "segment" arm, in three dimensions, would already be a splendidly mobile device, and its helical geometry problem does not look very difficult. Going to four "segments" may raise more serious problems about friction and tendon-crossovers. Higher terms in the series get us involved with long cables (the "high frequency" sinusoids wrap around the rod a lot) and would probably cause severe frictional binding problems because the forces normal to the tube would become large. These forces don't look serious for the two and four segment designs.

FININ'S PRINTED CONDUCTOR FOLLOWER

The final two parts of this section are devoted to two preliminary studies involving electronic circuit board inspection and repair -- a general problem area with which we are debugging the mini-robot concept. Finin's contribution is a program for tracking printed conductors, and Lozano's is a program for finding resistors.

In both cases part of the purpose was to see if ideas worked out in the previous blocks world work are generally useful. The results demonstrate that complex heterarchical systems can now be built routinely.

Maximum efficiency was not a consideration and it is understood that simpler programs might very well work satisfactorily. Finin's task, for example, was to see how heterarchy might be used in a vision program for undestanding the layout of printed conductors on the back of circuit boards. For this reason he did not use the more obvious and perhaps computationally more efficient multiple pass idea in which "feature points" are first detected by local predicates and then later grouped together to form wires.

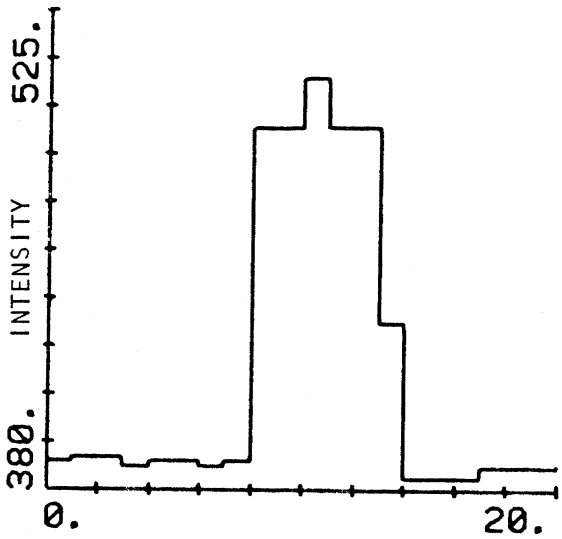
GOALS AND STRATEGIES

In this work, several applications were envisioned:

1. Find a particular pad and trace the wires leading from it to another pad.
2. Find a particular pad and inspect the soldering.
3. Find all the pads in a specified area.
4. Find all the pads on the board.
5. Develop a complete description of the board, including the location of pads, paths followed by wires, etc.

Overall strategy must depend on what the image of the back of a printed circuit board looks like under our input device. Several boards have been examined, and the programs tested on them. Figure 1 shows the result of scans across typical wires on the board. Note that there is good contrast between the dark insulating material and the bright plated wires.

Were contrast this good throughout the picture, the problem would be considerably simplified to that of dealing with a binary image. Contrast is significantly reduced, however, in certain cases. Figure 2 shows another scan across a wire. Here, the scan



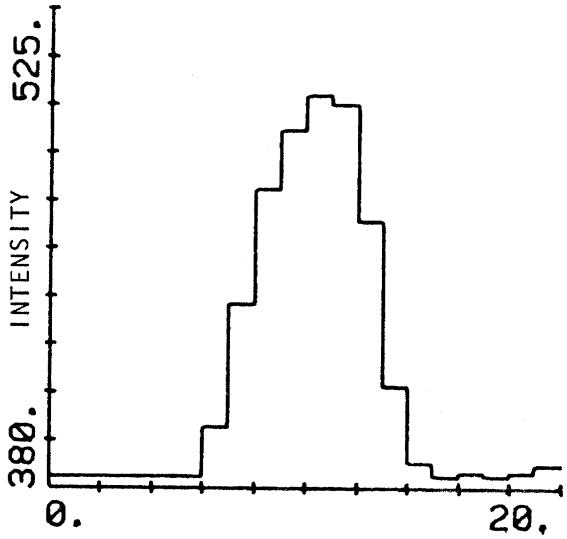


FIGURE 1

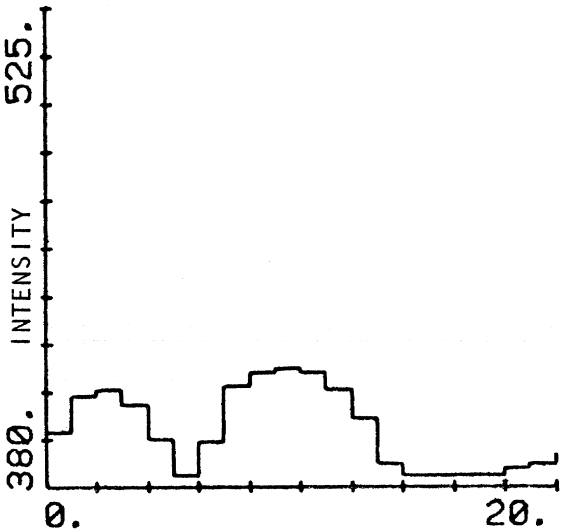


FIGURE 2

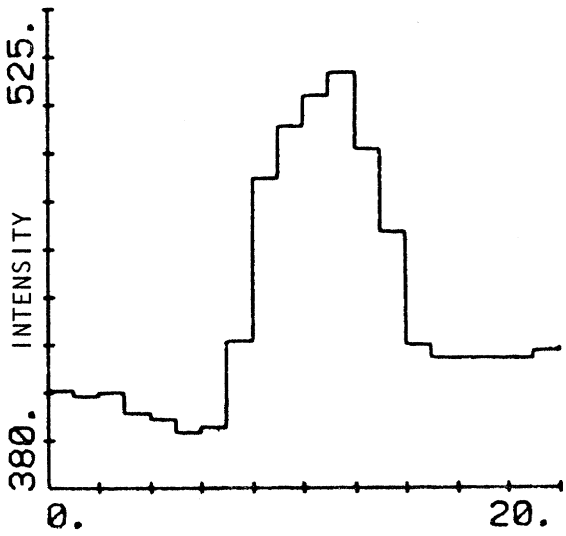


FIGURE 3

has fallen very close to a pad. The blob of solder on the pad casts a shadow which has greatly reduced the intensity. Additional variations in the contrast can occur when the board is constructed of a semi-translucent material. Figure 3 shows a scan where the intensity of the background insulator is raised by printing on the front of the board. Because of this, a tracking type approach was taken for this work.

Tracking is also the natural choice for the first of the previously mentioned goals, that of following a wire from one pad to another. Recognizing pads is also more easily done by a tracking procedure because of their non-local nature. A pad can be easily recognized as a "bulge" in a wire. Pads can thus be detected by monitoring the width of the wire as we track it. If the width suddenly increases, and then decreases, a pad is hypothesized. A tracking process will also allow us to make use of local context in other ways. For example, if we hypothesize a pad when tracking a wire, we can anticipate shadows and adjust our thresholds accordingly.

REPRESENTATION

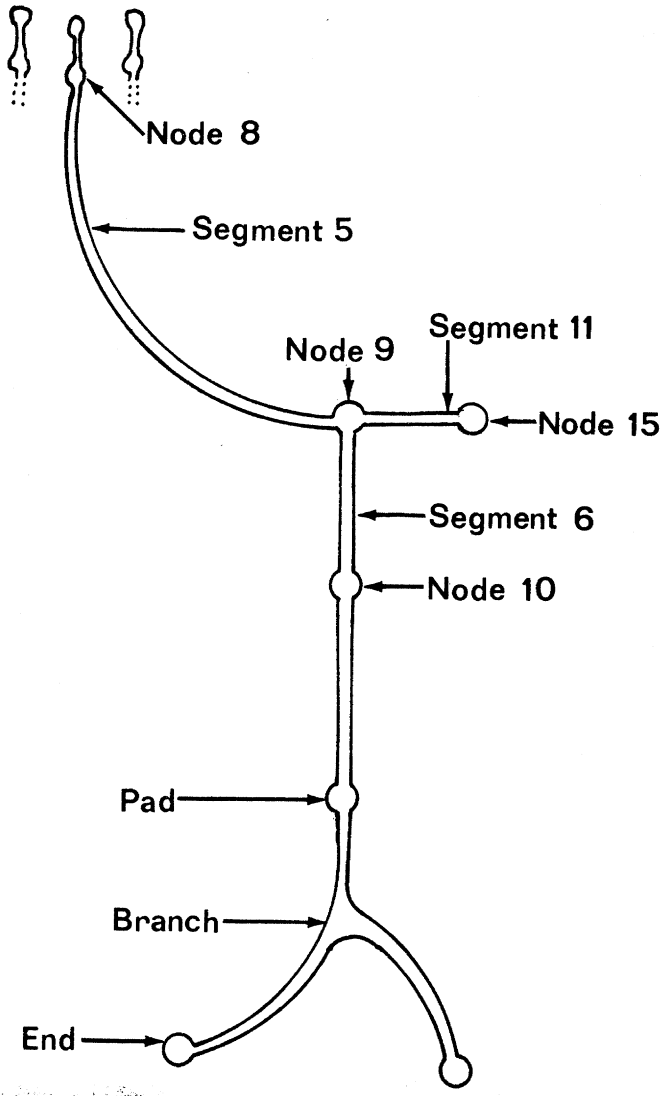
For the purposes of this work, the following conventions are used. A circuit board is made up of one or more CIRCUITs, each of which is comprised of a connected (and possibly branching) path of plated wire. NODEs divide a circuit into one or more SEGMENTs, which are sections of a circuit. A NODE is either :

1. The end of a branch of the circuit.
2. A place where the circuit branches.
3. A PAD, or place where a wire comes through from the front of the board and is soldered down.

Figure 4 shows a typical circuit and its parts.

The data base is made up of a network of information which is stored on property lists. The network character is achieved by the judicious use of "back pointers." For example, NODE9 in figure 4 would look like this:

```
                    NODE9
COORDINATES         :    (512 432)
CIRCUIT             :    circuit3
SEGMENTS            :    (segment5 segment6 segment11)
TYPE                :    branch
SUSPECTS            :    (51.3 178.5 260.7 355.4)
FAILED-SUSPECTS     :    (51.3)
HYPOTHESIZED-TYPE   :    pad
NEIGHBOR-NODES      :    (node8 node10 node11)
```



A Typical Circuit

FIGURE 4

THE PROGRAMS

This section contains a description of the general strategies used in examining the backs of circuit boards and a more detailed explanation of some of the algorithms. Figure 5 shows the major higher level functions and the flow of control between them.

The top level function, TRACE-CIRCUIT, is given the coordinates of a point on or very near a piece of conductor. It centers itself on the conducting wire and calls TRACE-FROM-NODE to begin tracking the circuit.

The action of the system revolves around two main routines: TRACE-FROM-NODE and TRACE-SEGMENT. Starting at a node, the function TRACE-FROM-NODE finds all segments leaving that node and applies TRACE-SEGMENT to each of them in turn. TRACE-SEGMENT tracks the segment until it ends, or until another node is hypothesized. It then applies TRACE-FROM-NODE to this point. Thus a circuit is explored in a sort of "depth first tree search" manner.

TRACING FROM A NODE

Finding and tracking all the segments leaving a node is done using a suspect-generation/verification technique. The basic procedure is outlined as follows:

1. Suspect segments leaving the node are proposed.
2. If any segments are known to end at this node, the suspects which correspond to them are deleted from the suspect-list. Unless the node is the initial one created by TRACE-CIRCUIT, there will be at least one such known subject.
3. A suspect is either verified or rejected.
4. If verified, the segment is passed to the function TRACE-SEGMENT for tracking.
5. Steps (2) and (3) are repeated until all the suspects found in (1) have been checked.
6. The node is then re-examined to determine its type.

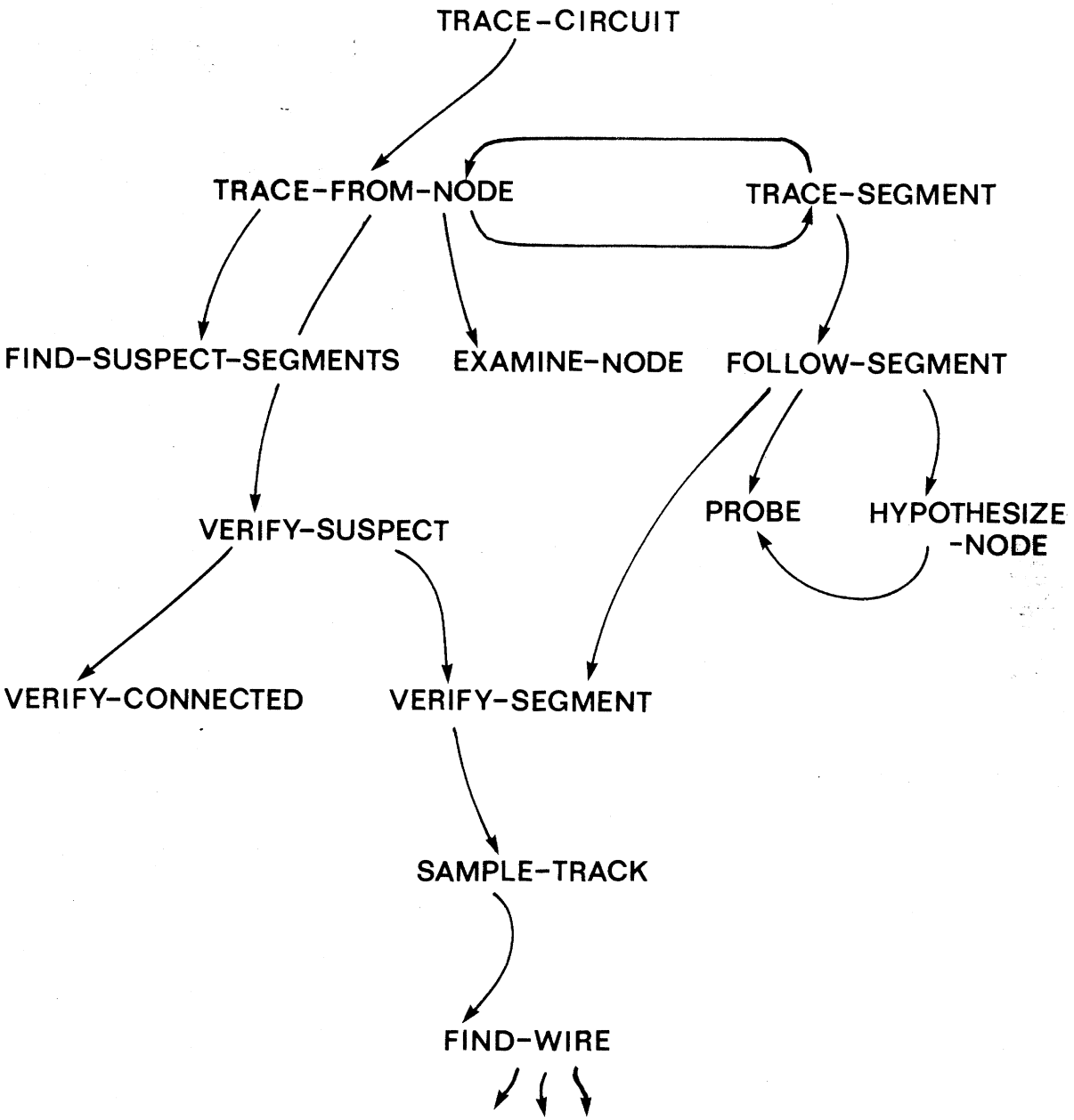

TRACE-CIRCUIT
TRACE-FROM-NODE
TRACE-SEGMENT
FIND-SUSPECT-SEGMENTS
EXAMINE-NODE
FOLLOW-SEGMENT
VERIFY-SUSPECT
PROBE
HYPOTHESIZE -NODE
VERIFY-CONNECTED
VERIFY-SEGMENT
SAMPLE-TRACK
FIND-WIRE

FIGURE 5

FINDING SUSPECT SEGMENTS

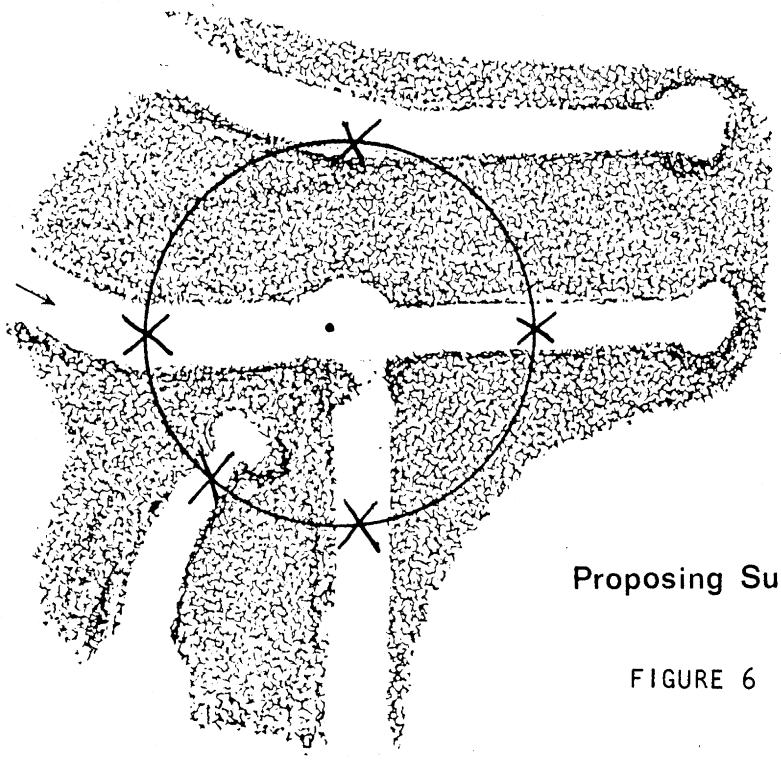
Circular search is the basic technique used for generating suspects. A circle of points around the node is scanned and examined to find suspect segments. The radius of this circle is determined by trading off several factors. Since nodes often appear as large blobs which cast shadows, we want to sample points far enough away to assure a good view of the segments leading into it. Similarly, a large radius will be more accurate if we start with a poor initial estimate of the node's center. On the other hand, a large circle will intersect unconnected neighboring circuits, resulting in many spurious suspects. A good value for this radius (FIND-SUSPECTS-RADIUS) was found to be about two to three times the width of a typical wire (see figure 6).

Depending on various conditions, one of two procedures is used in examining the points. If the picture is noisy, the contrast poor, or there is a good deal of low-frequency intensity drift across the scene, these points are filtered by a first differencing operation and averaged over a distance FIRST-DIFFERENCE-DELTA (about one half the typical wire width). Maximum and minimum peaks in these values are then found using an algorithm from Shirai (Shirai 1972). The peaks are then matched to form positive and negative pairs, which correspond to cross-sections of the wire. Criteria for pairing are:

1. The height of the two peaks must be within 75% of one another.
2. The separation between them must be between certain minimum and maximum values (MINIMUM-WIRE-WIDTH and MAXIMUM-WIRE-WIDTH).

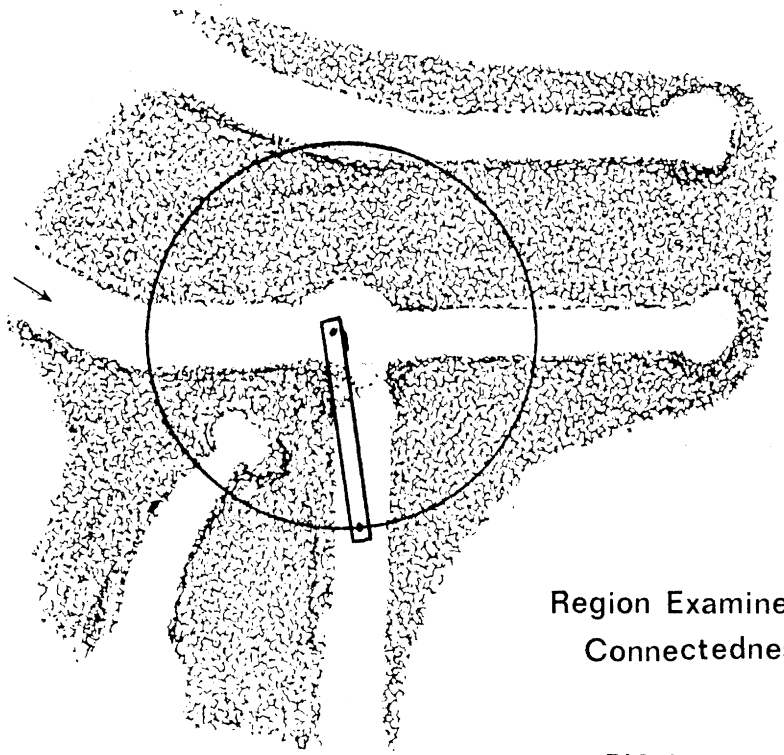
When two peaks are so matched, they are merged to form a "wire," and its width, maximum intensity, average intensity, and position are noted. In addition, the average intensity of the "background" around the wire is noted. This, together with the wire's average intensity, is later used to calculate intensity thresholds which can be applied to detect the wire.

If the picture is of better quality, the filtering-averaging step is eliminated. In this case, the circular intensity array is examined directly for positive peaks, again using an algorithm similar to Sharai's. Any peak above the threshold WIRE-TH which is within the minimum and maximum allowable widths is taken as a possible wire. Again the widths, intensities, and positions of these peaks are recorded.



Proposing Suspects

FIGURE 6



Region Examined to Verify Connectedness

FIGURE 7

SUSPECT VERIFICATION

Once a suspect has been found, a two step verification procedure is applied. We first determine if the suspect segment actually runs into the node, or if it is part of a separate unconnected circuit. This is done by scanning a band of points along a line between the center of the node and the location of the suspect and comparing their intensities to a threshold.

A node is typically a pad where a wire from the front of the circuit board comes through and is soldered down. Under all but the most diffused lighting, the lead and solder blob cast shadows which can obscure part of the node and segments leading into it. Another problem arises when we are not situated on the center of the node. In this case the band of points may skirt along the edge of the background insulating material. For these reasons, the following criteria for connectedness are used.

The sampled points are examined for values less than the threshold CONNECTED-TH. This threshold is chosen to be slightly less than the current value of WIRE-TH to offset the problem of weakly shadowed regions. To handle the problem of small darkly shadowed patches and slight excursions into the background, we will accept points below the threshold if they are isolated or occur in short sequences. The maximum length of such a sequence, CONNECTED-GAP, is chosen to be about one-third the typical wire width. If no long dark stretches are found, then the segment is taken to be connected to the node (see figure 7).

In the second part of the verification, we determine that the suspect is indeed a segment, and get a better estimate on its width, average brightness, and local direction. This is done by the function SAMPLE-TRACK which applies FIND-WIRE at several points along the hypothesized direction of the segment (see figure 8). The function FIND-WIRE scans a band of points perpendicular to the hypothesized direction of the segment and uses procedures similar to those in the circular search to detect the wires. If more than one is found, the one closest to the center of the scan is returned.

If wires are found in three out of four attempts, SAMPLE-TRACK fits a line to their centers to determine the local direction of the segment. This direction, along with the wire's average width and brightness and the average brightness of the background around the wire is passed to the function FOLLOW-SEGMENT. If fewer than the requisite number are found, the suspect segment fails verification. Figure 9 shows the results of TRACE-FROM-NODE on the previous figure.

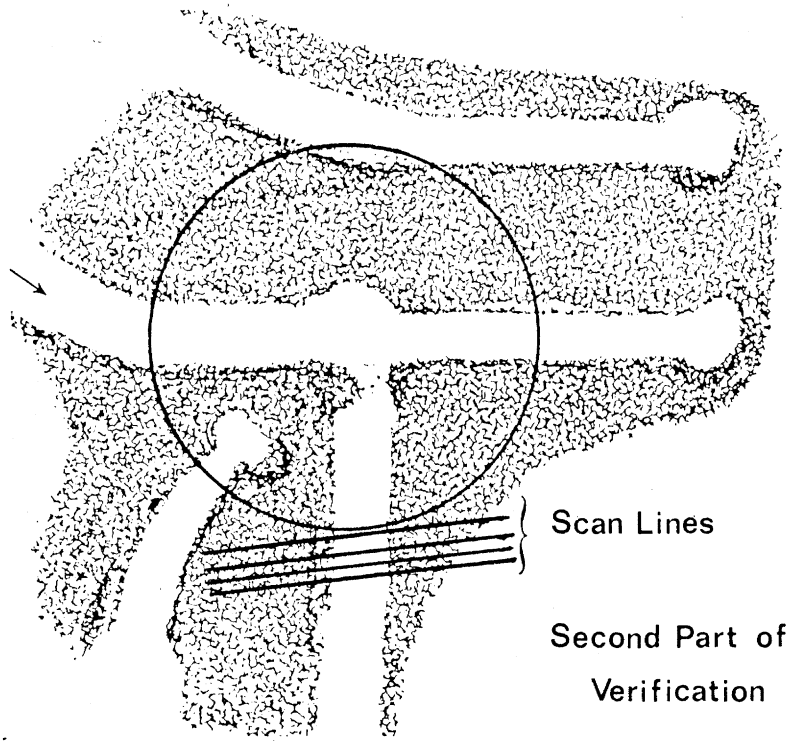


FIGURE 8

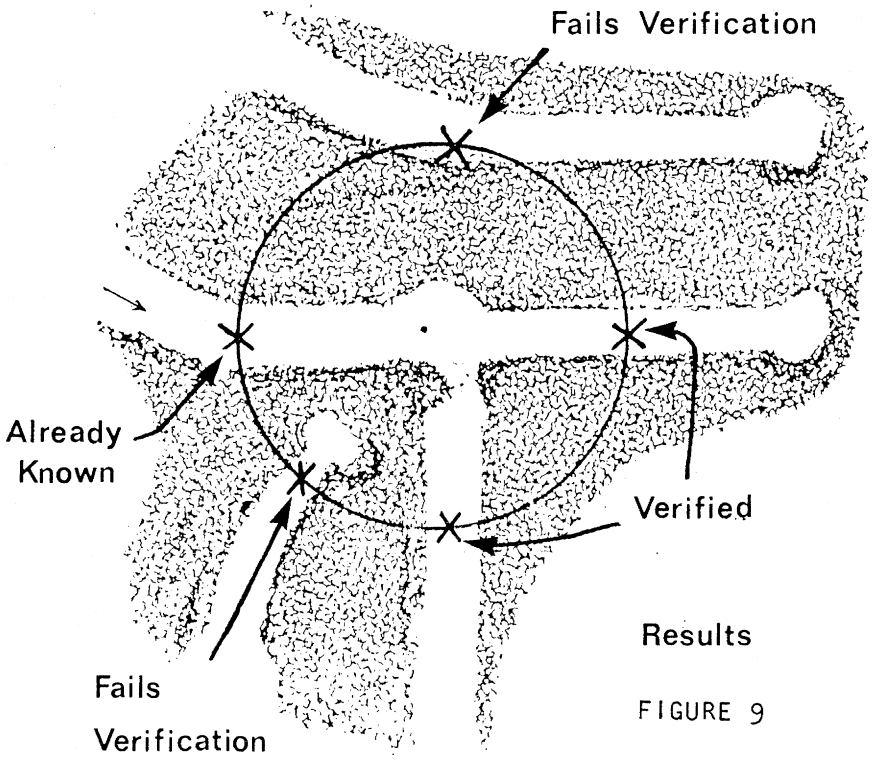


FIGURE 9

TRACKING A SEGMENT

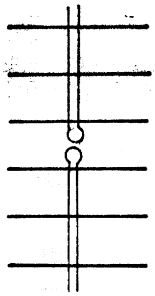
One can use standard line following ideas , but they leave several problems:

1. Insuring continuity.
   The back of a circuit board has many wires which may run very close together. We want to be sure that we do not jump from one to another (figure 10a). This forces us to keep the separation between scan lines small.
2. Detecting branches.
   Unless a scan line falls exactly on the place where a circuit branches, that branch will be missed (figure 10b). To detect these branches, we must either keep the scan separation small or scan along additional lines parallel to the segment.
3. Detecting pads.
   Pads are detected by noticing a "bulge" in the width of the wire. Pads may be missed unless we use closely set scan lines (figure 10c).

Although keeping the scan separation to a minimum solves all three problems, it forces us to sample and examine a large number of points, many more than are neccessary to merely follow the segment. For this reason, a different approach was developed. The basic idea is to scan along lines parallel to the direction of the segment. Points inside the wire and on either side are examined. The inside track insures that we are still on the wire and that it is continuous, while the side scans are used to servo the direction and propose pads and branches. Unless the picture is very bad, a simple threshold can be used to distinguish the conducting wire from the insulating background. This threshold is allowed to "float" so that it will be good for at least a local area of the picture. This procedure can be viewed as a special case of scanning along lines perpendicular to the segment -- one in which we sample a minimum number of points.

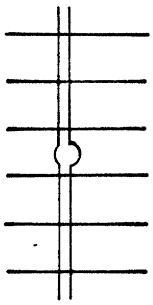
PROBING A SEGMENT

In following a segment, the basic procedure is to "step" the function PROBE along the hypothesized direction of the wire. This function samples points within and on either side of the wire, thresholds them, and determines whether we should go forward, turn right or left, or stop tracking. If PROBE's advice is to turn or go forward, a new point is calculated and the process continued. If PROBE suggests stopping, additional procedures are invoked which make a closer examination of the situation.



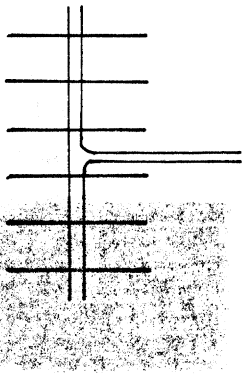
A

Skipping



B

Missed Pad



C

Missed Branch

FIGURE 10

THE PROBE

Figure 11 shows one application of PROBE along a segment. The point C, the hypothesized center of the wire, is first sampled. If its intensity is below the threshold PROBE-TH, 'STOP' is returned, indicating the end of the segment, a shadow, or noise. If it is above this threshold, then the two points L and R are sampled. The points are situated at a distance D1 from the point C in a direction perpendicular to the hypothesized segment direction. This distance is chosen to be 80% of the segment's typical width. If both these points are below the threshold, then we are roughly centered on the segment and PROBE returns 'FORWARD'. If both points are above the threshold, a pad is hypothesized, PROBE returning 'PAD'.

The other cases, in which one of the points is above the threshold and the other below, are ambiguous. We may simply have wandered from the center of the segment, or we may have reached a branch in the circuit. To differentiate these cases, an auxiliary point is sampled on the side which was above the threshold. This point is located a distance D2 from C, this distance being 1.75 times the width of the segment. If the intensity at this point is greater than the threshold, PROBE hypothesizes a branch toward that side (returning 'SUSPECT-LEFT', or -RIGHT'). If the point is below the threshold, then PROBE reports that we have wandered from the center of the wire, returning 'TURN-RIGHT' or 'TURN-LEFT'. Figure 12 shows a flow chart for PROBE.

It is important to sample these extra points only when the closer side scans suggest it. This is because the circuits may run close to one another, resulting in many false branch hypotheses if the outermost side scans were continually checked.

FOLLOW-SEGMENT

The function FOLLOW-SEGMENT does the work of guiding PROBE, stepping it along the wire (using the feedback to servo the direction), and adjusting the various parameters PROBE uses. FOLLOW-SEGMENT is initially given the following description of the segment (obtained from the call to SAMPLE-TRACK when the segment was verified):

1. An initial point on the wire.
2. An initial direction.
3. The wire's average width (W).
4. The average brightness of the wire (B1).
5. The average brightness of the background around the sampled part of the wire (B2).

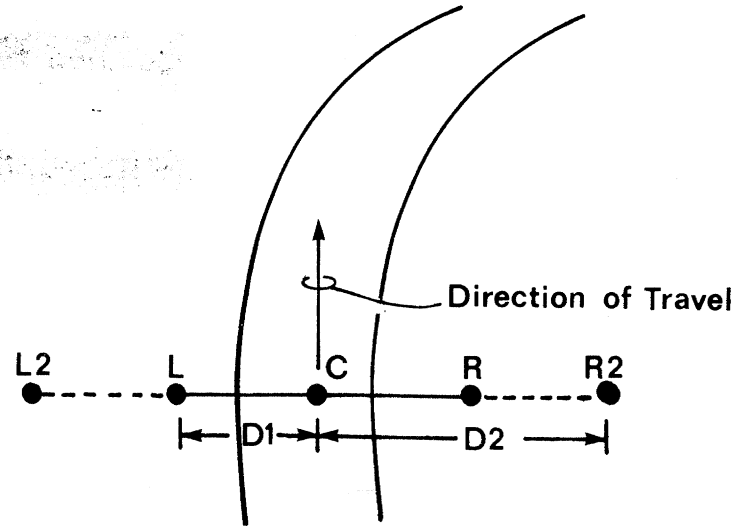


FIGURE 11

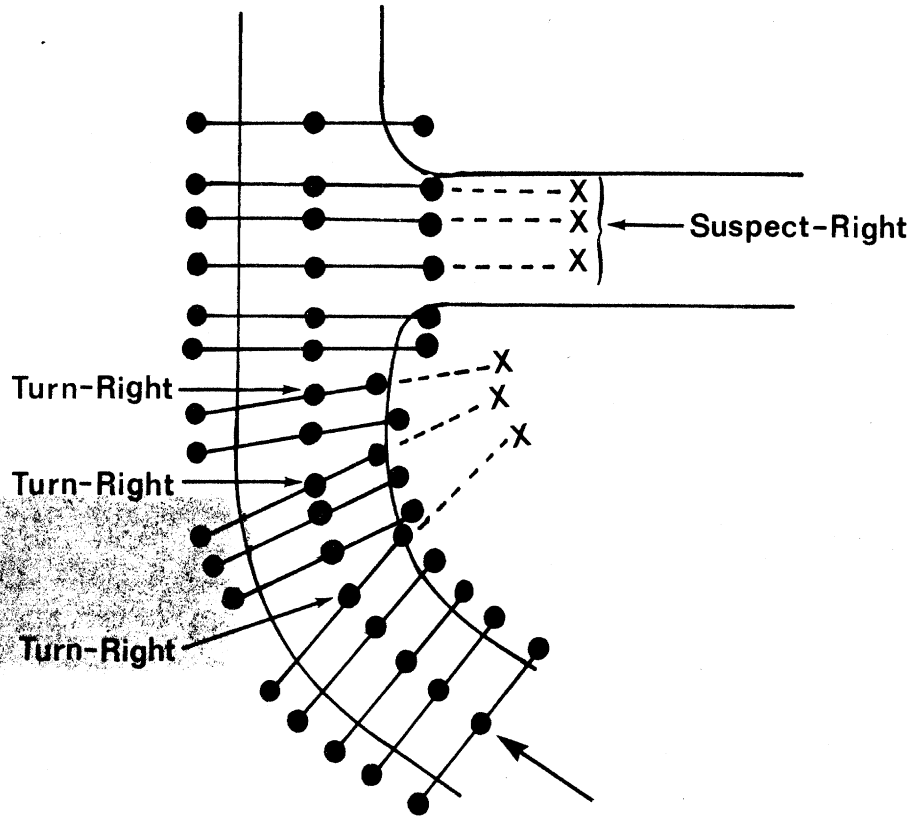


FIGURE 13

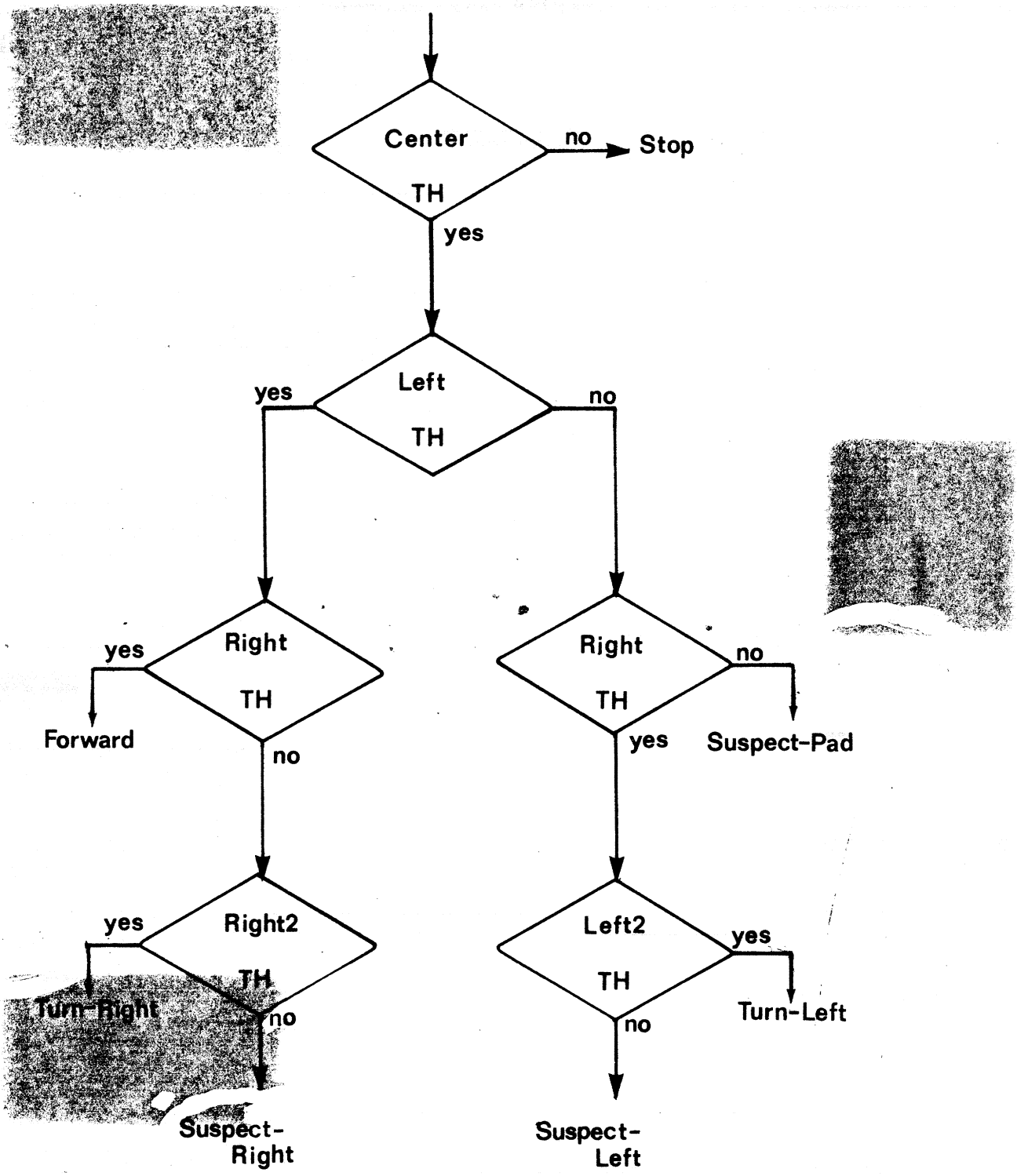

Center
TH
no
Stop
yes
Left
TH
yes
no
Right
TH
yes
Forward
no
Right
TH
no
Suspect-Pad
yes
Right2
TH
yes
Turn-Right
no
Suspect-
Right
Left2
TH
yes
Turn-Left
no
Suspect-
Left

FIGURE 12

From these values, various parameters are calculated for the tracking process. The major parameters are:

PROBE-TH <-- B2 + 0.4*(B1 - B2)
PROBE-STRIDE <-- 0.33*W
PROBE-SIDE-STEP <-- 0.25*W

Once these have been calculated, Probe is applied to the initial point. If a node or the end of the segment is not proposed, a new point is calculated and the process repeated. The coordinates of this new point depend on PROBE's return as follows:

1. FORWARD
   Step PROBE-STEP units forward.
2. TURN-RIGHT
   Decrement the current direction by PROBE-ANGLE-INCREMENT, step towards the left (perpendicular to the direction) PROBE-SIDE-STEP units, and step forward in the new direction.
3. TURN-LEFT
   Similar to above, but in the opposite direction.

If PROBE returns SUSPECT-RIGHT, SUSPECT-LEFT or PAD, FOLLOW-SEGMENT defers to HYPOTHESIZE-NODE. This function continues to step PROBE along the current direction until the nature of the segment changes. It then creates a hypothesized node midway between the first and last suspect reported by PROBE.

If PROBE reports the end of a segment, a closer look must be taken. A number of situations might cause the hypothesis of the end of a segment besides its actual termination. We may have run into a shadow near a node, some noise, or a corner too sharp to navigate with our simple curve following mechanism. It might also be the case that the threshold (PROBE-TH) we originally calculated is no longer suitable to detect the wire.

FOLLOW-SEGMENT attempts to find a continuation of the wire calling SAMPLE-TRACK to take a more detailed look. As described before, this function applies FIND-WIRE to points along the hypothesized direction of the segment. If wires are detected, the description of these wires is used to recalculate the "probing" parameters. The process is then resumed with the new initial point and direction. If no wires are found, the end of the segment is hypothesized, and control returned to TRACE-SEGMENT. Figure 13 attempts to give a graphic description of this process.

RESULTS AND CONCLUSIONS

These programs, together with an additional module TRACE-FROM-PINS were tested on several printed circuit boards with good results. A common type of circuit board has most of its circuits terminating at a strip of pins on one side of the board. TRACE-FROM-PINS, is given the endpoints of a line which crosses this strip and scans along this line. Whenever it discovers a wire, TRACE-CIRCUIT is applied to this point.

Figure 14 shows the results of this process applied to a typical board. TRACE-FROM-PINS was given the coordinates of the points A and B and correctly found all of the pins. Note that some of the pins are no more than solder blobs, with no circuits attached. The simple mechanism for following segments worked surprisingly well, all of the segments being properly tracked.

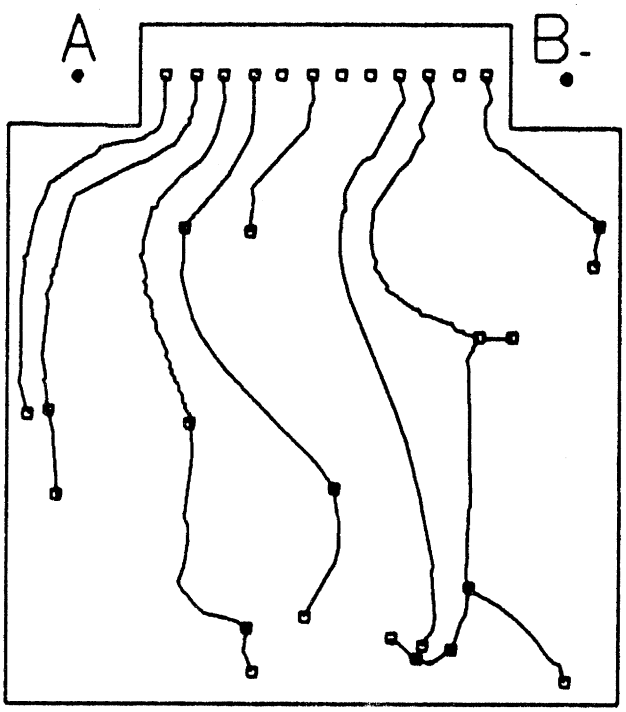


FIGURE 14

LOZANO'S RESISTOR FINDER

The goal of the work reported here was to write a set of programs that would recognize resistors. We begin by showing what typical circuit board components look like through the vidisector camera.

Resistors

1. Figure 1 shows the intensity profile through the dark part of a resistor. Points labelled B correspond to board material. Points labelled R correspond to the dark material.

2. Resistors often have a highlight along the center. This is due to specular reflection along the line of maximum curvature with respect to the light source. The matting spray used to reduce the intensity of these highlights and thus protect the vidisector also tends to spread them out. Figure 2 shows a glossy resistor. The point labelled G indicates the gloss. Bs indicate the board and Rs the dark resistor masterial.

3. Figure 3 shows a typical color code band. Notice the bands become quite dark near the edges of the resistor. This is because the intensity of the reflected light is roughly proportional to the cosine of the angle between the incident angle (which in this case happens to equal the viewing angle) and the normal to the surface (see (Krakhauer TR-234) for a discussion of this). Bs indicate board brightness.

4. Leads are at least as bright as bands but they do not become as dark at the sides. Figure 4 shows the profile of a lead. The L indicates the lead, Bs indicate board brightness and Rs indicate resistor brightness. Notice that the width (the number of points between points equal to board intensity) of the peak is very close to the width of the resistor, making it difficult to identify a lead by its width. Instead, the fact that it does not become quite as dark at the edges must be used.

5. Figure 5 shows the crossectional intensities along the length of a non-glossy resistor. Bs indicate bands, Rs indicate dark resistor material, L indicates the lead and Ps indicate board intensity. Figure 6 shows a glossy resistor.

Dark Ceramic Capacitors

A rough drawing of these capacitors is shown in figure 7. They appear as an almost rectangular region with a dark outline and a bright highlight along the center. The variation in intensity along this highlight is quite large due to surface irregularities. Figure 8 shows a crossection across such a capacitor. Bs indicate board material, Os indicate the dark outline and the H indicates the highlight. Figure 9 shows a profile along the capacitor, the labelings are the same.

Metal Case Transistors

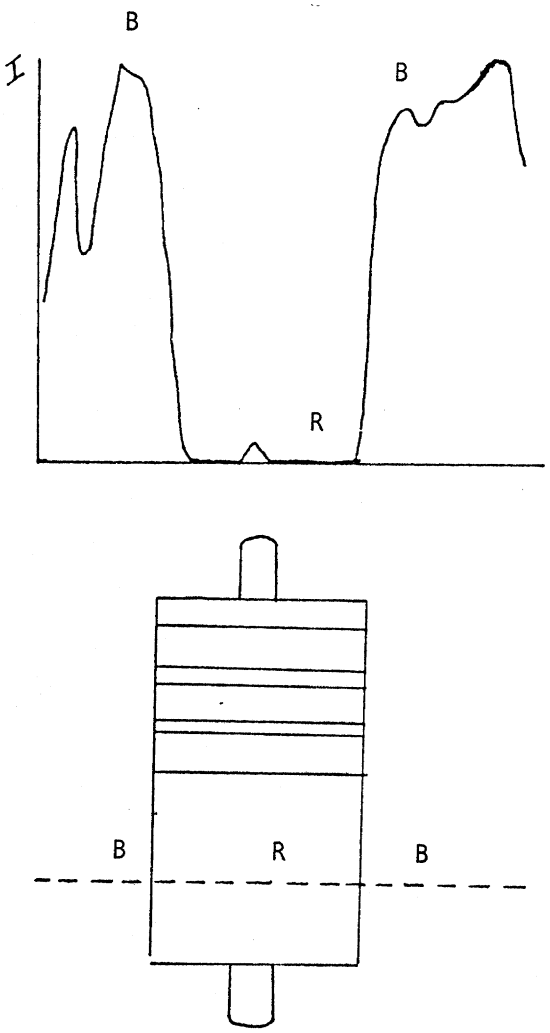


FIGURE 1

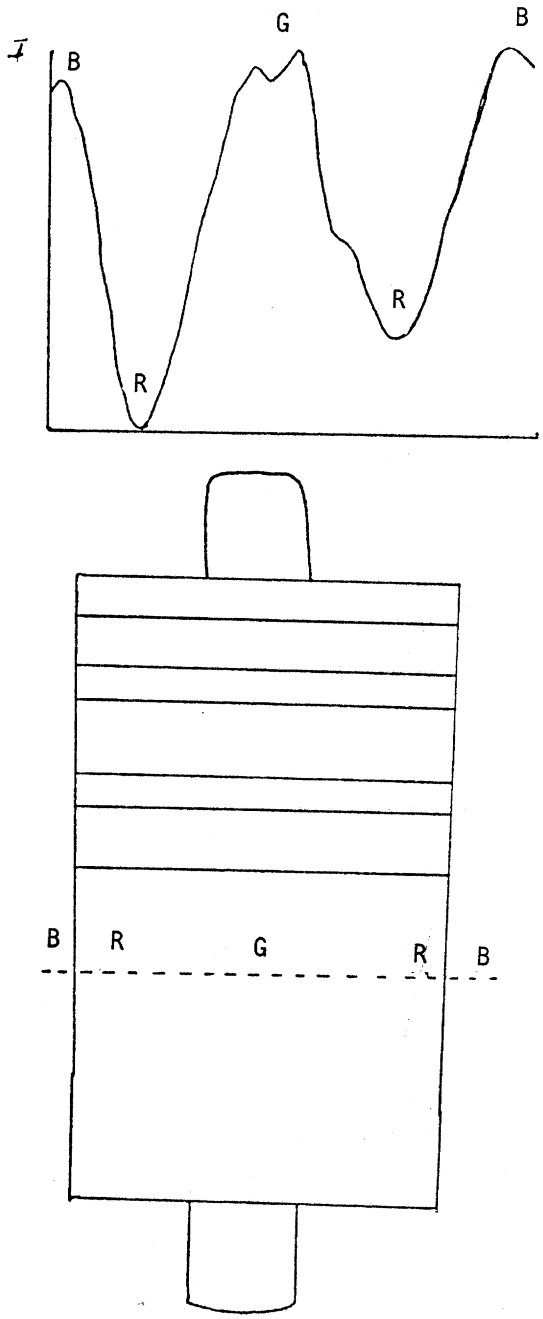
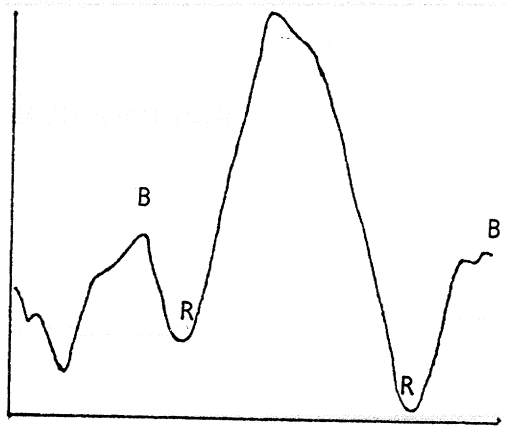
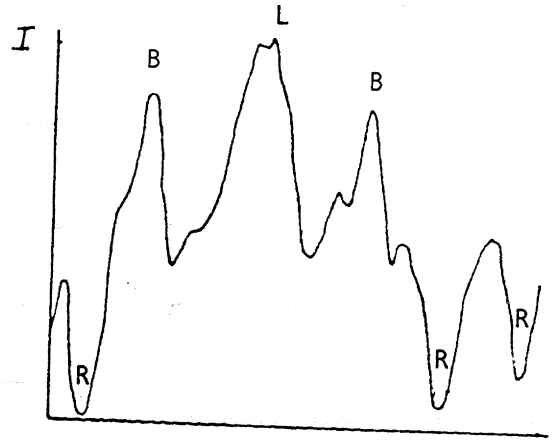
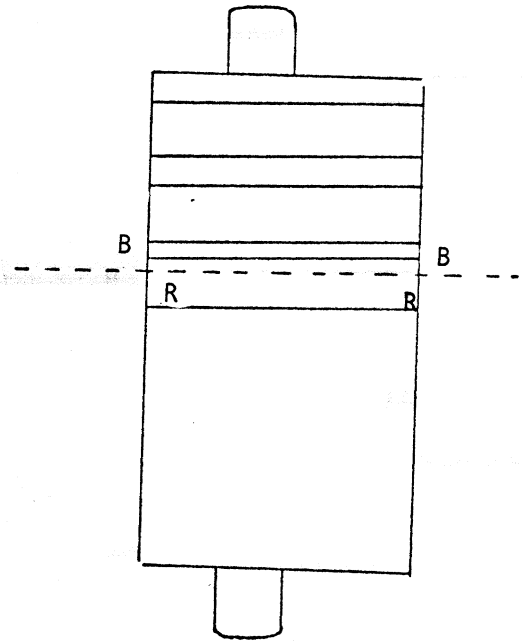


FIGURE 2







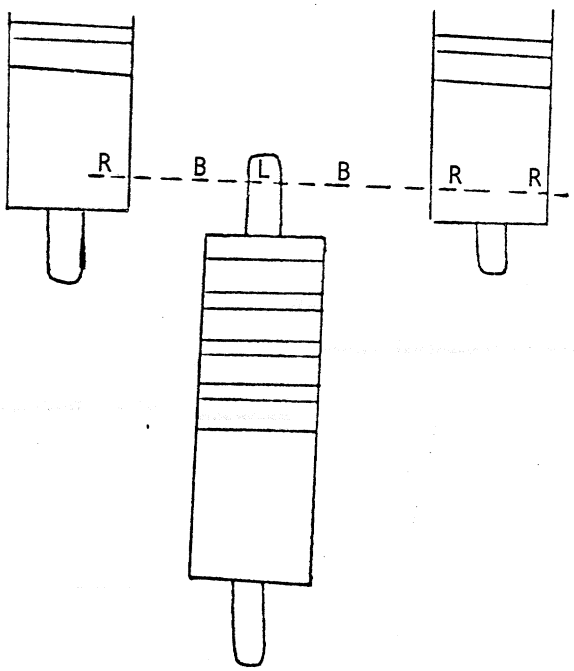


FIGURE 3

FIGURE 4

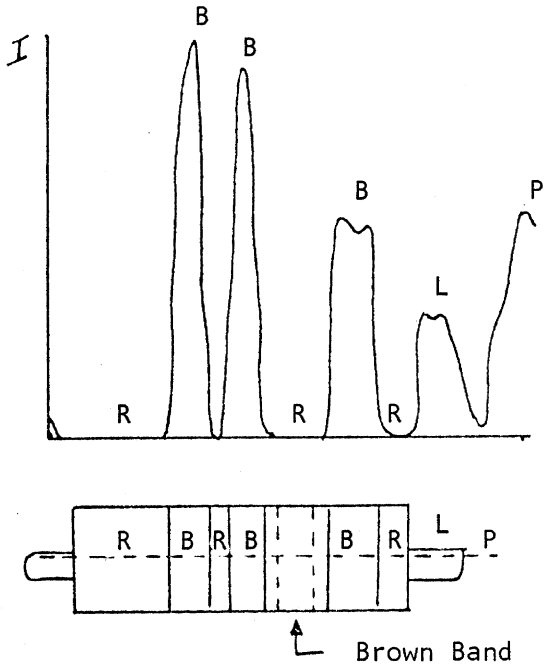


FIGURE 5

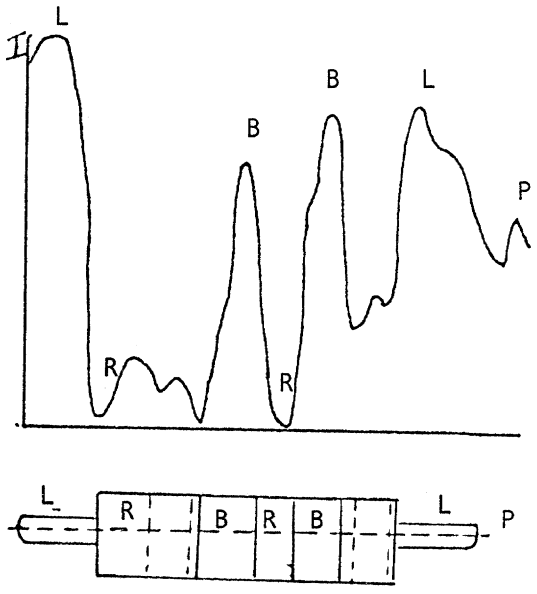


FIGURE 6

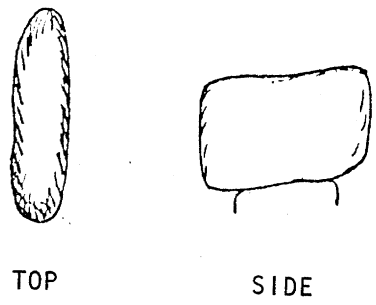


FIGURE 7

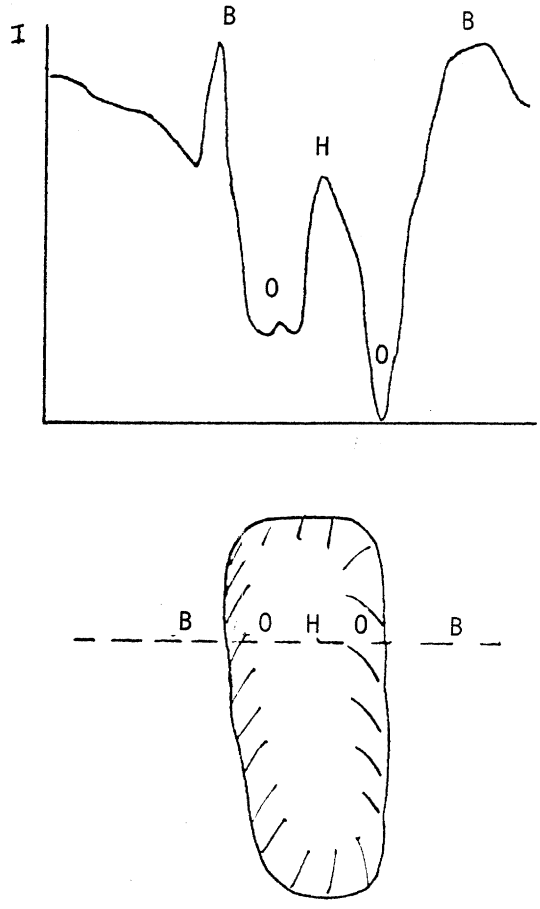


FIGURE 8

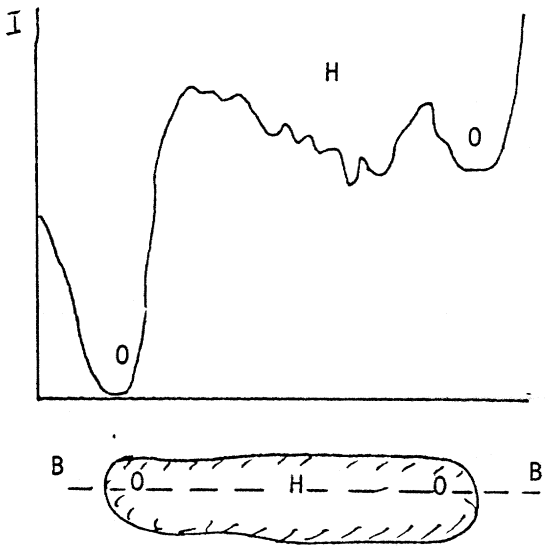


FIGURE 9

The key feature of these components is a very bright circular region. Figures 10 and 11 show orthogonal crossections across the top of a transistor. The gradual slope of the intensity profile in Figure 11 is caused by the angle of the reflecting surface with respect to the light source and the vidisector.

STRATEGIES AND APPROACHES

Given the problem of finding and identifying the components on a circuit board, there are, of course, several alternative approaches to the standard profile oriented approach used here. These alternatives include:

1. line finding
2. depth
3. color
4. texture
5. special markings

All were rejected for one reason or another, some because they seem inadequate,

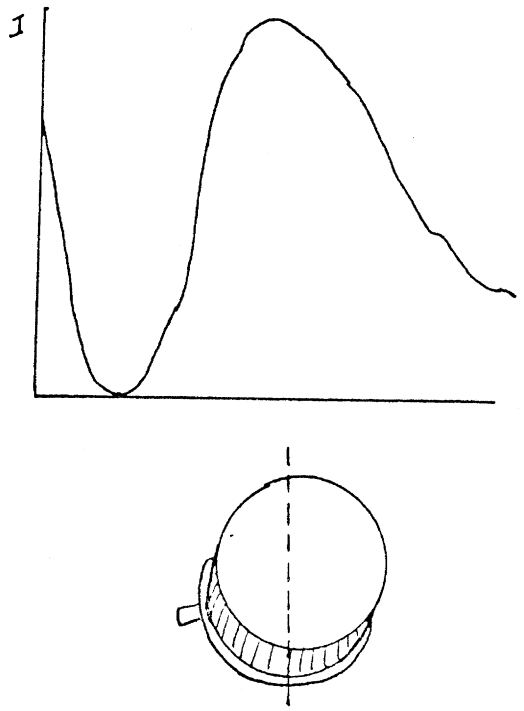


FIGURE 10

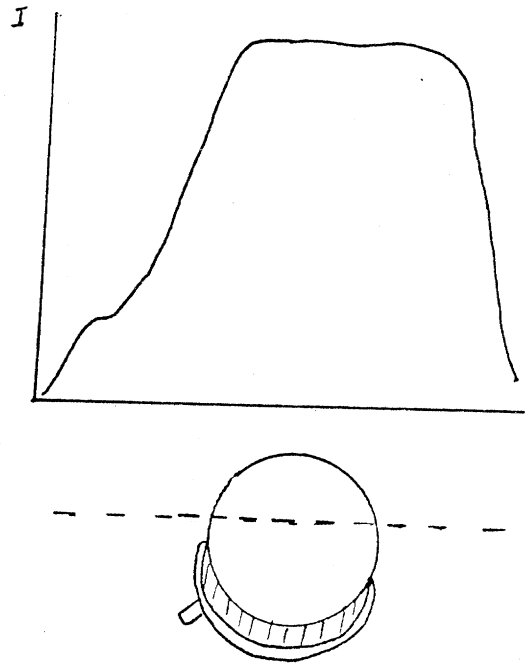


FIGURE 11

some because they require equipment not available, and some because important questions would not be squarely faced.

In any case the programs described attempt to address several issues:

1. The effectiveness of the use of typical intensity profiles.
2. The interaction and relative effectiveness of these profiles vs. line oriented data.
3. Organizational issues, such as how to construct and how to change hypotheses.

The overall structure of the programs is diagrammed in figure 12. Shown are the resistor hypothesizer and predicate along with their supporting programs, including a proposer and a verifier for bands, ends, leads and lines. The programs are given the coordinates of a section of a circuit board and they produce a data base containing a description of any resistors found there. We begin the explanation with an overview, and we will follow that with more detail.

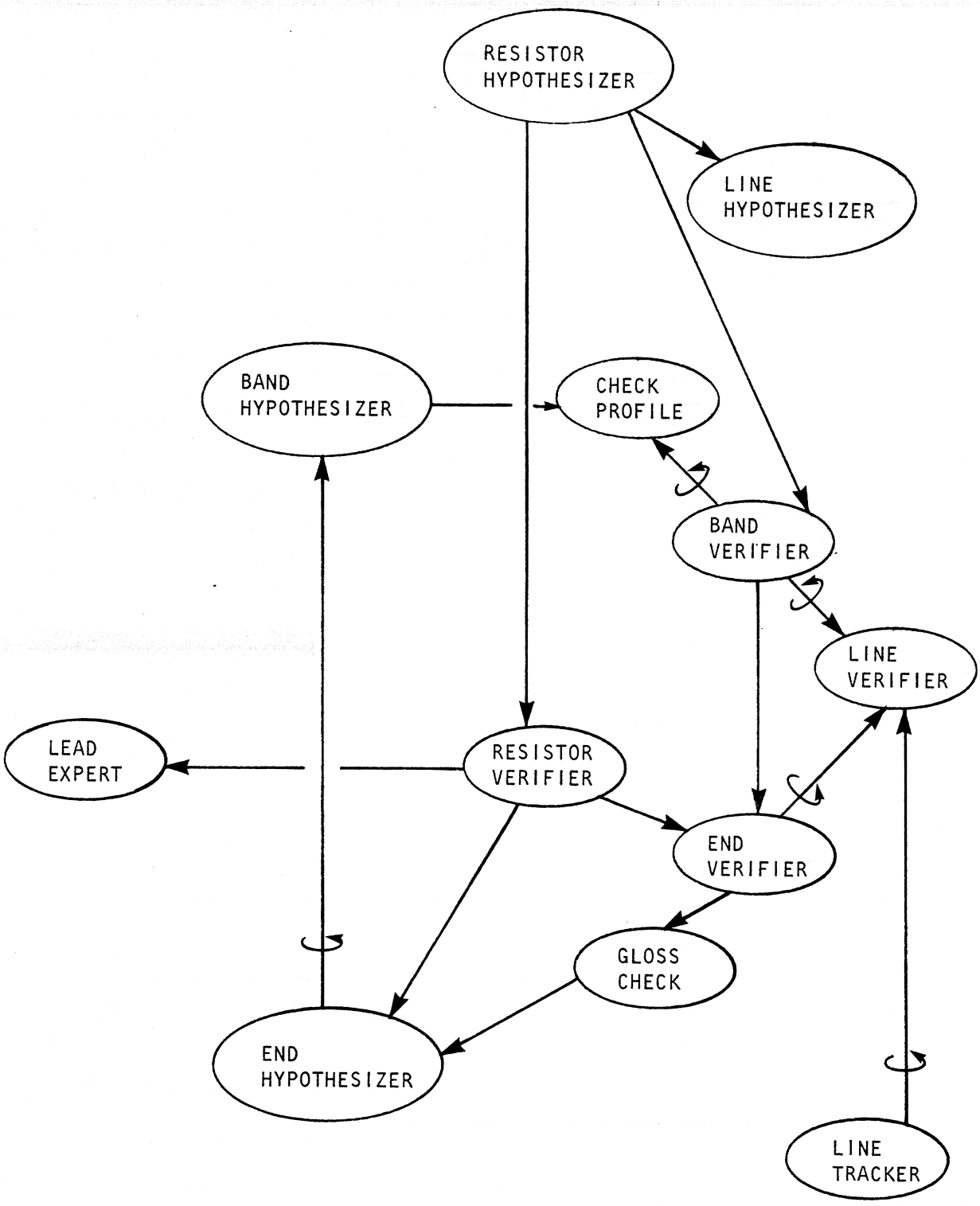

RESISTOR HYPOTHESIZER
LINE HYPOTHESIZER
BAND HYPOTHESIZER
CHECK PROFILE
BAND VERIFIER
LINE VERIFIER
LEAD EXPERT
RESISTOR VERIFIER
END VERIFIER
GLOSS CHECK
END HYPOTHESIZER
LINE TRACKER

FIGURE 12

Upon being called into a region, the hypothesizer scans the region horizontally and vertically in search of a line separating the bright board material from a dark region. If this were the side of a resistor a band should be near on either side of the point of entry. A search for this band is carried out parallel to the line. The hypothesizer calls the band proposer which looks for a profile that might belong to a band. If one is found then the band verifier is called. Provided that the band can be verified the resistor predicate is called, otherwise the hypothesizer continues its search elsewhere for another resistor.

The second side of the resistor is found in band verification. The resistor predicate is thus called with both sides of the proposed resistor and a band known. It then starts off by calling the end proposer, which, in turn, calls the band proposer. Any bands found are stored in the data base marked as likely. When a bright region is found which fails as a band, the band proposer suggests that it might be an end. The end proposer then checks to see if it fulfills the characteristics of an end. If so, the end is proposed and an attempt is made to verify it. If it succeeds the predicate calls the lead proposer and verifier. If an end cannot be verified the predicate goes back to the nearest band which has been proposed, and if it can be verified, attempts to find the end by tracking the side of the resistor. If none of the bands can be verified the resistor hypothesis is flushed. If in fact the ends can be verified the resistor is fairly sure. An attempt is then made to find any bands the band proposer might have missed.

DETAILS

We now look at the steps in the analysis in more detail. Note that most of the programs in the system accept a hypothesis and attempt to expand or verify it.

1. Given a section of a circuit board the hypothesizer's first task is to isolate a region of possible interest. This is done by sampling the whole region very coarsely and choosing regions that are darker than 80% of the board and larger than some minimum area. This is based on the fact that the resistors and ceramic capacitors are among the darkest region of a circuit board.

2. The hypothesizer scans the region horizontally and vertically for an edge going from the bright background to darkness. The hypothesizer assumes that it is entering the resistor from the side into one of the dark regions of the resistor (see figure 13). This assumption simplifies the programming at the cost of another scan of the region.

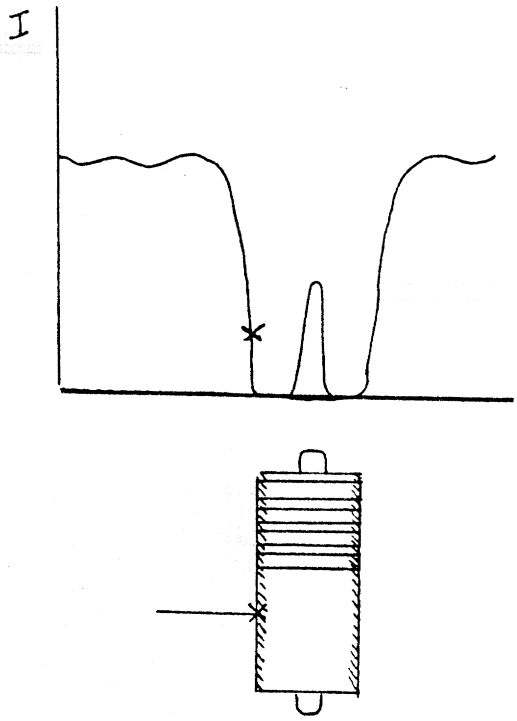


FIGURE 13

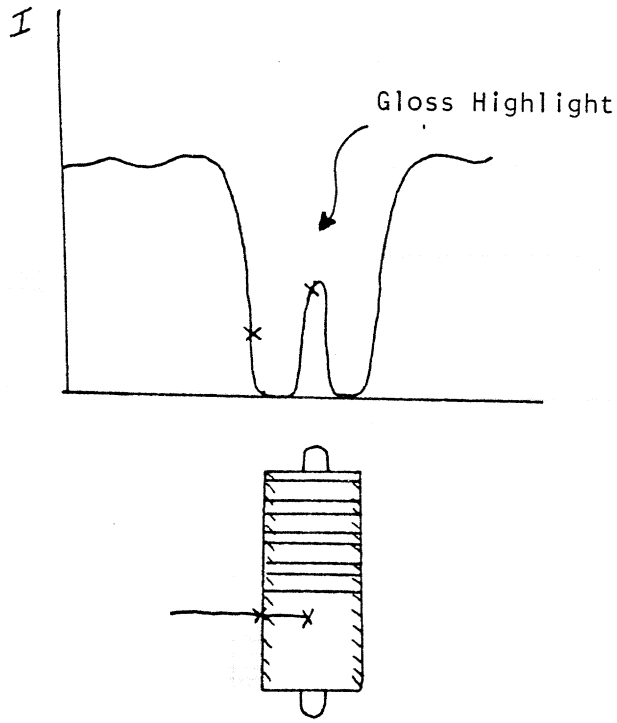


FIGURE 15

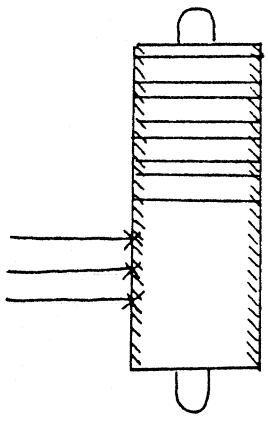


FIGURE 14

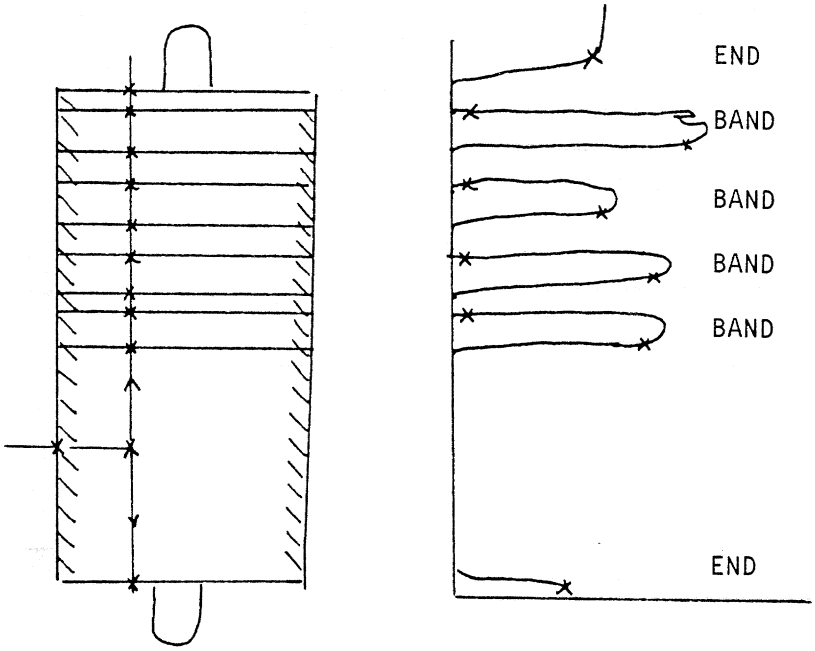


FIGURE 16

3. Once an edge has been found in 2, several scans are made parallel to the original one. If enough edges are found then a line equation is computed by a least squares fit (See figure 14).

4. A positive edge is now sought in the direction perpendicular to the line found in 3. This edge is conjectured to be either the other side of the resistor or a gloss highlight (see figure 15).

5. Now the band proposer is called and told to look for a band starting at a point half way between the edges found in 2. and 4. with a direction equal to that of the line found in 3 (see figure 16). The band proposer looks along this line for a sharp positive transition followed by a similar but negative transition. At the top of the positive transition it looks perpendicularly to its direction for a negative transition. It is thus using the typical profile of a band (as shown earlier) to help hypothesize its presence. The band must fulfill certain conditions in order to qualify. It must become dark enough at the sides of the resistor. Its width must also be within a bound set by previously discovered bands and by the known relationship between maximum band size and the width of the resistor.

6. The band verifier is then called. This function checks that the sides of the band coincide with those of the resistor. The verification of the sides of the band is used to update the equations of the lines representing the sides of the resistor. If, as in this case, the equation of one of the sides is not known the new equation is stored as the corresponding side of the resistor. It also checks the profile of the band along the resistors axis at several other crossections closer to the edges of the resistor (see figure 17). The assumption is that a real band extends almost to the sides of the resistor because the intensity goes down gradually with curvature while noise peaks and glossy highlights are more restricted to a narrow band along the middle of the resistor. If both the profiles are found to be end profiles then the verifier fails and returns a message suggesting the end. If a mixed verdict (band and end profiles) is received the program lowers the threshold of acceptance in the band proposer and calls it again. If the end profile is now found to be a band profile the verifier succeeds. Otherwise it fails with no suggestions.

7. If the band verification fails then the same process is carried out going in the opposite direction, along the axial line. If no bands are proposed then the resistor

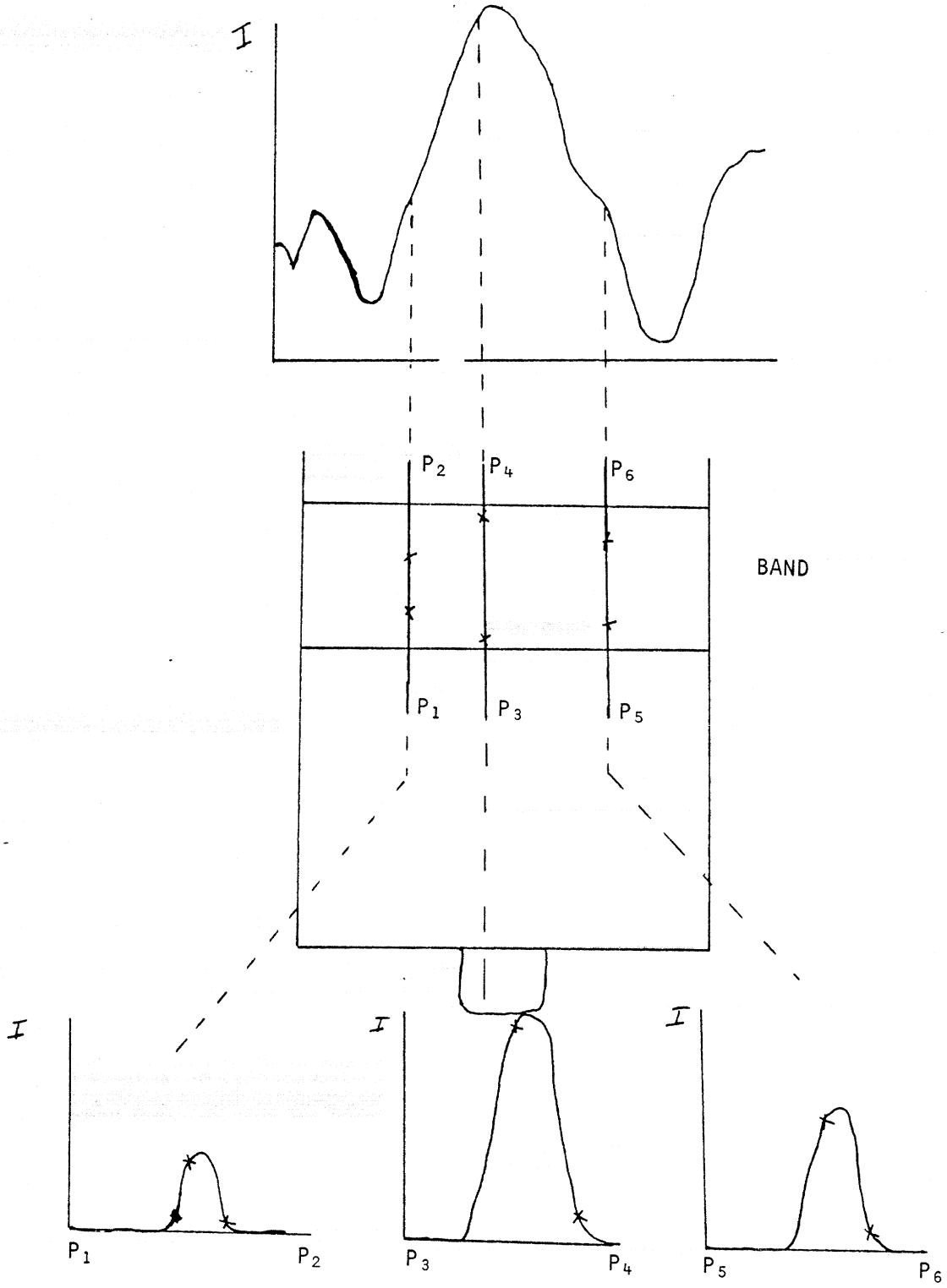


FIGURE 17

hypothesizer declares failure. This would result in a call to the capacitor hypothesizer since a dark glossy object without sharp bands might be a ceramic capacitor.

8. If the band is verified then one knows where both sides of the resistor are (from steps 2 and 6) and the coordinates of one band. Now the end proposer is called. This program calls the band proposer. Starting from the known band it searches for a band in the area the band can be expected to be from observed typical interband spacings. If a band is found no attempt is made to verify it. The program then repeats the process starting from this most recent band. If no bright region is found the program sets a "careful" flag and keeps moving down the resistor. If a band is proposed while in 'careful' mode an attempt is made to verify it. Success of the verification procedure causes a "possible missing band" to be signaled between the two bands. This mechanism serves to help overcome the vidisector's inability to detect red, brown and green bands very well. If the band verification fails an end is hypothesized there.

9. On the other hand, if while looking for a band, a bright region is found but with no corresponding transition to darkness within the maximum allowed width for the band, the band proposer fails. This causes an end to be proposed. The band program also fails when the sides of the proposed band are either not dark enough or else cannot be found at all.

   The use of the band proposer to propose ends illustrates a general technique used to change hypotheses. A new hypothesis can be suggested either by the occurrence of a particular feature or by the failing in a particular fashion (i.e. with a known message) of another hypothesis. Thus it is with ends and capacitors. An end is a failed band and a capacitor is a failed resistor. The process of failing is handled by a system function which keeps a data base of what to suggest upon failure of a particular routine. In some sense the entries in this data base are "similarity pointers" in the sense of Winston's "similarity nets." This process not only allows transfer between different concepts but is an efficient way of storing procedural knowledge without unnecessary duplication.

10. An attempt is then made to verify the proposed end. This is done by verifying that the sides of the resistor continue up to that point but stop there. If the sides are found to continue beyond the expected end of the resistor the program tracks the

lines to their end. If in the original end verification step no lines are found near the expected sides of the resistor an attempt is made to track the side of the resistor from the closest proposed band which can be verified. If none of the bands can be verified the resistor predicate declares failure. Once an end is proposed by the tracking program an attempt is made to confirm it by looking for a sharp discontinuity in the gloss highlight at that point. This is very helpful in pinpointing shadowed ends. If the gloss check fails then the end proposer is called again. The profiles are then used to suggest a new location for the end. Failure of this operation causes failure of the current resistor hypothesis.

11. If the one end succeeds a lead proposer and a verifier are called. These look for a thin bright region midway along the end of the resistor. It expects a dark region where the lead goes into the board.

12. Knowing where one end of the resistor is, the end proposer duplicates steps 8, 9, 10, and 11 but in the opposite direction. It uses its knowledge of the average size of previous resistors to set a limit on its exploration.

13. Once both ends have been found, a check is made to see how many bands have been proposed. If more than four have been found then an attempt is made to verify them. If less than four then the resistor predicate checks to see if a "possible missing band" has been proposed. The band verifier is then called.

14. The predicate then attempts to check these proposed bands by running the band proposer with lowered thresholds. If no success is had in finding them, then the proposal is left in the data base marked as tentative.

15. The predicate then updates any parameters such as average resistor length etc. by using information about the current resistor.

It is interesting to note that even though the edge finder and the line verifier are very sensitive to noise, the global information obtained from the profiles and the other constraints work together to produce fairly fast and accurate results.

7 ARTIFICIAL INTELLIGENCE AND THE PHILOSOPHY OF EDUCATION

Artificial Intelligence can be defined in a narrow and instrumental sense as the enterprise of endowing machines with certain abilities one would describe as intelligent (or "cognitive") when seen in animals or human beings. The history of the subject shows clearly that the pursuit of such goals has interacted richly with a broader set of activities which lead to a definition of Artificial Intelligence as a certain style of studying cognitive processes in general, whether these happen to be embodied in computers, humans, animals or, conceivably, other forms such as social organizations. The fundamental characteristic of this style of cognitive theory is the systematic use of technical ideas drawn from the theory of computation to serve as a conceptual framework for understanding intelligence in all its manifestations.

GENERAL REMARKS

If a new style of cognitive theorizing leads to insights into human intelligence, it ought to be possible to use it to produce improvements in human ability. The following pages describe a number of explorations, theoretical and experimental, in the direction of developing new approaches to human learning and education. The use of the word "exploration" is deliberate. The work is still very far from having the status of rigorous experimentation or of fully understood theoretical issues. But here, as in work on machine intelligence, we believe that the methodological strategy has to rely on good judgment to achieve a balance between getting systems to work and developing theories of why and how they do. In the case of machine intelligence a system consists typically of a computer program or of a complex of programs and sensory and effector devices. In the case of an experiment in education a system typically takes the form of a learning environment which we shall describe presently by distiguishing (somewhat artificially) its material, conceptual, and social components. In both cases the experiment is, of course, meaningless unless results are obtained. When they are, there remains the task of understanding the interactions of components of a complex system. From an immediate practical point of view the important fact might be that the system works well enough to be used. Our explorations in Education have produced results of both sorts -- theoretically rich areas for further study and an immediately applicable spin-off which is attracting the attention of educators in several sectors: grade school, handicapped children or adults, adult education and others.

A LEARNING ENVIRONMENT -- A MINI-WORLD OF TURTLES

The example of a Learning Environment we have chosen for most detailed description shares with many of the more successful A.I. programs the feature of focusing on a well defined mini-world. The choice (or rather, the creation) of suitable mini-worlds is an important (and not at all easy) part of the research strategy. In this case the mini-world is intended for use with children, and so one might have expected that the well known "blocks-world" would be suitable. In fact it is not, for reasons which will appear later, and instead we invented a new mini-world, the "turtle world," which has since shown itself to be well-suited as a mini-world for machine intelligence programs as well as for children. The simplest turtle is an entity whose state is a position on a plane surface and a heading. There are two state-change operations:

1. The command FORWARD 20 causes the turtle to change the position component of its state while keeping the heading fixed; in fact the turtle moves 20 units in the direction of its heading.
2. The command RIGHT 30 causes the turtle to change its heading by rotating 30 degrees clockwise; its position is not changed.

THE MATERIAL COMPONENT OF A LEARNING ENVIRONMENT

There are several material forms in which the turtle world can be embodied for a child. In each there is some kind of computer terminal through which the child and the turtles can interact -- ranging from specially designed terminals for very young or physically handicapped children to quite traditional terminals made for regular computer scientists. The turtles, too, can take different forms appropriate for different learning experiments. The three basic types are "floor turtles" -- small mechanical robots with wheel structures designed to allow the motions FORWARD and RIGHT; display turtles which draw a line on a CRT; and "plotter turtles." The common feature across all these variants is that the child learns to control the turtles by using the commands FORWARD and RIGHT embedded in a programming language, usually LOGO, in our experiments with elementary school children and older subjects.

Given a turtle, computer access and a programming language a child is able to build up behavior patterns which might cause turtles with writing capability to draw or turtles with sensory capability to move in simple environments. In typical experiments children are exposed to one form or another of a turtle environment for between one and forty sessions each lasting between a half and two hours.

SOME CONCEPTUAL COMPONENTS OF A LEARNING ENVIRONMENT

A computer and a turtle can lead to engrossing activities (e.g. "display hacks") provided one has an appropriate knowledge base. The conceptual component of the learning environment is a set of knowledge structures designed to allow the child to progress as rapidly as possible towards gaining power of action both in relation to making the turtle do interesting things and in relation to increasing his own knowledge. Considerations about what knowledge is relevant play a key role in the choice of turtle and the design of the turtle world. It is in these considerations that one sees most clearly the influence of the A.I. (and general computerist) styles of thinking about cognitive problems. We confine ourselves here to indicating a few major aspects.

TURTLE GEOMETRY

The geometric knowledge relevant to describing turtle paths has been developed under the name Turtle Geometry, a branch of computational geometry with an emphasis on describing geometric forms by procedures which generate them.

Comparing Turtle Geometry with Euclidean Geometry there is a shift of emphasis from "static," "declarative" knowledge towards "procedural," or "process" knowledge. This remark in no way implies adhesion to any thesis of the form "procedural knowlege is good knowledge;" rather, it opens the door to investigations concerned with where and when procedural forms are appropriate and when not.

INTUITIVE AND FORMAL KNOWLEDGE

Where traditional thinking resorts to vague classifications of knowledge as "intuitive vs. formal" or "tacit vs. explicit," the A.I. conceptual framework allows us to substitute one or more technically precise distinctions such as classification according to the manner in which procedures or other items of a knowledge base are accessible from within the system. Whereas the older distinctions were sophisticated and mysterious, their computational forms are accessible to small children. A fine example arises when a child accustomed to writing programs for straight-line drawings asks how to make the turtle draw a circle. The answer might take the form of the following very general and powerful advice: "PLAY TURTLE. . . WALK IN A CIRCLE. . . DESCRIBE WHAT YOU ARE DOING. . . SAY IT IN TURTLE LANGUAGE. . . WRITE A TURTLE PROCEDURE." Subjects who have had a little experience with this kind of procedure are very likely to

see quickly that when a person walks in a circle he is running a procedure like forward a little and turn a little and keep doing that. This translates into a recursive LOGO procedure:

```
TO CIRCLE
 FORWARD 1
 RIGHT 1
 CIRCLE
END
```

This program will draw a very good circle. It still needs work to make it halt, to allow the size to be variable and so on. But the child is able to make a start by following heuristic advice of a general kind.

KNOWLEDGE ABOUT DEBUGGING

Another powerful concept cluster which is poorly represented in grade school work is associated with bugs and debugging. A plausible contemporary A.I. thesis is that an intelligent program capable of learning must have strong expertise in debugging. If true of computer programs, should chidren be less deserving?

SOCIAL COMPONENTS OF A LEARNING ENVIRONMENT

One does not need A.I. theory or computers to recognize a fifth grade mathematics class as being thoroughly asocial in that there is an exceedingly low degree of communication, collaboration or other interaction about the material being taught.

> The lack of social interaction and the slowness of learning are in fact manifestations of the same underlying phenomena. Good learning requires the learner to have a good internal representation of the changes in his knowledge structures of which the learning is comprised. In simpler terms he needs to be able to "talk to himself" about what he is doing. But most fifth graders are very poor in their ability to talk to themselves or anyone else about mathematical activity. They cannot relate it significantly to other activities and have only the most rudimentary and usually wrong theories about its nature. This intellectual deprivation shows itself in their inability to communicate with peers or with teachers and in their inability to think and learn as effectively as possible.

The example of the turtle-circle program points to a very different possible

relationship between mathematics and the individual's larger knowledge structures. The accumulation of such examples provides a basis for developing a meta-language for talking about the heuristic processes involved in working on a mathematical problem. And the fact that the purpose of it all is to cause something to happen (in that case the turtle to draw a circle) changes the child's relationship to mathematics from the judgmental stance of "right or wrong" to the debugging stance of "how can we make it work" typical of engineers, managers and even real mathematicians.

SYMBOLIC VERSUS NON-SYMBOLIC KNOWLEDGE

The issue of what knowledge can be verbalized is very general. One of the major sources of doubts about the validity of information processing theories of intelligence is the fear that only certain knowledge can be represented in any of the symbolic forms typical (or thought to be typical) of computer programs. In discussions about Education the same issues come up in the form of debates about whether children should be explicitly taught or encouraged to verbalize certain "concepts" or "skills." In popular folklore we have the verse

A CENTIPEDE WAS HAPPY QUITE (Anonymous)

A centipede was happy quite,
Until a frog in fun
Said, "Pray, which leg comes after which?"
This raised her mind to such a pitch,
She lay distracted in the ditch
Considering how to run.

It is worth considering a number of possible theories about the poor centipede. One thing is that her mind was blown by attempting the essentially impossible task of verbalizing an action such as walking (or perhaps of examining her walking program at the same time as executing it). Another theory is that someone else equipped with an appropriate language and degree of time-sharing ability could easily have coped with the situation. Finally there is the theory that she was not in any trouble after all, but in fact blissfuly happy inventing the idea of a higher order programming language so as to formulate a shorter, more comprehensible and mnemonic description of walking than the awful enumeration of legs into which the tricky questioner tried to trap her.

Exploring alternative interpretations in this spirit has led us to put in question many established "facts" (derived from formal experiment or everyday observation) about the impossibility of "telling" how to ride a bicycle. . . or juggle. . . or recognize patterns in IQ tests. What can be told depends (of course!) on what "language" is available. Thus we are led into a new paradigm of experiment for education research. Instead of asking questions like:

> Can such and such be taught by verbal presentation in English to children with a "naturally occurring" state of knowledge?

We ask questions more like

> How can we develop a language which could be acquired by the children so as to enable such and such to be learned through "verbal" formulations?

The conclusions we are drawing are not of the form "yes or no. . . verbal instruction can help in learning this or that. Once the question is properly formulated the answer has to be "yes." The significant conclusions are of a more structured form, for example, how to use various styles of language and the analogs of programming devices to achieve various effects in the learning process.

If one sees "learning" as made up of such processes as "describing," "debugging," and "frame-formation" then all of this points very simply to experiences quite accessible to young children and very close to the heart of the theory of learning.

8 POTPOURRI

This report has gone over considerable ground but nevertheless has not fully covered the spectrum of recent research activities in A.I. at M.I.T. What follows is an enumeration of some of the things omitted because they are incomplete works in progress, because they will be better reported more fully later when a little time has made their implications clearer, or because they will be reported with the work of Project MAC, with whom we work closely.

* The work of SUSSMAN, BROWN, and MCDERMOTT on the problem solving aspects of electronic circuit debugging.

* The work of FREUDER on a region-oriented heterarchical system for recognizing real-world tools.

* The work of GREENBLATT and the work of BAISLEY on the expert problem solving required by chess.

* The work of HOLLERBACH on describing the shape of complicated polyhedra and of curved surface objects.

* The work of DUNLAVEY on how knowledge of different sorts on different levels on be made to interface and work together.

* The work of LAVIN on issues related to how changes in time aid visual analysis.

* The work of MARCUS on parsing.

* The work of MCDONALD and others on Winograd's SHRDLU system.

* The work of NEVINS and the work of GOLDSTEIN on geometry theorem proving.

* The work of RUBIN on medical diagnosis.

* The work of TAENZER on circuit board inspection.

* The work of INOUE on mechanical assembly.

* The work of WOODHAM on visual feedback.

* The work of WHITE and his colleagues in Project MAC on LISP development.

* The work of MARTIN and his students in Project MAC on Automatic Programming which we feel has made important contributions to A.I. theory even though the focus is applied.

* The work of HEWITT and his students in Project MAC on the ACTORS programming language concept.

* The work of MOSES in Project MAC on the MACSYMA system for applied mathematics, which is not, properly speaking, concerned much with A.I., but which is nontheless of great interest to people in the laboratory since it was an early spin-off project.

* The work of GREENBLATT, KNIGHT, and others on the development of the now germinating LISP-machine concept.

* The work of the entire system development staff in creating the truly first class computing resource so central to our progress in artificial intelligence.

APPENDIX I: STAFF

Acting Director: Patrick H. Winston

Co-Directors: Marvin Minsky
Seymour Papert

Faculty:
Harold Abelson
Jeanne Bamberger
Ira Goldstein
Carl Hewitt
Berthold Horn
Marvin Minsky
Seymour Papert
Vaughan Pratt
Gerald Sussman
Patrick Winston

Staff:
Michael Beeler*
Meyer Billmers
Gerald Brown
Rosanne Brown
Thomas Callahan
Joseph Cohen
Frederick Drenckhahn
Donald Eastlake*
Tim Finin
William Freeman
James Geiser*
Cheryl Goodman
Richard Greenblatt
John Holloway
Suzin Jabari
Pitts Jarvis
Don Johnstone
Eva Kampits
Barbara Kohl
Thomas Knight
Noble Larson
Ron Lebel
David Marr
Penny Minsker*
Arthur Nevins*
Russell Noftsker
Kyoko Okumura*
Gordon Oro
Allison Platt*
Bruce Roberts
David Silver
Cynthia Solomon
Ray Solomonoff*
Richard Stallman
Carl Waldrop
George Wallace
Wade Williams*

Visitors:
Hirochika Inoue
Erik Sandewall

(* --> No longer with the laboratory)

Graduate Students:

Howard Austin
James Baratz
Allen Brown
Richard Brown
Candice Bullwinkle
Doug Cannon
Greg Clemenson
Byron Davies
Johan deKleer
Andrea diSessa
Michael Druke
Michael Dunlavey
Scott Fahlman
Michael Freiling
Eugene Freuder
Jan Galkowski
Nat Goodman
Richard Grossman
John Hollerbach
Paul Hudak
Fred Kern
Ben Kuipers
Mitch Marcus
Mark Lavin
Drew McDermott
David McDonald
Mark Miller
Robert Moore
Tomas Lozano
Radia Perlman
Steve Purcell
Charles Rich
Andee Rubin*
David Tanzer
Shimon Ullman
Richard Waters
Robert Woodham
Aki Yonezawa

Undergraduate Students:

James Adams
Harry Bochner
Bruce Edwards
Jim Evans
Ken Forbus
Fred Hanna
Sandra Kelly
Craig Latham
Henry Lieberman
Paul Mailman
David Outzs
Robert Young

APPENDIX II: PUBLISHED PAPERS

ABELSON, H.

"Topologically Distinct Conjugate Varieties with Finite Fundamental Group", Topology, June 1974.

BAMBERGER, J.

The Art of Listening, with H. Brofsky, Harper and Row, 1975 (3rd edition).

"Musical Significance of Beethoven's Original Fingerings", Music Symposium, 1974.

"What's In A Tune", in The Arts and Cognition, in press 1974.

"The Luxury of Necessity", Symposium on Music Consumers, National Association of Schools of Music, 1974.

"Learning to Think Musically", Music Educators Journal, April 1973.

FAHLMAN, S.

"A Planning System for Robot Construction Tasks", Artificial Intelligence 5, Spring 1974.

GEISER, J.

"Formalization of Yesserin Volpin's Proof of Theoretical Studies by Means of Non-Standard Analysis", Journal of Symbolic Logic, Vol. 39, March 1974.

"Decidability of a Fragment of Classical First Order Logic", Research Report, Boston University, Department of Mathematics, April 1973.

"E-Semantics and Recursive Approximations to Theories", Recursive Function Newsletter, Supplement No. 1, University of California, Berkeley, Ca., February 1973.

GOLDSTEIN, I.

"Planning Paradigms - Knowledge for Organizing Models into Programs", Proceedings of Nato Advanced Study Institute on Computer Oriented Learning Processes, August 1974.

"Understanding Simple Picture Programs", Proceedings of Sussex Conference on Artificial Intelligence and Simulation of Behavior, July 1974.

GREIF, I. "A More Mechanical Heuristic Approach to Program

Verification", with R. Waldinger, Proceedings of Institut de Programmation's Colloque Sur La Programmation, April 1974.

HORN, B.K.P.

"Determining Shape from Shading", in The Psychology of Computer Vision, editor Patrick Winston, in press.

"The Determination of Lightness from an Image", Computer Graphics and Image Processing, to be published.

"The Legacy of a Poor Formulism - a tree is not the key", SYSTEMS - The Bulletin of the Computer Society of South-Africa, Vol. 3, No. 5, September/October 1973.

"A New Machine Organization - something to look forward to", SYSTEMS - The Bulletin of the Computer Society of South-Africa, Vol. 3, No. 4, July/August 1973.

"Lymphocytic Infiltration in Bladder Cancer", with R.A. Leigh and P.J.P. van Blerk, South African Medical Journal, February 1973.

INOUE, H.

"Articulated Manipulators and Their Control Software, with K. Sato, K. Okamoto, and K. Takase, Proceedings of International Symposium on Robots, Manipulators and Systems, September 1973.

KUIPERS, B.

"An Hypothesis-Based Recognition System for the Blocks World", Proceedings of Second International Joint Conference on Pattern Recognition, August 1974.

LOZANO, T.

"Attribute Based File Organization in a Paged Memory Environment", with J. Rothnie, Communications of the ACM, Vol. 17, No. 2, February 1974.

MARR, D.

"Computing Lightness in the Primate Retina", Vision Research, to be published.

MCDERMOTT, D.

"From PLANNER to CONNIVER - A Genetic Approach", Proceedings of FJCC (AFIPS), December 1972.

MILLER, M.

"Reasoning from Incomplete Knowledge", with A. Collins, E.

Warnock, and N. Aiello, Proceedings of Carbonell Symposium, 1974.

MINSKY,M.

Artificial Intelligence, Condon Lectures, Oregon State System of Higher Education,with Seymour Papert, 1974.

NEVINS,A.

"Plane Geometry Theorem Proving Using Forward Chaining", Artificial Intelligence, to be published.

"A Relaxation Approach to Splitting in an Automatic Theorem Prover", Artificial Intelligence, to be published.

"A Human Oriented Logic for Automatic Theorem Proving", Journal of Association of Computing Machinery, to be published.

PAPERT,S.

Artificial Intelligence, Condon Lectures, Oregon State System of Higher Education, with Marvin Minsky, 1974.

Lecture Notes in Process Models for Psychology, Rotterdam University Press, Rotterdam, The Netherlands, 1973. (Lectures at a NUFFIC International Summer Course, 1972.)

PRATT,V.

"A Linguistics Oriented Programming Language", Proceedings of IJCAI III, August 1973.

SUSSMAN,G.

"Virtuous Nature of Bugs", Proceedings of Sussex Conference on Artificial Intelligence and Simulation of Behavior, July 1974.

"From PLANNER to CONNIVER - A Genetic Approach", with D. McDermott, Proceedings of FJCC (AFIPS), December 1972.

WINSTON,P.

"Learning Structural Descriptions from Examples," in The Psychology of Computer Vision, edited by Patrick H. Winston, McGraw-Hill Publishing Company, 1974.

"Learning, Problem Solving, and The Search for Intelligence," in Computers and Thought II, edited by E. Feigenbaum, in publication.

"Using 3-Dimensional Models in Scene Analysis," Proceedings of the Sixth Annual Princeton Conference on Information Science, 1972.

"The M.I.T. Robot -- A Case Study in Artificial Intelligence

Research," in Machine Intelligence 7, Edinburgh University Press, 1972.

"Learning Structural Descriptions," in Contemporary Issues in Cognitive Psychology, edited by Robert Solso, Loyola Symposium on Cognitive Psychology, 1972.

"Scene Understanding Systems," in Frontiers of Pattern Recognition, edited by Watenabe, Prentice-Hall Publishing Company, 1971.

APPENDIX III: TALKS and LECTURES

ABELSON, H.

Princeton, N.J., April 1973: "More Non-homeomorphic Conjugate Varieties", Princeton University Topology Colloquium.

BAMBERGER, J.

Cambridge, Ma., July 1974: "Intuitive and Formal Models of Representi Music", Harvard University Graduate School of Education.

Vail, Colorado, June 1974: "Cognitive Aspects of Musical Perception", Carnegie Cognition Conference.

Houston, Texas, February 1974: "The Luxury of Necessity", National Association of Schools of Music, Symposium on Music Consumers.

Rochester, N.Y., November 1973: "Is Music Dead", Eastman School of Music.

DUNLAVEY, M.

Boston, Ma., October 1973: "The Procedural Paradigm of Learning", Boston State College Evening Division.

FAHLMAN, S.

Lake Tahoe, Ca., June 1974: "The BUILD Planning System", Stanford Research Institute Workshop on Interactive Knowledge Systems.

GEISER, J.

Cambridge, Ma., May 1974: "Noema and Frames", Joint Seminar with Isaac Miller of M.I.T. Philosophy Dept.

GOLDSTEIN, I.

Paris, France, August 1974: "Planning Paradigms - Knowledge for Organizing Models into Programs", Nato Advanced Study Institute on Computer Oriented Learning Processes.

Brighton, England, July 1974: "Understanding Simple Picture Programs", Sussex Conference on Artificial Intelligence and Simulation of Behavior.

Palo Alto, Ca., June 1974: "Debugging Simple LOGO Programs", Xerox Palo Alto Research Center.

Lake Tahoe, Ca., June 1974: "Summary of MYCROFT", Stanford Research Institute Workshop on Interactive Knowledge Systems.

Troy, N.Y., November 1973: "Frontiers of Artificial Intelligence", Rensselaer Polytechnic Institute, Department of Mathematics.

HEWITT,C.

Boston, Ma., May 1973: "A Modular ACTOR Formalism", with P. Bishop and R. Steiger, ACM Greater Boston SIGPLAN Meeting.

Savannah, Georgia, April 1973: "The Democratic Ethos or How a Society of Noncoercable ACTORS can be Incorporated into a Structured System", with P. Bishop and R. Steiger, SIGPLAN-SIGOPS Interface Meeting.

MARR, D.

Warwick,England, May 1974: "Symbolic Computation in Perception and in Artificial Intelligence" and "Why Catastrophe Theory is Irrelevant to Developemnt, and to the Brain", University of Warwick, Conference on Topological Models in Biology.

Erice, Sicily, May 1974: "Computing Lightness within the Retina", "From Simple Cells to Simple Symbols", "Information Storage in Neural Hardware", International School of Biophysics.

Chicago, Ill., May 1974: "Computing Lightness within the Retina", University of Chicago.

London, Ontario, April 1974: "Computational Issues in Writing Motor Programs", University of Ontario.

Boston, Ma., April 1974: "Lightness and the Retina", Boston University

Amherst, Ma., April 1974: "Low Level Vision", University of Massachusetts.

Buffalo, N.Y., April 1974: "Lightness and the Retina" and "Perspectives on Neurophysiology and Artificial Intelligence", S.U.N.Y. at Buffalo.

Pasadena, Ca., May 1973: "Some Biological Computer Architecture", California Institute of Technology.

Berkeley, Ca., May 1973: "A Theory of Cerebellar Cortex" and "The Hippocampus", University of California at Berkeley, Department of Physiology-Anatomy.

Iowa City, Iowa, May 1973: "Theories of Local Cortical Function", University of Iowa, Division of Neurobiology.

Cambridge, Ma., April 1973: "A Theory of the Cerebellum", MIT course 6.544 (Professors Minsky and Papert).

Cambridge, Ma., March 1973: "Some Biological Computer Architecture", MIT Artificial Intelligence Seminar.

MCDERMOTT, D.

Pittsburgh, Pa., July 1973: "Assimilation of New Information by a Natural Language-Understanding System", Carnegie-Mellon University Workshop on Information Processing and Psychology.

MINSKY,M.

Boston, Mass., August 1974: Participant in Discussion of the Foundations of Mathematics Workshop on the Historical Development of Mathematics, American Academy of Arts and Sciences.

Philadelphia, Pa., April 1974: Discussant and Chairman, Session on Memory and Extendable Knowledge Structures, 5th Annual Interdisciplinary Conference on Structural Learning, University of Pennsylvania.

Holmdel, N.J., March 1974: "A Review of Recent Work on Artifical Intelligence", Bell Telephone Laboratories Research Colloquium.

New York, N.Y., November 1973: "Artificial Intelligence...on how the new machines are made", 11th Annual New Horizons in Science Seminar, Council for the Advancement of Science Writing, Inc.

Cambridge, Ma., November 1973: "Artificial Intelligence", M.I.T. Graduate Economics Association Seminar, Sloan School, M.I.T.

Washington, D.C., October 1973: "The Computer Revolution", Perspectives in Science and Technology Symposium, Naval Research Laboratory, 50th Anniversary Symposium.

Stanford, Ca., August 1973: Chairman, Robot Implementations Session, Third International Joint Conference on Artificial Intelligence, Stanford University.

Washington, D.C., April 1973: "Automation and Robotics", National Science Fondation Meeting.

Menlo Park, Ca., March 1973: "A.I. and Educational Technology", Stanford Research Institute.

Berkeley, Ca., March 1973: "Theories of Imagery", Institute of Human Learning.

Berkeley, Ca., March 1973: "A.I. at M.I.T.", University of California at Berkeley, Mathematics Department.

Austin, Texas, February 1973: 3 talks on A.I., University of Texas.

Cambridge,Ma., February 1973: "Modern Theories of Intelligence", M.I.T. Alumnae Association.

Buffalo, N.Y., January 1973: 3 lectures on A.I. and Psychology, University of Buffalo, Cognitive Psychology Department.

New Haven, Conn., January 1973: "A.I. Research at M.I.T.", ACM Chapter.

Cambridge, Ma., November 1972: "Vision and Robotics", M.I.T. Industrial Liason Symposium.

Pittsburgh, Pa., October 1972: "A.I. and Computer Science", Carnegie-Mellon University.

Cambridge, Ma., September 1972: "A.I., Learning, and Knowledge", M.I.T. Physics COlloquium.

Pasadena, Ca., September 1972: "A.I. Research at M.I.T.", Jet Propulsion Laboratory.

San Jose, Ca., August 1972: "A.I. Manipulators and Inniative Spacecraft", Ames NASA Laboratory.

Boulder, Colorado, July 1972: "A.I. and Cognitive Psychology", University of Colorado.

Princeton, N.J., June 1972: Discussant at Princeton Institute for Advanced Study.

Washington, D.C., June 1972: "Artificial Intelligence Research at M.I.T.", Office of Naval Research, (Published in Naval Research Reviews).

MOORE, R.C.

Stanford, Ca., August 1973: "D-SCRIPT: A Computational Theory of Descriptions", IJCAI III.

NEVINS,A

Kingston, R.I., April 1974: "Automatic Theorem Proving by Natural Deduction", University of Rhode Island, Department of Computer Science.

Providence, R.I., March 1974: "Plane Geometry Theorem Proving",Brown University, Department of Applied Mathematics.

Austin Texas, January 1973: "Some Current Trends in Automatic Theorem Proving", University of Texas, Department of Mathematics and Computer Science.

PAPERT,S.

Geneva, Switzerland, June 1974: Participant in Symposium of the Centre d'Epistemologie Genetique at the University of Geneva.

Greenville, South Carolina, June 1974: "Computerized Educational Devices", National Meeting of College and University Eleven Thirty Users Group, Furman University.

Cambridge, Ma., June 1974: "A New Concept in Education", M.I.T. Alumni Day Speaker, M.I.T.

New York, N.Y., April 1974: "Artificial Intelligence and the Philosophy of Education", Fordham University, Division of Science and Mathematics.

Salt Lake City, Utah, March 1974: "Building a Modern Substitute for the Elementary School", University of Utah, Communications Lecture Series.

Durham, N.H., March 1974: "Recent Work in Artificial Intelligence", University of New Hampshire, Junior Science and Humanities Symposium.

Newark, Del., March 1974: "Artificial Intelligence", University of Delaware.

San Francisco, Ca., February 1974: "Artificial Intelligence and Its Education Applications", AAAS Meeting.

Rochester, N.Y., December 1973: "Artificial Intelligence and Education", M.I.T. Club of Rochester.

Berkeley, Ca., October 1973: "Fysics in the Phingertips, a Deeper than Usual Reconceptualization of Physics for Education", SESAME Colloquium, University of California, Berkeley.

Berkeley, Ca., October 1973: "Computers and the Mind", Foerster Lectureship Symposium, University of California, Berkeley.

Hamburg, Germany, October 1973: Main Lecturer on Artificial Intelligence, 3rd Annual Gesellschaft fur Informatik Conference, University of Hamburg.

Pittsburgh, Penn., September 1973: "Artificial Intelligence as a New Kind of Theory of Education", Fall Lecture Series, Immigration Course in Computer Science, Carnegie-Mellon University.

Pajaro Dunes, Watsonville, Ca., September 1973: Conducted Workshops on How Computers Might Effect Education More Deeply, Joint M.I.T. and Xerox Learning Center Workshops.

Stanford, Ca., August 1973: "Artificial Intelligence and Education", 3rd International Joint Conference on Artificial intelligence, Stanford University.

Quebec City, Canada, August 1973: "On the Possibilities of Much Deeper Revolutions in Math Education", Laval University's International Conference of C.I.E.A.E.M.

Hanover, N.H., July 1973: Lecture Demonstration on New Developments in LOGO, Dartmouth College, Kiewit Computation Center.

Snowmass, Colorado, June 1973: "Direction for Curriculum Reform", Snowmass Conference on the K-12 Mathematics Curriculum sponsored by the National Science Foundation.

New York, N.Y., May 1973: "The Computer and the Beanbag", Conference on the Imaginative Uses of the Computer in Education", Graduate Center of the City University of New York.

East Lansing, Michigan, April 1973: "Teaching Children to be Mathematicians", Michigan State University, Department of Mathematics.

Waltham, Ma., April 1973: "A.I. and the Philosophy of Education", BIT Meeting sponsored by Mathematics Teachers of New England.

Philadelphia, Pa., April 1973: "A.I. and the Philosophy of Education", Jean Piaget Society and Structural Learning Conference, University of Pennsylvania.

Cambridge, Ma., April 1973: "A.I. and Human Education", Center for Advanced Engineering Study, M.I.T.

Montreal, Canada, March 1973: "A.I. Research at M.I.T.", Montpetit College.

Atlantic City, N.J., February 1973: "Teaching Thinking -- Computers and Kids", American Association of School Administrators.

Brighton, England, February 1973: "A.I. and Education", The University of Sussex, Laboratory for Experimental Psychology.

Edmonton, Canada, January 1973: "A.I. as a New Theory of Education", The University of Alberta, Department of Computing Science.

Calgary, Alberta, Canada, January 1973: "A.I. as New Theory of Education", The University of Calgary, Department of Mathematics, Statistics and Computer Science.

Vancouver, Canada, January 1973: "A.I. as a New Theory of Education", The University of British Columbia, Department of Computer Science.

Lake Arrowhead, Ca., December 1972: Participant, Symposium on

Research in Education, University of California Conference Center.

SUSSMAN, G.

Brighton, England, July 1974: "Virtuous Nature of Bugs", Sussex Conference on Artificial Intelligence and Simulation of Behavior.

Lake Tahoe, Ca., June 1974: "Strategies for the Qualitative Design and Repair of Electronic Circuits", Stanford Research Institute Workshop on Interactive Knowledge Systems.

New Haven, Conn., April 1974: "Bugs: A Study in Causal and Teleological Reasoning", Yale University, Department of Computer Science.

Ann Arbor, Michigan, October 1973: "A Computational Model of Skill Acquisition", University of Michigan Computer and Communication Science Colloquium.

Cambridge, Ma., June 1973: "Automatic Programming", MIT Sloan School, Seminar on Knowledge-Based Systems for Operations Management.

Pajaro Dunes, Watsonville, Ca., June 1973: "HACKER", Joint M.I.T. and Xerox Learning Center Workshops.

New York, N.Y., February 1973: "A Problem Solver that Improves with Practice", Courant Institute (NYU).

Anneheim, Ca., December 1972: "From PLANNER to CONNIVER - A Genetic Approach", FJCC (AFIPS) Conference.

WALTZ,D.

Urbana, Ill., March 1973: "Current Research at the MIT A.I. Lab", University of Illinois, Coordinated Science Laboratory.

WINSTON,P.

Chicago, Illinois, October 1974: "Tutorial on Artificial Intelligence," National Electronics Conference.

Rochester, New York, October 1974: "New Progress in Artificial Intelligence," University of Rochester.

Canberra, Australia, May 1974: "Methodologies in Artificial Intelligence," symposium, "Objectives and Methodologies in Artificial Intelligence," Research School of Physical Sciences, Australian National University.

Worcester, Masssachusetts, April 1974: "Computers and the Future," panel discussion, Worcester State College.

Stanford, California, August 1973: "Learning, Problem Solving, and the Search for Intelligence," Computers and Thought Lecture, Third International Artificial Intelligence Conference.

Moscow, USSR, August 1973: "The Winston Learning Paradigm," Institute for Communication.

Moscow, USSR, August 1973: "Artificial Intelligence and the Waltz Theory," Academy of Science.

Tokyo, Japan, July 1973: "Progress in Vision Research," Electrotechnical Laboratory.

Kyoto, Japan, July 1973: "Progress in Vision Research," Japan-U.S. N.S.F. Conference.

Pasadena, California, April 1973: "The Waltz Approach to Scene Analysis," Computer Science Colloquium, California Institute of Technology.

La Jolla, California, April 1973: "Artificial Intelligence and Cognitive Psychology," Cognitive Psychology Colloquium, University of California.

Stanford,California, April 1973: "The M.I.T. Approach to Artificial Intelligence," Guest lecture, graduate course in Artificial Intelligence, Stanford University.

Stanford, California, April 1973: "The Waltz Effect," Computer Science colloquium, Stanford University.

Champaign-Urbana, Illinois, March 1973: "New Ideas in Vision Processing," Computer Science Colloquium, University of Illinois.

Warwick, Rhode Island, March 1973: "Artificial Intelligence," Joint Meeting of the Data Processing Management Association and the Association for System Management.

Tokyo, Japan, January 1973: "Machine Vision," Public lecture sponsored by Japan Electronic Industry Association.

Kyoto, Japan, January 1973: "Picture Processing in Artifcial Intelligence," Computer Science Colloquium, Kyoto University.

Osaka, Japan, January 1973: "Visusal Scene Analysis", Computer Science Colloquium, Osaka University.

Tokyo, Japan, January 1973: "Visual Scene Analysis," Computer Science Colloquium, Tokyo University.

Osaka, Japan, January 1973: "Artificial Intelligence Today," Mitsubishi Research Center.

Tokyo, Japan, January 1973: "Analyzing Visual Information," Hitachi Central Laboratories.

Tokyo, Japan, January 1973: "Advanced Programming Languages for Artificial Intelligence," Toshiba Research and Development Center.

Tokyo, Japan, January 1973: "Six Lectures on Artificial Intelligence," Delivered to 100 staff scientists, Electrical Technical Laboratory.

Cambridge, Massachusetts, November 1972: "Industrial Applications for Seeing Machines," Industrial Liason Office Colloquium, M.I.T.

Boston, Massachusetts, November 1972: "Machines that See and Understand," NEREM.

Edinburgh, Scotland, June 1972: "The M.I.T. Robot," Machine Intelligence Conference.

London, Canada, 1972: "Trends in Vision Research," Computer Science Colloquium, Also consulted with the computer science visiting committee about the advisability of establishing an Artificial Intelligence effort, University of Western Ontario.

Chicago, Illinois, 1972: Talk delivered to the Loyola Symposium on Cognitive Psychology. Subject was the use of artificial intelligence tools in psychological research, Loyola University.

Cambridge, Massachusetts, 1972: Talk on Artificial Intelligence and robotics research for visiting heads of electrical engineering departments, M.I.T.

Princeton, New Jersey, 1972: Participant at the Sixth Annual Princeton Conference on Information Sciences and Systems. Talk given on the subject of using three-dimensional models in scene analysis, Princeton University.

Austin, Texas, 1972: Department of Computer Science Colloquium on the use of high level knowledge in vision systems, University of Texas.

M.I.T. A.I. BIBLIOGRAPHY

*1-7 Concerned with early LISP development (see A.I. Memo 50).

*8 Recursive Functions of Symbolic Expressions and their Computation by Machine, John McCarthy, published in: Computer Programming and Formal Systems, North-Holland Publishing Co., Amsterdam, 1963.

*9-14 Concerned with early LISP development (see A.I. Memo 50).

*15 SML-Examples of Proofs by Recursion Induction, John McCarthy.

*16 A Question-Answering Routine, A.V. Phillips.

*17 Programs with Common Sense, John McCarthy. Published in Proc. Math. of Thought Processes, HMSO, 1958. Reprinted in Semantic Information Processing, Minsky (Ed.), M.I.T. Press, 1968.

*18 Some Results from a Pattern Recognition Program Using LISP, Louis Hodes.

*19 Concerned with early LISP development (see A.I. Memo 50).

*20 Puzzle Solving Program in LISP, John McCarthy.

*21 The Proofchecker, Paul Abrahams; see complete version in Mathematical Algorithms, April 1966, Vol. 1, No. 2; July 1966, Vol. 1, No. 3.

*22-26 Concerned with early LISP development (see A.I. Memo 50).

*27 Simplify, Tim Hart.

*28 Concerned with early LISP development (see A.I. Memo 50).

29 Introduction to the Calculus of Knowledge, Bertram Raphael, November 1961. ($.80)

*30 The Tree Prune (TP) Algorithm, Dec. 1961, Tim Hart & Daniel Edwards, Revised Oct. 1963 - The alpha-beta Heuristic. The subject is discussed in Slagle, J. and J.N. Dixon: JACM, Vol. 16, No. 2, April 1969, pp. 189-207.

*31 A Basis for a Mathematical Theory of Computation, John McCarthy, Jan. 1962. Published in: Western Joint Computer Conference, 1961.

32 On Efficient Ways of Evaluating Certain Recursive Functions, John McCarthy. ($.80)

*33 Universality of (p=2) Tag Systems and a 4 Symbol 7 State Universal Turing Machines, Marvin Minsky; a better version is in Chapter 14 of Computation, M. Minsky, Prentice Hall, 1967.

*34 A New Eval Function, John McCarthy.

*35 Concerned with early LISP development (see A.I. Memo 50).

36 On the Problem of the Effective Definition of Random Sequence, Michael Levin, Marvin Minsky, Roland Silver. ($.80)

*37 Some Identities Concerning the Function subst [x;y;z], Lewis Norton.

*38 Machine Understanding of Linguistic Information - A Survey and Proposal, Bertram Raphael. Reprinted in Semantic Information Processing, Marvin Minsky (Ed.), M.I.T. Press, 1968.

*39 Concerned with early LISP development (see A.I. Memo 50).

*40 A Note on the Feasibility of Application of the Davis Putnam Proof Procedure to Elementary Number Theory, Donald Dawson.

41 A Chess Playing Program, Alan Kotok. ($.80)

*42 Proposed Research on Learning, Marvin Minsky.

*43 Proposal for a General Learning Machine, Bertram Raphael. Reprinted in Semantic Information Processing, M. Minsky (Ed.), M.I.T. Press, 1968.

*44 A Simple Direct Proof of Post's Normal Form Theorem, Marvin Minsky. A better version is in: Chapter 13 of Computation, M. Minsky, Prentice Hall, 1967.

*45 A Question-Answerer for Algebra Word Problems, Daniel Bobrow. Final form in Semantic Information Processing, M. Minsky (Ed.), M.I.T. Press, 1968.

*46 A Heuristic Program to Solve Geometric Analogy Problems, T.G. Evans, Oct. 1962. Final form in: Semantic Information Processing, Minsky (Ed.), M.I.T. Press, 1968.

*47 A Proposal to Investigate the Application of a Heuristic Theory of Tree-searching to a Chess-playing Program, Burton Bloom, February 1963.

48 Neural Nets and Theories of Memory, Marvin Minsky, March 1963. ($.80)

*49 Computer Representation of Semantic Information, Bertram Raphael, April 1963. Reprinted in: Semantic Information Processing, Minsky (Ed.), M.I.T. Press, 1968.

*50 This is concerned with early LISP development. More information is in: The LISP 1.5 Programming Manual, M.I.T. Press, Cambridge, Mass.; The Programming Language LISP, Berkeley Enterprises, Newton, Mass.; LISP 1.5 Primer, Clark Weissman, Prentice Hall, Englewood Cliffs, N.J.

*51 METEOR: A LISP Interpreter for String Transformation, Daniel Bobrow, 1963.

*52 Universality of Tag Systems with p=2, John Cocke & Marvin Minsky, April, 1963; better version is in Chapter 14 of Computation, Minsky, Prentice Hall, 1967.

*53 ARGUS: Real-Time handwritten character-recognition system, Warren Teitelman, May 1963.

*54 Proposal for a FAP Language Debugging Program, Joel Winett, June, 1963.

*55 Primitive Recursion, Michael Levin, July, 1963.

*56 A Proposal for a Geometry Theorem Proving Program, Tim Hart, September 1963.

*57 MACRO Definitions for LISP, Tim Hart, October, 1963.

58 A LISP Garbage Collector Algorithm Using Serial Secondary Storage, Marvin Minsky, October, 1963 (MAC-M-129). Revised. ($.80)

*59 Operation of a Semantic Question-Answering System, Bertram Raphael, November 1963. Results in Chapter 2 of Semantic Information Processing, Minsky (Ed.), M.I.T. Press, 1968.

*60 Recent Improvements in DDT, D. Edwards, M. Minsky, November 1963.

*61 Mathscope: Part I: A Proposal for a Mathematical Manipulation-Display System, Marvin Minsky, November 1963. More advanced results in Ph.D. thesis of William Martin MAC-TR-36, January 1967. Available through the Defense Supply Agency DDC No. AD-657-283.

*62 Derivator I. A Program for Visual Inspection of Solutions to First-Order Non-Linear Differential Equations, Marvin Minsky, Dec. 1963, MAC-M-124.

63 Secondary Storage in LISP, Daniel Edwards, Dec. 1963, MAC-M-128. ($1.20)

*64 LISP Exercises, Tim Hart, Michael Levin, Jan. 1963 MAC-M-134. Reprinted in: The Programming Language LISP, Berkeley Ent., Newton, Mass.

65 The Graphical Typewriter, A Versatile Remote Console Idea, MAC-M-135, Marvin Minsky. ($.80)

*66 Natural Language Input for a Computer Problem Solving System, Daniel Bobrow, MAC-M-148. Reprinted in Chapter 3 of Semantic Information Processing, Minsky (Ed.), M.I.T. Press, 1968.

*67 Revised User's Version, Time Sharing LISP for CTSS, William Martin, Tim Hart. MAC-M-153.

*68 Syntax of the New Language, Michael Levin, May, 1963, MAC-M-158.

*69 New Language Storage Conventions, Michael Levin, May 1964, MAC-M-159.

*70 Hash-Coding Functions of a Complex Variable, William Martin, June 1964.

Included in his Dissertation: Symbolic Mathematical Laboratory, MAC-TR-36. Available from Defense Supply Agency DDC No. AD-657-283.

*71 String Manipulation in the New Language, Daniel Bobrow, July 1964, MAC-M-176.

*72 Proposed Instructions on the GE 635 for List Processing and Push Down Stacks, Michael Levin Sept. 1964, MAC-M-183.

*73 Unrecognizable Sets of Numbers, Marvin Minsky, Seymour Papert, Nov. 1964. Revised Sept. 1965. Published in: JACM, Vol. 13, No. 2, April, 1966.

*74 CTSS LISP Notice-Supplement to A.I. Memo No. 67, T. Hart, Dec. 1964, MAC-M-206.

*75 Television Camera-to-Computer Adapters: PDP-6 Device 770, Marvin Minsky, Jan. 1965, MAC-M-215.

*76 The COMIT Feature in LISP II, Daniel Bobrow, Feb. 1965, MAC-M-219.

77 Matter, Mind and Models, Marvin Minsky, March 1965. Published in 1965 IFIP Congress. Also Ch. 9 of Semantic Information Processing, Minsky, 1968. ($.80)

*78 Topics in Model Theory, Michael Levin, May 1965, MAC-M-240.

*79 PDP-6 LISP Input-Output for the Dataphone, William Martin, June 1965, MAC-M-241.

*80 PDP-6 LISP Input-Output for the Display, William Martin, June 1965, MaC-M-242.

81 PDP-6 TECO, Peter Samson, July 1965, MAC-M-250.

*82 MAC PDP-6 DECtape File Structure, Peter Samson, July 1965, MAC-M-249.

*83 Use of MACOMP, Peter Samson, July 1965, MAC-M-248. Obsolete, replaced by A.I. Memo 118.

*84 EDIT and BREAK Functions for LISP, Warren Teitelman, MAC-M-264.

*85 Syntax and Display of Mathematical Expressions, William Martin, July 1965. See his Dissertation: Symbolic Mathematical Laboratory, MAC-TR-36. Available through Defense Supply Agency DDC No. AD-657-283.

86 Design of the Hand, Marvin Minsky, August 1965, MAC-M-258. ($.80)

*87 FLIP - A Format List Processor, Warren Teitelman, MAC-M-263. Replaced by BBN Report No. 10, 15 July 1967. (50 Moulton St., Cambridge, Mass.)

*88 MACTAP, A PDP-6 DECtape Handling Package, Peter Samson, Sept. 1965, MAC-M-267.

*89 A Theory of Computer Instructions, Ward Douglas Maurer, Sept. 1965. See Chapter 12 of Programming: An Introduction to Computer Languages and Techniques, Maurer, Holden-Day Inc., San Francisco.

89A Computer Experiments in Finite Algebra, W.D. Maurer, June 1965. ($.80)

89B Computer Experiments in Finite Algebra II, W.D. Maurer, Dec. 1965. ($1.20)

90 MIDAS, Peter Samson, Oct. 1965. Revised Dec. 1967. Revised Oct. 1968, MAC-M-279. ($1.60)

*91 A Useful Algebraic Property of Robinson's Unification Algorithm, Tim Hart, Nov. 1965, MAC-M-285.

*92 Topics in Model Theory, Michael Levin, Jan. 1966, MAC-M-294.

*93 A New Version of CTSS LISP, Joel Moses, Robert Fenichel, MAC-M-296.

*94 A New Machine-Learning Technique Applied to the Game of Checkers, Arnold Griffith, March 1966, MAC-M-299.

*95 A Program Feature for CONVERT, Adolfo Guzman, Harold McIntosh, April 1966, MAC-M-305.

*96 POLYBRICK: Adventures in the Domain of Parallelepipeds, Adolfo Guzman, May 1966. See his M.S. thesis: MAC-TR-37 available through the Defense Supply Agency DDC No. AD-656-041.

*97 Symbolic Integration, Joel Moses, June 1966.

*97A Symbolic Integration, II, Joel Moses, Oct. 1966. See his Dissertation Symbolic Integration MAC-TR-47, available through Defense Supply Agency DDC No. AD-662-666.

*98 PDP-6 LISP, Peter Samson, June 1966, MAC-M-313. See A.I. Memo 116A.

99 CONVERT, Adolfo Guzman, Harold McIntosh, June 1966, MAC-M-316. Published in COMM, ACM 9, 8, Aug. 1966, pp. 604-615. ($1.60)

*100 The Summer Vision Project, Seymour Papert, July 1966.

*101 SIDES 21, Richard Greenblatt, Jack Holloway; described by Donald Sordillo, August 1966, MAC-M-320.

*102 A Quick Look at Some of Our Programs, Gerald Sussman, Adolfo Guzman, July 1966 .

*103 Additions to the Vision Library, John White, Aug. 1966.

*104 Output to the PDP-6 Calcomp Plotter, Jack Holloway, Aug. 1966.

*105 Modifications to PDP-6 Teletype Logic, Tom Knight, Aug. 1966.

*106 An Input Macro for TECO, Donald Eastlake, Sept. 1966, MAC-M-324.

*107 Music Playing on the PDP-6, Donald Sordillo, Aug. 1966, MAC-M-326.

*108 A Primitive Control P Feature, Donald Eastlake, Oct. 1966, MAC-M-329.

*109 SCPLOT BIN, Donald Sordillo, Oct. 1966, MAC-M-333 (see A.I.Memo 112)

*110 Figure Boundary Description Routings for the PDP-6 Vision Project, John White, Sept. 1966, MAC-M-328.

*111 Summer Vision Programs, Leslie Lamport, Oct. 1966, MAC-M-332.

*112 CHAR PLOT, Donald Sordillo, Oct. 1966, MAC-M-334. (Replaced by A.I. Memo 125).

*113 A Description of the CNTOUR Program, Larry Krakauer, Nov. 1966, MAC-M-335.

*113A A Description of the CNTOUR Program, Revised, Larry Krakauer.

*114 A Step by Step Computer Solution of Three Problems in Non-Numerical Analysis, William Martin, July 1966. See his Dissertation Symbolic Mathematical Laboratory MAC-TR-36, available from Defense Supply Agency DDC No. AD-657-283.

*115 Program Memo, Peter Samson, Dec. 1966.

116 PDP-6 LISP (LISP 1.6), January 1967, Revised April 1967. ($1.20)

This is a mosaic description of PDP-6 LISP, intended for readers familiar with the LISP 1.5 Programmer's Manual or who have used LISP on some other computer. Many of the features, such as the display, are subject to change. Thus, consult a PDP-6 systems programmer for any differences which may exist between LISP of Oct. 14, 1966 and present LISP on the system tape.

*117 Memo, Russell Noftsker.

118 PDP-6 Software Update, Thomas Knight, Jan. 1967; Revised June 1967, Donald Eastlake (System Program). ($.80)

*119 A Primitive Recognizer of Figures in a Scene, Adolfo Guzman, Jan. 1967; see his Dissertation: Some Aspects of Pattern Recognition by Computer, MAC-TR-37. Available through Defense Supply Agency DDC No. AD-656-041.

*120 Vision Memo, Marvin Minsky, Feb. 1967.

121 Estimating Stereo Disparities, Marvin Minsky, Feb. 1967. ($.80)

An interesting practical and theoretical problem is putting bounds on how much computation one needs to find the stereo-disparity between two narrow-angle stereo scenes. By narrow angle I mean situations wherein the angle subtended by the eyes is a very few degrees; the kind of correlation-disparity method

discussed here probably is not applicable to the wide-angle stereo we will usually use for scene-analysis in the Project.

122 Remarks on Correlation Tracking, Marvin Minsky, March 1967. ($.80)

123 Computer Tracking of Eye Motions, Marvin Minsky, Seymour Papert, March 1967. ($.80)

This memo explains why the Artificial Intelligence group of Project MAC is developing methods for on-line tracking of human eye movements. It also gives a brief resume of results to date and the next steps.

*124 Paradoxical Perception, Seymour Papert (never written).

125 CHAR PLOT, Michael Speciner, March 1967; (see A.I. Memo 138) ($.80)

CHAR PLOT is a routine which enables one to use the Calcomp plotter as a versatile output device.

126 A Quick Fail-Safe Procedure for Determining whether the GCD of 2 Polynomials is 1, Joel Moses, March 1967, MAC-M-345. ($.80)

Experiments by Collins have shown that given two polynomials chosen at random, the GCD has a high probability of being 1. Taking into account this probability and the cost of obtaining a GCD (some GCDs of polynomials of degree 5 in two or three variables can take on the order of a minute on the 7094), it appears that a quick method of determining whether the GCD is exactly 1 would be profitable. While no such method is known to exist, a fail-safe procedure has been found and is described here.

*127 Incorporating MIDAS Routines into PDP-6 LISP, Roland Silver, March 1967; revised in Nov. 1967 (127A).

*128 Hardware and Program Memo, Michael Beeler, March 1967.

129 EUTERPE: A Computer Language for the Expression of Musical Ideas, Stephen Smoliar, April 1967. (see also A.I. Memo 141) ($.80)

The electronic medium has vastly increased the amount of material available to the contemporay composer. The various pieces of electronic equipment available today allow one to produce any conceivable sound; yet because of the complex nature of their output, these devices are generally difficult to control and the composer of electronic music may take several hours to prepare but a few minutes of his creation.

130 A Miscellany of Convert Programming, Adolfo Guzman, Harold McIntosh, April 1967, MAC-M-346. ($1.60)

*131 POLYSEG, Arnold Griffith, April 1967.

*132 Additions to LAP, John White, July 1967. Superseded by A.I. Memo 152.

*133 A Glossary of Vision Terms, Russ Abbott, June 1967.

*134 PSEG: Standardization of Data, Jim Bowring, June 1967.

135 Automata on a 2-Dimensional Tape, Manuel Blum, Carl Hewitt, July 1967. ($.80)

This paper explains our approach to the problem of pattern recognition by serial computer. The rudimentary theory of vision presented here lies within the framework of automata theory. Our goal is to classify the types of patterns that can be recognized by an automaton that scans a finite 2-dimensional tape. This paper should be viewed as a Progress Report on work done to date.

*136 Matrix Inversion in LISP, John White, July 1967.

*137 PLANNER: A Language for Proving Theorems, Carl Hewitt, July 1967. Replaced by A.I. Memo 168, MAC-M-386, Oct. 1968. Revised June 1969.

*138 The Calcomp Plotter as an Output Device in TS and LISP, Michael Speciner, July 1967.

*139 Decomposition of a Visual Scene into Bodies, Adolfo Guzman, Sept. 1967, MAC-M-357.

*140 Perceptrons and Pattern Recognition, Marvin Minsky, Seymour Papert, Sept. 1967. Superceded by the book: Perceptrons, Minsky & Papert, M.I.T. Press, 1968.

*141 EUTERPE-LISP: A LISP System with Music Output, Stephen Smoliar, Sept. 1967. (see also A.I. Memo 243)

*142 STRING, Peter Samson, Sept. 1967.

143 Stereo and Perspective Calculations, Marvin Minsky, Sept. 1967. ($1.20)

*144 10 Test, Michael Beeler, Sept. 1967.

145 A Fast-Parsing Scheme for Hand-Printed Mathematical Expressions, William Martin, Oct. 1967, MAC-M-360. ($1.20)

A set of one-line text-book-style mathematical expressions is defined by a context free grammar. This grammar generates strings which describe the expressions in terms of mathematical symbols and some simple positional operators, such as vertical concatenation. The grammar rules are processed to abstract information used to drive the parsing scheme. This has been called syntax-controlled as opposed to syntax-directed analysis.

The parsing scheme consists of two operations. First, the X-Y plane is searched in such a way that the mathematical characters are picked up in a unique order. Then, the resulting character string is parsed using a precedence algorithm with certain modifications for special cases. The search of the X-Y plane is directed by the particular characters encountered.

*146 PICPAC: A PDP-6 Picture Package, Roland Silver, Oct. 1967.

*147 A Multiple Procedure DDT, Thomas Knight, Jan. 1968.

147A DDT Reference Manual, Eric Osman, Sept. 1971. ($2.00)

This memo describes the version of DDT used as the command level of the AI Laboratory Time Sharing System (ITS). Besides the usual program control, examination, and modification features, this DDT provides many special utility commands. It also has the capability to control several programs for a user and to a single instruction continue mode and interrupt on read or write reference to a given memory location.

*148 SUBM - A CONVERT Program for Constructing the Subset Machine Defined By a Transition System, Harold V. McIntosh, Jan. 1968.

*149 REC/8 - A CONVERT Compiler of REC for the PDP-6, Harold V. McIntosh, Jan. 1968.

*150 CGRU and CONG - CONVERT and LISP Programs to Find the Congruence Relations of a Finite State Machine, Harold V. McIntosh, Jan. 1968.

*151 Functional Abstraction in LISP and PLANNER, Carl Hewitt, Jan. 1968. (Replaced by AI TR-258.)

*152 PDP-6 LAP, John White, Jan. 1968; replaced by A.I. Memo 190.

*153 REEX - A CONVERT Program to Realize the McNaughton-Yamada Analysis Algorithm, Harold McIntosh, Jan. 1968.

154 The Artificial Intelligence of Hubert L. Dreyfus; A Budget of Fallacies, Seymour Papert, Jan. 1968. ($2.00)

This paper reviews the earlier writing of Hubert Dreyfus on Artificial Intelligence. It reveals many fallacies in Dreyfus' thinking.

155 A Left to Right then Right to Left Parsing Algorithm, William Martin, Feb. 1968. ($.80)

Determination of the minimum resources required to parse a language generated by a given context free grammar is an intriguing and yet unsolved problem. It seems plausible that any unambiguous context free grammar could be parsed in time proportional to the length, n, of each input string. Early has presented an algorithm which parses many grammars in time proportional to n, but requiires n on some. His work is an extension of Knuth's algorithm, which leads to a very efficient parse proportional to n of deterministic languages. This memo presents a different extension of Knuth's method. The algorithm is probably more efficient than Early's on certain grammars; it will fail completely on others. The essential idea may be interesting to those attacking the general problem.

*156 Linear Decision and Learning Models, Marvin Minsky, March 1968. Superceded by Ch. 12 in Perceptrons, Minsky & Papert, M.I.T. Press, 1968.

*157 Time-Sharing LISP for the PDP-6, John White, March 1968. Replaced by A.I. Memo 190.

*158 SARGE: A Program for Drilling Students in Freshman Calculus Integration Problems, Joel Moses, March 1968, MAC-M-369.

159 Numerical Solution of Elliptic Boundary Value Problems by Spline Functions, Jayant Shah, April 1968, MAC-M-371. ($1.20)

A numerical method for solving linear, two-dimensional elleptic boundary value problems is presented. The method is essentially the Ritz procedure which uses polynomial spline functions to approximate the exact solution. The spline functions are constructed by defining a polynomial function over each of a set of disjoint subdomains.

160 Focusing, B.K.P. Horn, May 1968. ($1.60)

This memo describes a method of automatically focusing the new vidisector. The same method can be used for distance measuring.

161 ITS 1.5 Reference Manual, D. Eastlake, R. Greenblatt, J. Holloway, T. Knight, S. Nelson, MAC-M-377. Revised July 1969 (ITS 1.4 Reference Manual, June 1968). (see also A.I. Memo 238) ($2.50)

This reference manual consists of two parts. The first (sections 1 through 6) is intended for those who are either interested in the ITS 1.5 time sharing monitor for its own sake or who wish to write machine language programs to run under it. Some knowledge of PDP-6 (or PDP-10) machine language is useful in reading this part. The second part (sections 7, 8, and 9) describes three programs that run under ITS. The first program (DDT) is a modified machine language debugging program that also replaces the monitor command level (where the user is typing directly at the monitor) present in most time-sharing systems. The remaining two (PEEK and LOCK) are a status display and a miscellaneous utility program. It should be remembered that the McCulloch Laboratory PDP-6 and PDP-10 installation is undergoing continuous software and hardware development which may rapidly outdate this manual.

*162 Remarks on Visual Display and Console Systems, Marvin Minsky, June 1963, revised July 1968.

163 Holes, Patrick Winston, August 1968, revised April 1970. ($.80)

This memo discusses line drawings of objects with holes.

*164 Producing Memos Using TJ6, TECO and the Type 37 Teletype, Larry Krakauer, Sept. 1968. Replaced by Memo 164A.

164A The Text-Justifier TJ6, R. Greenblatt, B.K.P. Horn, L. Krakauer, June 1970. ($1.20)

This memo describes the TJ6 type justifying program, which can be used in the production of memos. In addition, Appendices of this memo contain related information about TECO, thus gathering most of the information needed for

producing write-ups into one location.

165 Description and Control of Manipulation by Computer-Controlled Arm, Jean-Yves Gresser, Sept. 1968. ($2.00)

The immediate purpose of the research on Intelligent Automata is to have an autonomous machine able to understand uncomplicated commands and to manipulate simple objects without human intervention. This thesis is concerned with the programming of a special output device of the present machine existing at Project MAC: an arm with eight degrees of freedom, made of our identical segments. Classical approaches through hill-climbing and optimal control techniques are discussed. A new method is proposed to decompose the problem, in an eight-dimensional space, into a sequence of sub-routines in spaces with fewer dimensions. Each subproblem can then be solved with simple analytical geometry. A simulation program, which applies this method, is able to propose several configurations for a given goal (expressed as a point in a five-dimensional space).

*166 Recognition of Topological Invariants by Modular Arrays, Terry Beyer, September 1968.

*167 Linear Separation and Learning, Marvin Minsky and Seymour Papert, October 1968. This is a reprint of page proofs of Ch. 12 of Perceptrons, Minsky and Papert, M.I.T. Press, 1968. It replaces A.I. Memo 156.

*168 PLANNER, Carl Hewitt, MAC-M-386, October 1968, revised June 1969. (Replaced by AI-TR-258.)

*169 PEEK and LOCK, Donald Eastlake III, MAC-M-387, November 1968; replaced by revised A.I. Memo 161A, 260, 261.

*170 WIRElist, John Holloway, January 1969.

171 Decomposition of a Visual Scene Into Three-Dimensional Bodies, Adolfo Guzman, January 1969. MAC-M-391. ($.80)

The program described here takes as it input a collection of lines, vertices and surfaces describing a scene, and analyzes the scene into a composition of three-dimensional objects. The program does not need to know the form (model, or pattern) of the objects which are likely to appear: the scene is not searched for cubes, wedges, or houses, with an a priori knowledge of the form of these objects; rather, the program pays attention to configurations of surfaces and lines which would make plausible three-dimensional solids, and in this way bodies are identified. Partially occluded bodies are handled correctly. The program is restricted to scenes formed by straight lines, where shadows or noise are not present. It has been tested in rather complicated scenes composed by rather simple objects. Examples are given.

*172 Robot Utility Functions, Stewart Nelson, Michael Levitt, February 1969; replaced by Revised A.I. Memo 161A, July 1969.

*173 A Heuristic Program That Constructs Decision Trees, Patrick Winston, March 1969.

174 The Greenblatt Chess Program, Richard Greenblatt, Donald Eastlake III, Stephen Crocker, April 1969. ($.80)

Since mid-November 1966 a chess program has been under development at the Artificial Intelligence Laboratory of Project MAC at M.I.T. This paper describes the state of the program as of August 1967 and gives some of the details of the heuristics and algorithms employed.

*175 On Optimum Recognition Error and Reject Trade-off, C.K. Chow, April 1969.

*176 Discovering Good Regions for Teitelman's Character Recognition Scheme, Patrick Winston, May 1969.

177 Preprocessor for Programs Which Recognize Scenes, H.N. Mahabala, August 1969. ($1.60)

A visual scene is tranformed from a very simple and convenient format, to an internal format which describes the same scene, but is more akin to complex manipulations. This format is compatible with programs like SEE. The entire analysis is done using a basic primitive which gives the orientation of a point with respect to a directed line. A novel handling of inaccuracies in the scene is achieved by considering the lines to be strips of small but not negligible width. The criterion is very general and easy to modify.

178 The Image Dissector Eyes, B.K.P. Horn, August 1969. ($1.60)

This is a collection of data on the construction, operation and performance of the two image dissector cameras. Some of this data is useful in deciding whether certain shortcomings are significant for a given application and if so, how to compensate for them.

179 The Arithmetic-Statement Pseudo-Ops: .I and .F, B.K.P. Horn, August 1969. ($.80)

This is a feature of MIDAS which facilitates the rapid writing and debugging of programs involving much numerical calculation. The statements used are ALGOL-like and easy to interpret.

180 The Integration of a Class of Special Functions with the Risch Algorithm, Joel Moses, MAC-M-421. ($.80)

We indicate how to extend the Risch algorithm to handle a class of special functions defined in terms of integrals. Most of the integration machinery for this class of functions is similar to the machinery in the algorithm which handles logarithms. A program embodying much of the extended integration algorithm has been written. It was used to check a table of integrals and it succeeded in finding some misprints in it.

*181 PROGRAMMAR: A Language for Writing Grammars, Terry Winograd, November 1969. (see A.I. Memo 282 & AI-TR-235)

*182 Display Functions in LISP, Thomas Binford, 1970. ($.80)

183 On Boundary Detection, Annette Herskovits and Thomas Binford, July 1970. ($2.00)

A description is given of how edges of prismatic objects appear through a television camera serving as visual input to a computer. Two types of edge-finding predicates are proposed and compared, one linear in intensity, the other non-linear. A statistical analysis of both is carried out, assuming input data distorted by a Gaussian noise. Both predicates have been implemented as edge-verifying procedures, i.e. procedures aiming at high sensitivity and limited to looking for edges when approximate location and direction are given. Both procedures have been tried on actual scenes. Of the two procedures, the non-linear one emerged as a satisfactory solution to line-verification because it performs well in spite of surface irregularities.

184 Parsing Key Word Grammars, William Martin, MAC-M-395. ($.80)

Key word grammars are defined to be the same as context free grammars, except that a production may specify a string of arbitrary symbols. These grammars define languages similar to those used in the programs CARPS(1) and ELIZA(2). We show a method of implementing the LR(k) parsing algorithm for context free grammars which can be modified slightly in order to parse key word grammars. When this is done the algorithm can use many of the techniques used in the ELIZA parse. Therefore, the algorithm helps to show the relation between the classical parsers and key word parsers.

*185 Proposal to ARPA for Research on Artificial Intelligence at MIT, 1970-1971, Marvin Minsky and Seymour Papert, December 1970.

*186 Seymour Papert, LOGO report, Never written (see A.I. Memo 246).

187 Form and Content in Computer Science, Marvin Minsky, December 1969; ACM 'Turing Lecture' August 1969. ($1.20)

*188 A Stability Test for Configurations of Blocks, Manuel Blum, Arnold Griffith, Bernard Neumann, February 1970.

189 Construction of Decision Trees, E. Roger Banks, February 1970. ($1.20)

The construction of optimal decision trees for the problem stated within can be accomplished by an exhaustive enumeration. This paper discusses two approaches. The section on heuristic methods gives mostly negative results (e.g. there is no merit factor that will always yield the optimal test, etc.), but most of these methods do give good results. The section entitled Exhaustive Enumeration Revisited indicates some powerful shortcuts that can be applied to an exhaustive enumeration, extending the range of this method.

190 Interim LISP Progress Report, John White, March 1970. ($2.00)

191 A. I. Bibliography, April 1970, updated later (FREE!)

*192 Removing Shadows in a Scene, Richard Orban, August 1970.

*193 Learning, Teaching and A.I., Marvin Minsky and Seymour Papert. Never

written.

194 Movie Memo, Michael Beeler, April 1970. ($1.20)

This is intended as a brief explanation of how to use the Kodak movie camera in sync with a display. Please use care in use of the camera, as it is fairly expensive and more or less irreplaceable.

195 INSIM1: A Computer Model of Simple Forms of Learning, Thomas L. Jones, April 1970. ($.80)

196 Hypergeometric Functions in MATHLAB, Lewis Wilson, May 1970. ($1.60)

*197 A Simple Algorithm for Self-Replication, Terry Winograd, term paper written November 1967, released as A.I. Memo May 1970.

198 Cellular Automata, E. Roger Banks, June 1970. ($2.00)

This paper presents in order:

1) a brief description of the results,
2) a definition of cellular automata,
3) discussion of previous work in this area by Von Neumann and Codd, and
4) details of how the prescribed behaviors are achieved (with computer simulations included in the appendices).

The results include showing that a two state cell with five neighbors is sufficient for universality.

199 The Function of FUNCTION in LISP, Joel Moses, June 1970. ($.80)

A problem common to many powerful programming languages arises when one has to determine what values to assign to free variables in functions. Different implementational approaches which attempt to solve the problem are considered. The discussion concentrates on LISP implementations and points out why most current LISP systems are not as general as the original 1.5 system. Readers not familiar with LISP should be able to read this paper without difficulty since we have tried to couch the argument in ALGOL-like terms as much as possible.

*200 1968-1969 Progress Report, Marvin Minsky and Seymour Papert. ($1.65)

201 Comparative Schematology, Michael Paterson and Carl Hewitt, November 1970. ($.80)

202 Peter Samson's Music Processor, BIG, Michael Beeler, July 1970. ($.80)

*203 Micro-Planner Reference Manual, Gerald Sussman and Terry Winograd, July 1970.

203A Micro-Planner Reference Manual, (revised), Gerald Sussman, Terry Winograd, and Eugene Charniak, December 1971. ($1.20)

This is a manual for the use of the Micro Planner interpreter, which implements a

subset of Carl Hewitt's language, PLANNER and is now available for use by the Artificial Intelligence Group.

204 Extending Guzman's SEE Program, Martin Rattner, July 1970 (M.I.T. B.S. Thesis June 1970). ($2.00)

Adolfo Guzman's SEE program groups the region of a two-dimensional scene into bodies, using local evidence in the scene to link regions together. This paper discusses an extended version of the SEE procedure that makes extensive use of evidence in the scene which indicates that two regions should be split into separate bodies.

*205 Look-Ahead Strategies in One Person Games with Randomly Generated Game Trees, David S. Johnson, M.I.T. M.S. thesis August 1968, released as A.I. Memo July 1970.

206 The Vision Laboratory, Part One, Thomas O. Binford, July 1970. ($1.20)

Some of the facilities for vision programming are discussed in the format of a user's manual.

*207 More Comparative Schematology, Carl E. Hewitt, August 1970. (Replaced by AI TR-258.)

*208 Teaching Procedures in Humans and Robots, Carl Hewitt. Paper presented April 1970, issued as A.I. Memo September 1970. (Replaced by AI TR-258.)

*209 Digital Flight Simulation, David Silver, March 1971. Program listing available from line printer FLIGHT >, DSK DS;

210 A User's Guide to the A.I. Group LISCOM LISP Compiler, Jeffrey P. Golden, December 1970. ($.80)

The operation of a version of a fast arithmetic LISP compiler for ITS on the PDP-10 is discussed.

*211 Equivalence Problems in a Model of Computation, Michael S. Paterson, Ph.D. thesis (Trinity College, Cambridge University), 1967; issued in November 1970.

*212 Terry Winograd, Using the Language-Understanding System, Never written (see A.I. Memo 282).

213 The Computer as a Performing Instrument, Gordon Mumma and Stephen Smoliar. Presented as a MAC seminar February 1970, issued as A.I. Memo February 1971. ($.80)

This is a discussion of the use of analog and/or digital computers in the performance of music, from a semi-technical viewpoint. Historical performances are described, leading to the author's own Conspiracy 8 piece, with references.

*214 Linking Loader for MIDAS, Peter Samson. Originally printed as MAC Memo in January 1966, issued as A.I. Memo March 1971.

215 How to Get Onto the System, Mark Dowson, April 1971. ($.80)

This memo is intended to get very new users started on the AI system. It presents some simple rituals for making and editing files, getting print out, making microtapes and so on. Most of the rituals given are not the only ways of doing something or even necessarily the simplest, but they do work (sometimes).

215A Instant TJ6, How to Get the System to Type Your Papers, M. Dowson, Sept. 1971. ($.80)

The simple features of TJ6 are described. This is a (primitive) primer to accompany A.I. Memo 164A, The Text-Justifier TJ6.

216 Theories, Pretheories, and Finite State Transformations on Trees, Mitchell Wand, May 1971. ($1.20)

The closure of an algebra is defined as a generalization of the semigroup of a finite automaton. Pretheories are defined as a subclass of the closed algebras, and the relationship between pretheories and the algebraic theories of Lawvere (1963) is explored. Finally, pretheories are applied to the characterization problem of finite state transformations on trees, solving an open problem of Thatcher (1969).

217 Computer Proofs of Limit Theorems, W.W. Bledsoe, Robert S. Boyer, W.H. Henneman, June 1971. ($.80)

Some relatively simple concepts have been developed which when incorporated into existing automatic theorem proving programs (including those using resolution), enable them to prove efficiently a number of the limit theorems of elementary calculus, including the theorem that differentiable functions are continuous. These concepts include: (1) A limited theory of types, to designate whether a given variable belongs to a certain interval on the real line, (2) An algebraic simplification routine, (3) A routine for solving linear inequalities, applicable to all areas of analysis, and (4) A limit heuristic, designed especially for the limit theorems of calculus.

218 Information Theory and the Game of Jotto, Michael Beeler, August 1971. ($.80)

The operation and use of a PDP-10 program for playing the word game JOTTO are described. The program works by making the move which will give it the most information in the information-theoretic sense.

*237 An Inquiry Into Algorithmic Complexity, Patrick E. O'Neil, September 1971.

238 ITS Status Report, Donald Eastlake, April 1972. ($2.00)

ITS is a time-shared operating system designed for the Artificial Intelligence Laboratory DEC PDP-10/PDP-6 installation and tailored to its special requirements. This status report described the design philosophy behind the ITS system, the hardware and software facilities of the system implemented with this philosophy, and some information on work currently in progress or desirable in

The following page (332) was rec'd as is - diskewed original.

theories of cognitive processes and to conjectures about the possibility of producing much larger changes than are usually though possible in the expected intellectual achievement of children. This proposal is formulated in terms of the self-sufficient immediate objectives.

247 Teaching Children Thinking, Seymour Papert, October 1971 (LOGO Memo No. 2). ($1.20)

The purpose of this essay is to present a grander vision of an educa tional system in which technology is used not in the form of machines for processing children but as something the child himself will learn to manipulate, to extend, to apply to projects, thereby gaining a greater and more articulate mastery of the world, a sense of the power of applied knowledge and a self-confidently realistic image of himself as an intellectual agent. Stated more simply, I believe with Dewey, Montessori and Piaget that children learn by doing and by thinking about what they do. And so the fundamental ingredients of educational innovation must be better things to do and better ways to think about oneself doing these things.

248 Twenty Things To Do With A Computer, Seymour Papert and Cynthia Solomon, June 1971 (LOGO Memo No. 3). ($1.60)

249 Teaching Children to be Mathematicians VS. Teaching About Mathematics, Seymour Papert, July 1971 (LOGO Memo No. 4). ($1.20)

Being a mathematician is no more definable as knowing a set of mathematical facts than being a poet is definable as knowing a set of linguistic facts. Some modern math ed reformers will give this statement a too easy assent with the comment: Yes, they must understand, not merely know. But this misses the capital point that being a mathematician, again like being a poet, or a composer or an engineer, means doing, rather than knowing or understanding. This essay is an attempt to explore some ways in which one might be able to put children in a better position to do mathematics rather than merely to learn about it.

250 PLANNER Implementation Proposal to ARPA, 1971-1972, Carl Hewitt, December 1971. ($1.20)

251 Mini-Robot Proposal to ARPA, Marvin Minsky, January 1972. ($1.60)

During the next decade it will become practical to use more and more sophisticated techniques of automation -- we shall call this robotics -- both in established industries and in new areas. The rate at which these techniques become available will depend very much on the way in which research programs are organized to pursue them. The issues involved are rather large and touch not only on technical matters but also on aspects of national economic policy and attitudes toward world trade positions. The project herein proposed is concerned with the development of two particular aspects of Robotics, namely: (1) Development of a miniature hand-eye system, (2) Developoment of remote, ARPA-NETWORK style operation of robotic systems, in which simple jobs are handled locally while more complex computations are done on larger, more centralized machines.

*252 Artificial Intelligence Progress Report, Marvin Minsky and Seymour

the near future.

239 HAKMEM, M. Beeler, R.W. Gosper, R. Schroeppel, February 1972. ($2.50)

HAKMEM is a collection of 191 miscellaneous items, most of which are too short to justify documentation individually. Topics are related to computers, but range over number theory and several other branches of mathematics, games, programming techniques, and electronic circuits. Items come from A. I. work, outside contributors, and computer folklore.

"I keep HAKMEM in the bathroom, it's great reading." - Knuth

*240 11SIM Reference Manual, Donald Eastlake, December 1971.

240A 11SIM Reference Manual, (revised), Donald Eastlake, February 1972. ($1.60)

program that simulates a Digital Equipment Corporation PDP-11 computer and many f its peripherals on the A.I. Laboratory Time Sharing System (ITS) is described om a user's reference point of view. This simulator has a built in DDT-like mmand level which provides the user with the normal range of DDT facilities but so with several special debugging features built into the simulator. The DDT mand language was implemented by Richard M. Stallman while the simulator was tten by the author of this memo.

An A.I. Approach to English Morphemic Analysis, Terry Winograd, September 1971. ($.80)

pape illustrates an approach toward understanding natural language through chniques of artificial intelligence. It explores the structure of English ndings both morpho-graphemically and semantically. It illustrates the use edures and semantic representations in relating the broad range of ge a language user brings to bear on understanding and utterance.

Using the EUTERPE Music System, Stephen W. Smoliar, October 1971.

describes the practical implementation of programs written of UTERPE. Details of this language are given in the auth rocessing Model of Musical Structures AI-TR-242) an e. We shall only be concerned with the preparati ource program.

k Dowson) Never written.

I.T., 1971-

osal to ARPA for Research on Artificiate objective is rt, October Marvin Minsky and Seymour Pape an environment of puter Laboratory for Elem jectives are related to (LOGO Memo No. 1). (

rch project on of new method mputer-contro

Papert, January 1972. (Identical with pp. 129-224 of the 1971 Project MAC Progress Report VIII.) AD-754-820. ($3.00)

253 The Computer-Controlled Oculometer: A Prototype Interactive Eye Movement Tracking System, Matthew J. Hillsman, R. Wade Williams and John S. Roe, February 1972. (Originally published as a report to NASA in September 1970.) ($2.50)

The Oculometer electro-optic subsystem utilizes near-infrared light reflected specularly off the front surface of the subject's cornea and diffusely off the retina, producing a bright pupil with an overriding corneal highlight. An electro-optic scanning aperture vidi-sector within the unit, driven by a digital eye-tracking algorithm programmed into the PDP-6 computer, detects and tracks the centers of the corneal highlight and the bright pupil to give eye movement measurements. A computer-controlled, moving mirror head motion tracker directly coupled to the vidissector tracker permits the subject reasonable freedom of movement. Various applications of this system, which are suggested by the work reported here, include:

(a) using the eye as a control device,
(b) recording eye fixation and exploration patterns,
(c) game playing,
(d) training machines, and
(e) psycho-physiological testing and recording.

254 NIM: A Game-Playing Program, Seymour Papert and Cynthia Solomon, Feb. 1972 (LOGO Memo No. 5). ($.80)

This note illustrates some ideas about how to initiate beginning students into the art of planning and writing a program complex enough to be considered a project rather than an exercise on using the language or simple programming ideas. The project is to write a program to play a simple game (one-pile NIM of 21) as invincibly as possible. We developed the project for a class of seventh grade children we taught in 1968-69 at the Muzzy High School in Lexington, Mass. (This work was supported by NSF Contract No. NSF-C 558 to Bolt, Beranek and Newman, Inc.) This was the longest programming project these children had encountered, and our intention was to give them a model of how to go about working under these conditions.

*255 Why Conniving Is Better Than Planning, Gerald Jay Sussman, Feb. 3, 1972.

255A Why Conniving Is Better Than Planning, Gerald Sussman and Drew McDermott, April 1972. (see also A.I. Memo 259) ($1.20)

This paper is a critique of a computer programming language, Carl Hewitt's PLANNER, a formalism designed especially to cope with the problems that Artificial Intelligence encounters. It is our contention that the backtrack control structure that is the backbone of PLANNER is more of a hindrance in the solution of problems than a help. In particular, automatic backtracking encourages inefficient algorithms, conceals what is happening from the user, and misleads him with primitives having powerful names whose power is only superficial. An alternative, a programming language called CONNIVER which avoids these problems, is presented from the point of view of this critique.

256 Efficiency of Equivalence Algorithms, Michael J. Fischer. Presented at Symposium on Complexity of Computer Computations, T.J. Watson Research Center, March 22, 1972; issued as A.I. Memo April 1972. ($.80)

257 A Two Counter Machine Cannot Calculate 2**n , Richard Schroeppel, May 1973. ($1.20)

*259 Conniver Reference Manual, Drew McDermott and Gerald Sussman, May 1972.

259A THE CONNIVER REFERENCE MANUAL, Drew McDermott and Gerald Sussman, January 1974. ($2.00)

This manual is an introduction and reference the latest version of the Conniver programming language, an A. I. language with general control and data-base structures.

*260 LOCK, Donald E. Eastlake, June 1972.

260A LOCK, Donald E. Eastlake, January 1974. ($.80)

LOCK is a miscellaneous utility program operating under the ITS system. It allows the user to easily and conveniently perform a variety of infrequently required tasks. Most of these relate to console input-output or the operation of the ITS system.

*261 PEEK, Eastlake, May 1973.

261A PEEK, Donald E. Easlake, February 1974. ($.80)

PEEK is a utility program designed to operate under the ITS time sharing system. It enables a user to monitor a variety of aspects of the time sharing system by providing periodically updated display output or periodic printed output to teletype or line printer.

262 A Concrete Approach To Abstract Recursive Definitions, Mitchel Wand, June 1972 (Paper presented at Symposium on the Theory of Automata Programming and Languages, I.R.I.A., Paris, July 1972). ($.80)

We introduce a non-categorical alternative to Wagner's Abstract Recursive Definitions (Wg-1, 2), using a generalization of the notion of clone called a u-clone. Our more concrete approach yields two new theorems: 1. The free u-clone generated by a ranked set is isomorphic to the set of loop-representable flow diagrams with function symbols in the set. 2. For every element of a u-clone there is an expression analogous to a regular expression. Several well-known theorems of language and automata theory are drawn as special cases of this theorem.

*263 A Heterarchical Program for Recognition of Polyhedra, Yoshiaki Shirai, June 1972 (see Artificial Intelligence Vol 4, No 2) (also part of AI-TR-281)

264 Developing a Musical Ear: A New Experiment, Jeanne Bamberger, July 1972. (LOGO Memo No. 6) ($.80)

This is a report on some ideas we have been developing at M.I.T. for self-paced, independent music study. The aim of our approach is to nurture in students that enigmatic quality called musical -- be it a musical ear or an individual's capacity to give a musical performance. While all of us cherish these qualities, rarely do we come to grips with them directly in teaching. More often we rely on our magical or mystical faith in the inspiration of music itself, and its great artists, to do the teaching. And for some (maybe ultimately all) this is the best course. But what about the others to whom we teach only the techniques of playing instruments or some facts about music -- its forms, its history and its apparent elements? How often do we have or take the time to examine the assumptions underlying these facts we teach, or to question the relation between what we teach and what we do as musicians?

265 Infants in Children Stories-Toward a Model of Natural Language Comprehension, Garry S. Meyer, August 1972. ($2.00)

How can we construct a program that will understand stories that children would understand? By understand we mean the ability to answer questions about that story. We are interested here with understanding natural language in a very broad area. In particular how does one understand stories about infants? We propose a system which answers such questions by relating the story to background real world knowledge. We make use of the general model proposed by Eugene Charniak in his Ph.D. thesis (Charniak 72). The model sets up expectations whih can be used to help answer questions about the story. There is a set of routines called BASE-routines that correspond to our real world knowledge and routines that are put-in which are called DEMONS that correspond to contextual information. Context can help to assign a particular meaning to an ambiguous word, or pronoun.

267 Manipulator Design Vignettes, Marvin Minsky, October 1972. ($1.20)

This memo is about mechanical arms. The literature on robotics seems to be deficient in such discussions, perhaps because not enough sharp theoretical problems have been formulated to attract interest. I'm sure many of these matters have been discussed in other literatures -- prosthetics, orthopedics, mechanical engineering, etc., and references to such discussions would be welcome. We raise these issues in the context of designing the mini-robot system in the A.I. Laboratory in 1972-1973. But we would like to attract the interest of the general heuristic programming community to such questions.

268 A Human Oriented Logic for Automatic Theorem Proving, Arthur J. Nevins, October 1972. ($1.60)

A deductive system is described which should replace the resolution principle as the primary instrument for proving theorems by computer. The system has been realized by a computer program whose performance compares favorably with the best resolution type theorem provers while at the same time it appears more consistent with the logical processes used by humans when proving theorems.

*269 Proposal to ARPA for Continued Research on A.I., Marvin Minsky, October 1972.

*270 Teaching of Procedures-Progress Report, Gerald Jay Sussman, October 1972.

(Replaced by AI-TR-297)

*272 How the GAS Program Works With a Note on Simulating Turtles with Touch Sensors, Michael Speciner, December 1972.

273 The Little Robot System, David Silver, January 1973. ($.80)

The Little Robot System provides for the I.T.S. user a medium size four degree of freedom six axis robot which is controlled by the PDP-6 computer through the programming language LISP. The robot includes eight force feedback channels which when interpreted by the PDP-6 are read by LISP as the signed force applied to the end of the fingers.

274 Proposal to ARPA for Continuation of Micro-Automation Development, Marvin Minsky, January 1973.

275 Differential Peceptrons, Martin Brooks, Jerrold Ginsparg, January 1973. ($1.20)

This paper shows that differential perceptrons are equivalent to perceptrons on the class of figures that fit exactly onto a sufficiently small square grid. By investigating predicates of various geometric transformations, we discover that translation and symmetry can be computed in finite order using finite coefficients in both continuous and discrete cases.

276 The Making of the Film, SOLAR CORONA, Michael Beeler, February 1973. (Film Memo No. 1.) ($.80)

The film SOLAR CORONA was made from data taken from August 14, 1969 through May 7, 1970, by OSO-VI, one of the Orbiting Satellite Observatories.

277 A Linguistics Oriented Programming Language, Vaughan R. Pratt, February 1973. ($1.20)

A programming language for natural language processing programs is described. Examples of the output of programs written using it are given. The reasons for various design decisions are discussed. An actual session with the system is presented, in which a small fragment of an English-to-French translator is developed. Some of the limitations of the system are discussed, along with plans for further development.

278 D-SCRIPT: A Computational Theory of Descriptions, Robert C. Moore, February 1973. ($1.20)

This paper describes D-SCRIPT a language representing knowledge in artifical intelligence programs. D-SCRIPT contains a powerful formalism for descriptions, which permits the representation of statements that are problematic for other systems. Particular attention is paid to problems of opaque contexts, time contexts, and knowledge about knowledge. The design of a theorem prover for this language is also considered.

279 Pretty-Printing, Converting List to Linear Structure, Ira Goldstein, February 1973. ($1.20)

Pretty-printing is the conversion of list structure to a readable format. This paper outlines the computational problems encountered in such a task and documents the current algorithm in use.

280 Elementary Geometry Theorem Proving, Ira Goldstein, April 1973. ($1.60)

An elementary theorem prover for a small part of plane Euclidean geometry is presented. The purpose is to illustrate important problem solving concepts that naturally arise in building procedural models for mathematics.

282 Grammar for the People: Flowcharts of SHRDLU's Grammar, Andee Rubin, March 1973. ($1.20)

The grammar which SHRDLU uses to parse sentences is outlined in a series of flowcharts which attempt to modularize and illuminate its structure. In addition, a short discussion of systematic grammar is included.

*284 Proposal To ARPA For Continued Research On A.I. For 1973, Marvin Minsky and Seymour Papert, June 1973. (see A.I.Memo 299)

285 The Binford-Horn LINEFINDER, B.K.P. Horn, Revised December 1973. (Originally published as A.I.Vision Flash 16, July 1971.) ($.80)

This paper briefly describes the processing performed in the course of producing a line drawing from an image obtained through an image dissector camera. The edge-marking phase uses a non-linear parallel line-follower. Complicated statistical measures are not used. The line and vertex generating phases use a number of heuristics to guide the transition from edge-fragments to cleaned-up line-drawing. Higher-level understanding of the blocks-world is not used. Sample line-drawings produced by the program are included.

*286 The FINDSPACE Problem, Gerald J. Sussman, March 1973. (Originally issued as A.I. Vision Flash 18, August 1971.)

*287 Finding The Skeleton of a Brick, Tim Finin, March 1973. (Originally issued as A.I. Vision Flash 19, August 1971.)

*288 Richard Schroeppel, Never written.

*289 Visual Position Extraction Using Stereo Eye Systems With A Relative Rotational Motion Capability, Daniel W. Corwin, March 1973. (Originally issued as Vision Flash 22, January 1972.)

290 Paterson's Worm, Michael Beeler, June 1973. ($.80)

A description of a mathematical idealization of the feeding pattern of a kind of worm is given.

*292 Telnet Reference Manual, Donald Eastlake. In progress.

*293 Minsky and Papert, Never written

295 On Lightness, Berthold K.P. Horn, October 1973. ($2.00)

The intensity at a point in an image is the product of the reflectance at the corresponding object point and the intensity of illumination at that point. We are able to perceive lightness, a quantity closely correlated with reflectance. How then do we eliminate the component due to illumination from the image on our retina? The two components of image intensity differ in their spatial distribution. A method is presented here which takes advantage of this to compute lightness from image intensity in a layered, parallel fashion.

296 An Essay on the Primate Retina, David Marr, January 1974. ($2.00)

This essay is considerably longer than the published version of the same theory, and is designed for readers who have only elementary knowledge of the retina. It is organized into four parts. The first is a review that consists of four sections: retinal anatomy, physiology, psychophysics, and the retinex theory. The main exposition starts with Part II, which deals with the operation of the retina in conditions of moderate ambient illumination. The account is limited to an analysis of a single cone channel -- like the red or the green one -- the rod channel being referred to frequently during the account. Part III considers various interesting properties of retinal signals, including those from the fully dark-adapted retina; and finally the thorny problem of bleaching adaptation is dealt with in Pari IV. The general flow of the account will be from the receptors to the ganglion cells, and an analysis of each of the retinal cells and synapses ius given in the appropriate place.

298 Uses of Technology to Enhance Education, Seymour Papert, June 1973 (LOGO Memo 8). ($2.50)

This paper is the substance of a proposal to the N.S.F. for support of research on children's thinking and elementary education. This work was supported by the National Science Foundation under grant GJ-1049 and conducted at the Artificial Intelligence Laboratory.

299 Proposal to ARPA for Research on Intelligent Automata and Micro-Automation, 1974-1976, September 1973. ($2.50)

This document consists of thorough descriptions of the various sections of the laboratory and their intent for the next two years.

300 Design Outline for Mini-Arms Based on Manipulator Technology, Carl R. Flatau, May 1973 (issued as A.I. Memo January 1974.) ($1.60)

The design of small manipulators is an art requiring proficiency in diverse disciplines. This paper documents some of the general ideas illustrated by a particular design for an arm roughly one quarter human size. The material is divided into the following sections:

A. General design constraints.
B. Features of existing manipulator technology.
C. Scaling relationships for major arm components.
D. Design of a particular small manipulator.
E. Comments on future possibilities.

301 A Mechanical Arm Control System, Richard C. Waters, January 1974.

($1.20)

This paper describes a proposed mechanical arm control system, and some of the lines of thought which lead to the current design. It is divided into five main sections:

I. Some Ideas on Control
II. The Basic Capabilities of the Arm Controller
III. The Internal Structure of the Arm Controller
IV. The Dynamic Level of the Arm Control System
V. The Procedural Level of the Arm Control System

302 A Relaxation Approach to Splitting in an Automatic Theorem Prover, Arthur Nevins, January 1974. ($1.20)

The splitting of a problem into subproblems often involves the same variable appearing in more than one of the subproblems. This makes these subproblems dependent upon one another since a solution to one may not qualify as a solution to another. A two stage method of splitting is described which first obtains solutions by relaxing the dependency requirement and then attempts to reconcile solutions to different subproblems. The method has been realized as part of an automatic theorem prover programmed in LISP which takes advantage of the procedural power that LISP provides.

303 Plane Geometry Theorem Proving Using Forward Chaining, Arthur Nevins, January 1974. ($1.20)

A computer program is described which operates on a subset of plane geometry. Its performance not only compares favorably with previous computer programs, but within its limited problem domain (e.g. no curved lines or introduction of new points), it also compares favorably with the best human theorem provers. The program employs a combination of forward and backward chaining with the forward component playing the more important role.

304 Acceleration of Series, William Gosper, March 1974. ($2.00)

The abstract of convergence of infinite series can be accelerated by a suitable splitting of each term into two parts and then combining the second part of the n-th term with the first part of the (n+1)-th term to get a new series and leaving the first part of the first term as an "orphan." Repeating this process an infinite number of times, the series will often approach zero, and we obtain the series of orphans, which may converge faster than the original series. heuristics for determining the splits are given. Various mathematical constants, orginally defined as series having a term ratio which approaches 1, are accelerated into series having a term ratio less than 1.

305 Summary of MYCROFT: A System for Understanding Simple Picture Programs, Ira Goldstein, May 1974. ($1.60)

A collection of powerful ideas--description, plans, linearity, insertions, global knowledge and imperative semantics--are explored which are fundamental to debugging skill. To make these concepts precise, a computer monitor called MYCROFT is described that can debug elementary programs for drawing pictures.

The programs are those written for LOGO turtles.

306 A Framework for Representing Knowledge, Marvin Minsky, June 1974. ($2.00)

This is a partial theory of thinking, combining a number of classical and modern concepts from psychology, linguistics, and A. I. Whenever one encounters a new situation(or makes a substantial change in one's viewpoint) he selects from memory a structure called a frame: a remembered framework to be adapted to fit reality by changing details as necessary.

307 LLOGO: An Implementation of LOGO in LISP, Ira Goldstein, June 1974. ($1.60)

This paper describes LLOGO, an implementation of the LOGO language written in MACLISP for the ITS, TEN50 and TENEX PDP-10 systems, and MULTICS. The relative merits of LOGO and LISP as educational languages are discussed. Design decisions in the LISP implementation of LOGO are contrasted with those of two other implementations: CLOGO for the PDP-10 and 11LOGO for the PDP-11, both written in assembler language. LLOGO's special facilities for character-oriented display terminals, graphic display "turtles", and music generation are also described.

308 Force Feedback in Precise Assembly Tasks, Hirochika Inoue, August 1974. 8$1.20)

This paper describes the execution of precise assembly asks by a robot. The level of performance of the experimental system allows such basic actions as putting a peg into a hole, screwing a nut on a bolt and picking up a thin piece from a flat table. The tolerance achieved in the experiments was 0.001 inch. The experiments proved that force feedback enabled the reliable assembly of a bearing complex consisting of eight parts with close tolerances. A movie of the demonstration is available.

309 Commenting Proofs, James R. Geiser, August 1974. ($.80)

This paper constitutes a summary of a seminar entitled "Commenting Proofs" given at the A.I. Lab at M.I.T. during the Spring of 1974. The work is concerned with new syntactic structures in formal proofs which derive from their pragmatic and semantic aspects. It is a synthesis of elements from Yessenin-Volpin's foundational studies (e.g. "Ultraintuitionism and the Antitraditional Program for the Foundation of Mathematics", Proceedings of the Summer Conference on Intuitionism and Proof Theory at Buffalo, New York, 1968) and developments in Artificial Intelligence concerned with commenting programs and the use of this idea in automatic debugging procedures (e.g. Gerald Sussman's thesis: "A Computational Model of Skill Acquisition", M.I.T. 1973).

BOOKS IN PRINT

Computation, Marvin Minsky, Prentice Hall, 1967.

Perceptrons, Marvin Minsky and Seymour Papert, MIT Press, 1968.

Semantic Information Processing, Marvin Minsky (Ed.) MIT Press, 1968.

Counter-Free Automata, Seymour Papert and Robert McNaughton, MIT Press, 1971.

Understanding Natural Language, Terry Winograd, Academic Press, 1972.

TECHNICAL REPORTS**

**Please note:

All technical reports accompanied by an 'AD' number may be obtained from the following sources:

Government contractors may obtain AI-TR reports from: Defense Documentation Center, Cameron Station, Alexandria, Va. 22314. Specify AD number.

Others may obtain the reports from: National Technical Information Service, Operations Division, Springfield, Va. 22151. Specify AD number. Prices vary for microfiche and paper documents.

---

*AI-TR-219 Bobrow, Daniel G., Natural Language Input for a Computer Problem Solving Language, June 1964. (MAC-TR-1) AD-604-730. (In Semantic Info. Proc.)

*AI-TR-220 Raphael, Bertram, SIR: A Computer Program for Semantic Information Retrieval, June 1964. (MAC-TR-2) AD-608-499. (In Semantic Info. Proc.)

*AI-TR-221 (Thesis) Teitelman, Warren, PILOT: A Step Toward Man-Computer Symbiosis, Sept. 1966. (MAC-TR-32) AD-638-446.

*AI-TR-222 (Thesis) Norton, Lewis M., ADEPT: A Heuristic Program for Proving Theorems of Group Theory, Oct. 1966. (MAC-TR-33) AD-645-660.

*AI-TR-223 (Thesis) Martin, William A., Symbolic Mathematical Laboratory, Jan. 1967. (MAC-TR-36) AD-657-283.

*AI-TR-224 (Thesis) Guzman-Arenas, Adolfo, Some Aspects of Pattern Recognition by Computer, Feb. 1967. (MAC-TR-37) AD-656-041.

*AI-TR-225 Forte, Allen, Syntax-Based Analytic Reading of Musical Scores, April 1967. (MAC-TR-39) AD-661-806.

*AI-TR-226 (Thesis) Moses, Joel, Symbolic Integration, Dec. 1967. (MAC-TR-47) AD-662-666.

*AI-TR-227 (Thesis) Charniak, Eugene, CARPS: A Program Which Solves Calculus Word Problems, July 1968. (MAC-TR-51) AD-673-670.

AI-TR-228 (Thesis) Guzman-Arenas, Adolfo, Computer Recognition of Three-Dimensional Objects in a Visual Scene, Dec. 1968. (MAC-TR-59) AD-692-200. ($4.25)

Methods are presented (1) to partition or decompose a visual scene into the bodies forming it; (2) to locate these bodies in three-dimensional space, by combining two scenes that make a stereoscopic pair; (3) to find the regions of zones of a visual scene that belong to its background; (4) to carry out the isolation of objects in (1) when the input has inaccuracies.

*AI-TR-229 (Thesis) Beyer, Wendell Terry, Recognition of Topological Invariants by Iterative Arrays, Oct. 1969. (MAC-TR-66) AD-699-502.

AI-TR-230 (Thesis) Griffith, Arnold K., Computer Recognition of Prismatic Solids, Aug. 1970. (MAC-TR-73) AD-711-763. ($3.25)

An investigation is made into the problem of constructing a model of the appearance to an optical input device of scenes consisting of plane-faced geometric solids. The goal is to study algorithms which find the real straight edges in the scene, taking into account smooth variations in intensity over faces of the solids, blurring of edges, and noise.

*AI-TR-231 (Thesis) Winston, Patrick H., Learning Structural Descriptions From Examples, Sept. 1970. (MAC-TR-76) AD-713-988.

AI-TR-232 (Thesis) Horn, Berthold K.P., Shape From Shading: A Method for Obtaining the Shape of a Smooth Opaque Object From One View, Nov. 1970. (MAC-TR-79) AD-717-336. ($3.25)

A method will be described for finding the shape of a smooth opaque object from a monocular image, given a knowledge of the surface photometry, the position of the light-source and certain auxiliary information to resolve ambiguities. This method is complementary to the use of stereoscopy which relies on matching up sharp detail and will fail on smooth objects. Until now the image processing of single views has been restricted to objects which can meaningfully be considered two-dimensional or bounded by plane surfaces.

*AI-TR-233 (Thesis) Banks, Edwin Roger, Information Processing and Transmission in Cellular Automata, Jan. 1971. (MAC-TR-81)

AI-TR-234 (Thesis) Krakauer, Lawrence J., Computer Analysis of Visual Properties of Curved Objects, May 1971. (MAC-TR-82) AD-723-647. ($2.25)

A method is presented for the visual analysis of objects by computer. It is particularly well suited for opaque objects with smoothly curved surfaces. The method extracts information about the object's surface properties, including measures of its specularity, texture, and regularity. It also aids in determining the object's shape.

*AI-TR-235 (Thesis) Winograd, Terry, Procedures as a Representation for Data in a Computer Program for Understanding Natural Language, February 1971. (MAC-TR-84) AD-721-399.

Now available in book form under the title: Understanding Natural Language, Terry Winograd, Academic Press (New York) 1972. In Britain and Europe: Edinburgh University Press, 1972.

*AI-TM-236 (Thesis) Jones, Thomas L., A Computer Model of Simple Forms of Learning, Jan. 1971. (MAC-TR-20). AD-720-337.

AI-TR-242 (Thesis) Smoliar, Stephen W., A Parallel Processing Model of Musical Structures, September 1971. (MAC-TR-91) AD-731-690. ($3.25)

EUTERPE is a real-time computer system for the modeling of musical structures. It provides a formalism wherein familiar concepts of musical analysis may be readily expressed. This is verified by its application to the analysis of a wide variety of conventional forms of music; Gregorian chant, Mediaeval polyphony, Bach counterpoint, and sonata form. It may be of further assistance in the real-time experiments in various techniques of thematic development. Finally, the system is endowed with sound synthesis apparatus with which the user may prepare tapes for musical performances.

*AI-TR-258 (Thesis) Hewitt, Carl, Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models In A Robot, April 1972. AD-744-620.

AI-TR-266 (Thesis) Charniak, Eugene, Toward A Model Of Children's Story Comprehension, December 1972. AD-755-232. ($4.25)

How does a person answer questions about children's stories? For example, consider "Janet wanted Jack's paints. She looked at the picture he was painting and said, 'Those paints make your picture look funny'". The question to ask is "Why did Janet say that?"

We propose a model which answers such questions by relating the story to background real world knowledge. The model tries to generate and answer important questions about the story as it goes along. Part of the information connected with a "concept" is the set of facts which might be relevant to stories which include the concept. When the concept occurs in the story these facts are "made available" in the sense that they can then be used to make deductions.

*AI-TR-271 (Thesis) Waltz, David L., Generating Semantic Descriptions From Drawings of Scenes With Shadows, November 1972. AD-754-080.

AI-TR-281 Winston, Patrick H., Editor, Progress In Vision And Robotics, May 1973. ($4.25)

This TR presents several selected vision flashes (informal working papers intended primarily to stimulate internal interaction among participants in the A.I. Lab's Vision and Robotics group). Many of them report highly tentative conclusions or incomplete work, while others deal with highly detailed accounts of local equipment and programs that lack general interest. The purpose of this report is to make the findings more readily available, but since they are not revised as presented here, readers should keep in mind that these are working papers.

AI-TR-283 (Thesis) Fahlman, Scott E., A Planning System For Robot Construction Tasks, May 1973. ($3.25)

This paper describes BUILD, a computer program which generates plans for buildings specified structures out of simple objects such as toy blocks. A powerful heuristic control structure enables BUILD to use a number of sophisticated construction techniques in its plans. Among the final design, pre-assembly of movable sub-structures on the table, and the use of extra blocks as temporary supports and counterweights in the course of the construction.

AI-TR-291 (Thesis) McDermott, Drew V., Assimilation of New Information by a Natural Language-Understanding System, February 1974. ($3.25)

This work describes a program, called TOPLE, which uses a procedural model of the world to understand simple declarative sentences. It accepts sentences in a modified predicate calculus symbolism, and uses plausible reasoning to visualize scenes, resolve ambiguous pronoun and noun phrase references, explain in events, and make conditional predications. Because it does plausible deduction, with tentative conclusions, it must contain a formalism for describing its reasons for its conclusions and what the alternaives are.

AI-TR-297 (Thesis) Sussman, Gerald J., A Computational Model of Skill Acquisition, August 1973. ($3.25)

This thesis confronts the nature of the process of learning an intellectual skill, the ability to solve problems efficiently in a particular domain of discourse. The investigation is synthetic; a computation performance model, HACKER, is displayed. HACKER is a computer problem-solving system whose performance improves with practice.